UNIVERSITÄT
KOBLENZ · LANDAU

Fachbereich 4: Informatik

Entwurf und Implementierung des Simulationsszenarios ‘CollaborativeWriting’ in EmIL-S

Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Informatik

vorgelegt von

Robin Emde

Erstgutachter:	Prof. Dr. Klaus G. Troitzsch Institut für Wirtschafts- und Verwaltungsinformatik
Zweitgutachter:	Ulf Lotzmann Institut für Wirtschafts- und Verwaltungsinformatik

Koblenz, im September 2009

Erklärung

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

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☒	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☒	☐

. . . . . .

(Ort, Datum) (Unterschrift)

Am Anfang jeder Forschung steht das Staunen.
Plötzlich fällt einem etwas auf.
Wolfgang Wickler (*1931), dt. Verhaltensforscher u. Zoologe

Kurzfassung

Theorien, Szenarien, Vermutungen und Annahmen lassen sich in den Sozialwissenschaften seit geraumer Zeit mittels computergestützter Simulation analysieren, erhärten oder entkräften. Im Feld der Simulationsumgebungen gibt es eine Vielzahl unterschiedlicher Softwarelösungen. Das Projekt EmIL hat zum Ziel eine Simulationsumgebung zu erschaffen, die einen Schritt weiter geht als die bisher etablierten Lösungen. So soll in dieser Simulationsumgebung namens EmIL-S die Möglichkeit geschaffen werden Normentstehung zu analysieren. Dies wird durch eine Auftrennung der Simulationsumgebung in verschiedene Bereiche erreicht. Diese Diplomarbeit beschreibt die Konzepte die hinter diesem Projekt stehen und stellt einen lauffähigen Prototyp für diese Simulationsumgebung zur Verfügung. Die Erstellung dieses Prototypens, das hierfür zugrunde liegende Szenario und alle verwendeten Komponenten werden detailliert in dieser Arbeit beschrieben. Auch werden die mittels dieses Prototypens gewonnen Erkenntnisse dargelegt und analysiert. Im Zuge dieser Beschreibungen werden konzeptionelle sowie technische Fehler an der Simulationsumgebung aufgezeigt und Lösungs- bzw. Verbesserungsvorschläge dargelegt. Diese Arbeit kann auch als Leitfaden für die Erstellung von Simulationen mittels EmIL-S hilfreich sein.

Abstract

For a significant length of time, computer simulation programs have been the preferred tools of sociologists to analyse – thus confirm or refute – sociological theories, scenarios, presumptions and assumptions. Many different simulation platforms for various domains exist. The EmIL project aims to create a new simulation platform specifically to analyse norm emergence. This platform is divided into sections, thereby allowing the analysis of individual norm emergence: this is not currently possible, thus EmIL-S may represent a significant improvement over current simulation platforms. This diploma thesis describes the concepts behind the project and provides a prototype for the simulation program EmIL-S. The creation of the prototype and the underlying development process are described in detail. The experience gained during this process is analysed and described. In the course of this analysis, the malfunctions and errors within EmIL-S are explained; the solutions to these malfunctions and errors are noted, along with other proposed improvements. This diploma thesis can also be used as a helpful manual to build simulations within the simulation platform.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

In unserer heutigen Gesellschaft ist es für Politik und Wirtschaft, aber auch für jeden einzelnen Bürger immer wichtiger geworden, die eigene Umwelt und die Mitmenschen zu verstehen. In Zeiten von Wertewandel, schnelllebiger Gesellschaft und dem Streben, besser zu sein als die Konkurrenten, wird die Kalkulierbarkeit menschlichen Verhaltens immer mehr zu einem wichtigen Wettbewerbsvorteil. Dies schließt sowohl die Politik, in der Parteien um die Gunst des Wahlvolkes buhlen, als auch den einzelnen Menschen, der ein Leben in Sicherheit und in Frieden und Freiheit anstrebt, mit ein. Diese und viele andere Gruppen haben ein großes Interesse daran, Menschen und ihr Verhalten besser zu verstehen, sogar meist den innigen Wunsch, dieses Verhalten vorhersagbar und damit weniger überraschend zu machen. Das Objekt „Mensch“ und sein Verhalten wurde vielfach und vielschichtig erforscht. Durch die komplexe Struktur seines Gehirns und die schiere Unmöglichkeit, die sich darin abspielenden kognitiven Prozesse verständlich abzubilden, bleiben jedoch viele Fragen weiter ungeklärt. Noch komplexer sind daher die Prozesse, die sich im Zusammenleben von Menschen ergeben. So ist bis heute strittig, wie innerhalb von Gesellschaften Normen entstehen und wieder verschwinden, welche Faktoren diese Normentwicklung auslösen und ob überhaupt und wenn ja, wie Normen in Gesellschaften propagiert und adaptiert werden. Ob es sich hierbei um einen Prozess handelt, der von außen auf den Menschen einwirkt oder ob es sich um einen Prozess von innen heraus handelt. Diesen Fragestellungen widmet sich das Projekt EmIL aus unterschiedlichen Perspektiven. So beschäftigt sich eine Projektgruppe mit den Fragen aus dem soziologischen Bereich, eine andere wiederum mit den theoretischen Grundlagen. Eine dieser Gruppen,

die der Universität Koblenz-Landau, verfolgt das Ziel, einen Simulator zu erstellen, mit dem Normentwicklung innerhalb von künstlichen Gesellschaften simuliert werden kann. Dieser Simulator soll es ermöglichen, verschiedenste Theorien innerhalb beobachtbarer künstlicher Gesellschaften, welche sich aus autonom agierenden Agenten zusammen setzen, zu testen und weiter zu entwickeln. Mit Hilfe der Informatik ist es somit möglich, Rückschlüsse auf die reale Welt zu ziehen und neue Theorien über deren Struktur zu entwickeln. In diesem Kontext ist die hier vorliegende Arbeit entstanden.

1.2 Gang der Untersuchung

Die theoretischen Grundlagen des Themas Normentwicklung wurden in diversen Publikationen z.B. [Hor01, JS93, Sar05, Wri05, und anderen] beschrieben und bilden ein solides Fundament für die Arbeit des Projektes EmIL. Im Rahmen dieses Projektes wird die Erstellung einer Multiagentensimulationsumgebung angestrebt, in der die Agenten autonom agieren und ihre kognitiven Reaktionen implementiert sowie später analysiert werden können.
Die Arbeitsgruppe der Universität Koblenz-Landau beschäftigt sich innerhalb dieses Projektes mit der Erstellung einer Softwarekomponente, welche die Erforschung von Normentstehung mittels Simulation möglich machen soll. Der Name der Softwarekomponente ist EmIL-S. Sie liegt in einer benutzbaren, aber noch nicht vollendeten Version vor und wird stetig weiterentwickelt. EmIL-S stellt jedoch nur einen von drei benötigten Teilen dar. Hinzu kommt noch eine Komponente mit der es möglich ist, eine Vielzahl von Simulationsläufen automatisiert ablaufen zu lassen und die hierbei gewonnenen Daten automatisiert auswerten zu können. Diese Komponente wird von einer Gruppe aus Ungarn entwickelt und nennt sich „The Model Exploration Module“ (MEME). Die dritte Komponente, die zusammen mit den beiden bereits beschriebenen die Simulationsumgebung vervollständigt, ist ein Simulationswerkzeug mit dem es möglich ist Methoden, Umgebungen, Wahrnehmungsbereiche und alle übrigen für die Simulation notwendigen Komponenten zu implementieren. Die beschriebenen Komponenten und ihre Beziehungen zueinander werden in Kapitel 2.1 umfassend erläutert.
Gegenstand dieser Arbeit ist es, ein vorhandenes Szenario derart zu implementieren, dass es in Zusammenarbeit mit EmIL-S in einer lauffähigen Simulation vorliegt. Das zu realisierende Szenario liegt sowohl in Form einer detaillierten Szenariobeschreibung als auch

in einer, mit Hilfe der Modellierungsumgebung NetLogo[1] implementierten, ausführbaren Fassung vor.
Der Gang der Untersuchung präsentiert sich wie folgt: Zuerst wird auf die einzelnen Teilaspekte einleitend eingegangen. Im Anschluss wird die Analyse des vorhandenen Szenarios vorgestellt. Darauf aufbauend wird ein konzeptioneller Entwurf entwickelt. Nach dieser ersten Phase soll eine Konfiguration für EmIL-S in Form einer XML-Datei erstellt werden. Die vorhandene XML-Sprache zur Beschreibung der Abhängigkeiten innerhalb des abzubildenden Modells wird während dieses Prozesses unter Umständen zu erweitern oder abzuändern sein, um diesen Schritt bewerkstelligen zu können. In einer darauffolgenden Phase soll das Simulationswerkzeug, das im Zusammenspiel mit EmIL-S und einer zu erstellenden Konfiguration in der Simulation Verwendung finden wird, gefunden und um die notwendigen Funktionen erweitert werden. Hierbei ist eine Programmierung der fehlenden Komponenten in Java angestrebt. Die Implementation der Software wird in Kapitel 4 detailliert beschrieben. Die Kombination von Konfiguration und Implementation wird in einer Testphase, in der mehrere Simulationsdurchläufe verarbeitet werden, auf Fehler untersucht. In der letzten Phase sollen gewonnene Ergebnisse aus Simulationsläufen der ursprünglichen Implementation mit der in dieser Arbeit erstellten neuen Implementation verglichen werden. Eventuelle notwendige Änderungen an der Implementierung werden nach diesem Vergleich aufgezeigt und beschrieben. Abschließend folgen ein Ausblick und ein Fazit.
Übergreifend soll diese Arbeit in jedem Schritt dabei helfen, EmIL-S weiterzuentwickeln, indem sie Unzulänglichkeiten, notwendige Erweiterungen oder sinnvolle Ergänzungen aufzeigt, die sich während ihrer Erstellung ergeben.

[1] Bei NetLogo handelt es sich um eine von Uri Wilensky im Jahre 1999 eigens für die Erstellung von Multiagentensystemen entwickelte Programmiersprache, welche zusätzlich noch um eine Modellierungsumgebung ergänzt wurde. Vgl. [Wil99].

Kapitel 2

Grundlagen

2.1 Das EmIL-Forschungsvorhaben

Das Projekt EmIL mit der Projektbezeichnung: „Emergence In the Loop: Simulating the two-way dynamics of norm innovation“ beschäftigt sich mit der Frage, wie Normen in Gesellschaften entstehen. Da dieses Thema sehr vielschichtig ist, wurden unterschiedliche Gruppen gebildet, die sich abhängig von ihrer Fachrichtung aus unterschiedlichen Richtungen diesem Thema nähern. Hauptziel ist es, Design-Strategien zu entwickeln, die helfen, die Dynamik in sozialen Systemen abzubilden und zu erklären. Diese Strategien sollen sowohl die Interaktionen zwischen Individuen als auch die Verknüpfung mit höheren Ebenen und das Herauskristallisieren von Normen und deren Entwicklung abbilden. Das Projekt ist auf sechs Jahre ausgelegt und wird von der EU gefördert. Vgl. [EmI09, Tro09a]. Eine Reihe von Forschungseinrichtungen beteiligen sich an diesem Projekt [EmI09]:

- Institute of Cognitive Science and Technology, National Research Council CNR-ISTC Italy
- University of Bayreuth, Dept. of Philosophy UBT Germany
- University of Surrey, Centre for Research on Social Simulation UNIS United Kingdom
- Universität Koblenz-Landau KL Germany
- Manchester Metropolitan University, Centre for Policy Modelling MMU United Kingdom

- AITIA International Informatics Inc. AITIA Hungary

Die drei Hauptziele definieren sich wie folgt. Vgl. [Tro09a]:

1. Erlangen von Verständnis und die Beherrschung von komplexen sozialen Systemen durch autonom agierende Agenten.
2. Verstehen, wie neue Konventionen und Normen entstehen und in diesen Systemen Verbreitung finden.
3. Das Studium der „Normen“-Entstehung mithilfe von agentenbasierter Simulation.

Das technische Hauptaugenmerk liegt auf der Erstellung einer Agenten basierten Simulationsumgebung, mit der es möglich sein soll, Normentwicklung zu erforschen. Vgl. [EmI09]. Hierzu wird an der Universität Koblenz-Landau diese Simulationsumgebung geplant und entwickelt. Die Hauptaufgabe der Koblenzer Gruppe ist die Erstellung der Softwarekomponente EmIL-S. Auch wird hier dafür Sorge getragen, dass alle Komponenten zu einer lauffähigen Simulationsumgebung zusammen geführt werden können. Wie eingangs beschrieben, besteht die Simulationsumgebung aus drei Komponenten: EmIL-S, MEME und einem Simulationswerkzeug. Die drei Komponenten und ihre Beziehungen zueinander werden nachfolgend erläutert und sind in Schaubild 2.1 dargestellt.

2.1.1 EmIL-S

Die Bezeichnung EmIL-S ist ein Akronym und steht für „EMergence In the Loop“, hierbei präzisiert das angehängte „S“, dass es sich um die Softwarekomponente des Projektes handelt. Diese soll es ermöglichen, Normentstehung verständlich zu machen. Hierfür übernimmt EmIL-S alle kognitiven Entscheidungen, die ein Agent während der Simulation zu treffen hat. EmIL-S speichert die bereits getätigten Entscheidungen jedes einzelnen Agenten und ermittelt mit Hilfe von Lernalgorithmen die zu tätigenden Aktionen für jeden einzelnen Agenten in Abhängigkeit von bereits gemachten Erfahrungen. Hierdurch wird es erstmals möglich, die Entscheidungsfindung an zentraler Stelle zu erfassen und auszuwerten. Die möglichen Aktionen, die von den Agenten ausgeführt werden können, werden für jedes Simulationsmodell durch eine Konfigurationsdatei für EmIL-S definiert und mit initialen Wahrscheinlichkeiten versehen. Jede Aktion wird innerhalb dieser Konfigurationsdatei einem oder mehreren Ereignissen zugeordnet. Daraus ergibt sich eine „event-action tree“ genannte Baumstruktur.

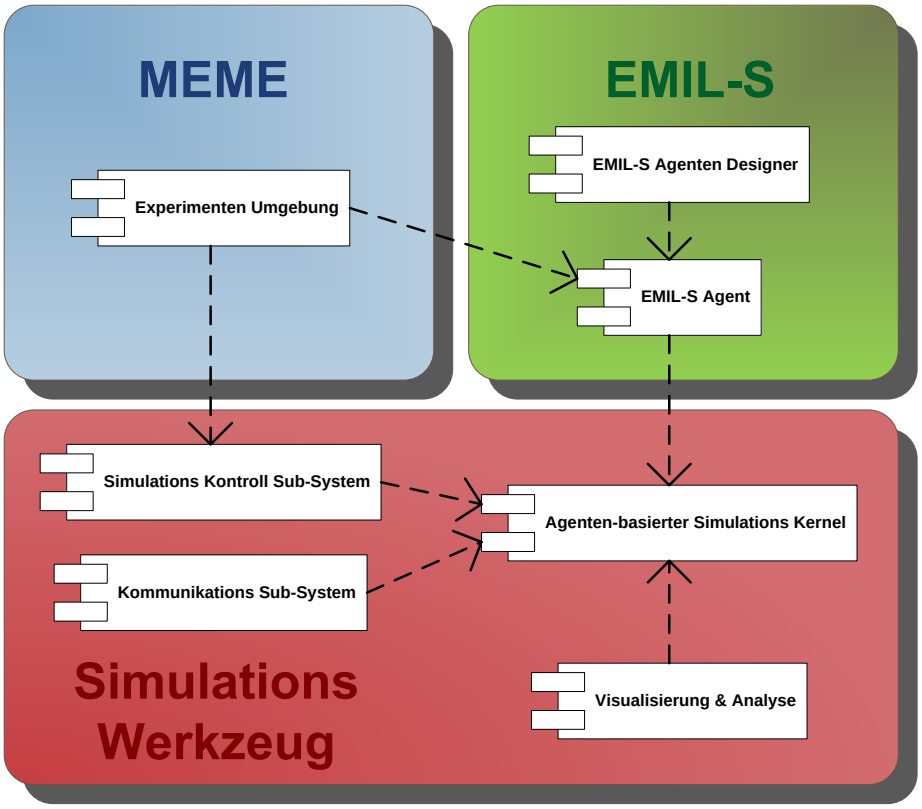

Abbildung 2.1: Simulationssystem. Vgl. [LM09b, Seite 5]

In Abbildung 2.2 ist hierfür ein Beispiel skizziert. In dieser Abbildung wird ein Modell vorausgesetzt, welches zwei Arten von Agenten beinhaltet. Die eine Art bewegt sich als Fußgänger, die andere als Fahrer eines Fahrzeugs durch eine virtuelle Umgebung. Das mögliche Verhalten der Agenten wird nun durch Entscheidungen bestimmt, die von EmIL-S getroffen werden. Hierfür werden in der Konfiguration für EmIL-S eine Reihe „events“ und zugehörige „actions“ definiert. So ist zum Beispiel ein event wie folgt definiert: Ein sich fortbewegender Fahrzeugführer macht einen Fußgänger in seinem Weg aus. Der Fahrer hat nun mehrere Möglichkeiten auf dieses event zu reagieren. So ist es ihm möglich, mit einer bestimmten Wahrscheinlichkeit die „action-group (G1)“ auszuwählen und darin eine der Aktionen „Beschleunigen (a10)“, „Geschwindigkeit drosseln (a11)“ oder das „Fahrzeug vollständig anhalten (a12)“ zu wählen. Zeitgleich ist es ihm möglich, zusätzlich noch „action-group (G2)“ auszuwählen und darin wahlweise die Aktionen: „Hupen (a20)“ oder „Rufen (a21)“ auszuwählen. Zu beachten ist hierbei, dass es

möglich ist, sowohl nur eine als auch beide Gruppen zusammen anzuwählen. Innerhalb einer Gruppe ist es jedoch nur möglich, eine einzige Aktion auszuwählen. Daher darf die Summe der Wahrscheinlichkeitswerte an den Aktionen 1.0 nicht übersteigen. Bei den Wahrscheinlichkeiten an den Aktions-Gruppen jedoch dürfen beliebige Wahrscheinlichkeiten kleiner oder gleich 1.0 annotiert werden. Durch diese event-action trees lässt sich

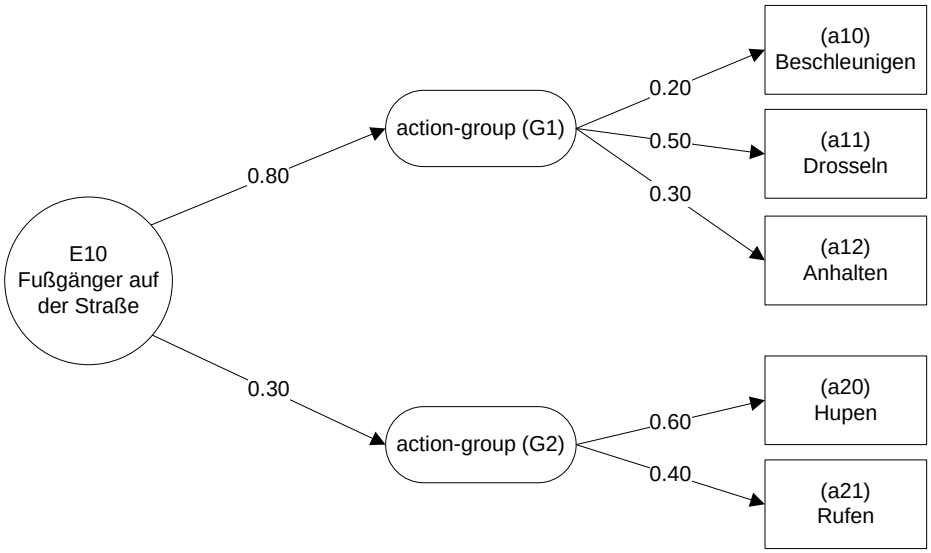

Abbildung 2.2: Event-action tree. Vgl. [LM09b, Seite 3]

beliebiges Verhalten von Agenten in einer Simulation abbilden. Die initial in der Konfigurationsdatei vorgegebenen Wahrscheinlichkeiten an den Kanten des Baumes werden von EmIL-S je nach Lernalgorithmus und negativer bzw. positiver Rückkopplung variiert. Auf diese Weise ergeben sich Lernverhalten und Verhaltensstrukturen. Hat ein Agent eine Entscheidung zu treffen, so wird für diese Entscheidung ein event in EmIL-S ausgelöst. Alle eingetroffenen events, welche sich auf die Agentenumgebung beziehen sowie alle getroffenen Entscheidungen werden von EmIL-S in einem sogenannten „event board“ abgespeichert. Durch das Vorhalten dieser Historie ist es EmIL-S möglich, anstehende Entscheidungen zu bestimmten Ereignissen in Abhängigkeit von gemachten Erfahrungen zu treffen. Das event board beinhaltet folgende Elemente. Vgl.[LMT08, Seiten 4-5]:

- Den Zeitpunkt, zu dem ein event eingetroffen ist.
- Das event selbst.
- Den aktuellen Zustand des Agenten.

- Den zugehörigen event-action tree.

Zusätzlich zu diesen vier Elementen ist es EmIL-S möglich, zu jedem beliebigen Zeitpunkt einen „classifier“ zu einer Teilmenge des event boards zu berechnen. Dieser soll event boards bzw. Teilmengen daraus miteinander vergleichbar machen. Er ermöglicht es, die Ähnlichkeit von event boards oder Teilmengen zueinander zu bestimmen. Erst durch diesen classifier wird es möglich, zu erkennen, ob Agenten ähnliches Verhalten aufweisen. Ein Beispiel für ein event board inklusive eines classifiers findet sich in Abbildung 2.3. Hat eine Mehrzahl der Agenten eine ähnliche Verhaltensstruktur für bestimmte Situatio-

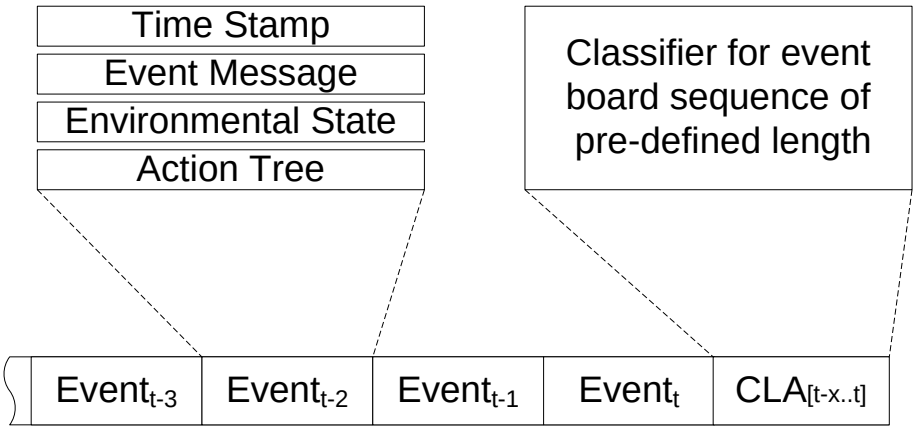

Abbildung 2.3: Event board. Vgl. [LM09b, Seite 2]

nen entwickelt, so kann man dies als eine Norm betrachten. Hierbei ist zu beachten, dass die Agenten in der Simulation miteinander kommunizieren und nicht einfach unabhängig für sich alleine agieren. Denn erst durch die Kommunikation und das dadurch entstehende Feedback anderer Agenten lassen sich Sozialstrukturen simulieren. Das Verhalten von Agenten wird maßgeblich von EmIL-S durch getroffene Entscheidungen bestimmt. Diese Entscheidungen werden in Form von Regeln in einem sogenannten „normative frame“ festgehalten. Ein normative frame beinhaltet folgende Elemente [LM09b, Seite 2]:

- Einen classifier, welcher diejenige Teilmenge des event boards abbildet, welche die Regel bestimmt.
- Die events der zugehörigen event board-Teilmenge.
- Die erstellte Regel, welche aus zusammengeführten event-action trees gebildet wurde.

- Den Zustand der Regel, der durch eine Einordnung in bestimmte Kategorien definiert wird. So kann eine Regel folgenden Kategorien zugerechnet werden:

  - *Regularity*, eine Regel wird zwar häufig angewendet, gehört jedoch nicht einer der nachfolgenden Kategorien an.
  - *Normative belief*, eine Regel wird als eine Art Norm interpretiert, ist jedoch noch nicht der Kategorie Norm zuzurechnen.
  - *Norm*, die Regel wird als eine Norm wahrgenommen.

  Zusätzlich enthält der Zustand noch einen Typ. Bei diesem kann es sich um eine Erlaubnis, eine Verpflichtung oder ein Verbot handeln.

- Die „valuation history“. Diese enthält statistische Werte, welche sich aus den gemachten Bewertungen errechnen, die zu dieser Regel abgegeben wurden.

Regeln werden gebildet, indem eine Teilmenge des event boards in eine Regel überführt wird. Dies geschieht, wenn ein sogenanntes „norm-invocation event“ ausgelöst wurde. Hierbei handelt es sich um ein besonderes event, welches das Erstellen von Regeln, beziehungsweise das Anpassen dieser Regeln, explizit anstößt. Ist zum Zeitpunkt des Eintreffens eines events noch keine Regel für dieses event im normative frame vorhanden, so wird innerhalb von EmIL-S eine norm-invocation ausgelöst. Ist bereits eine passende Regel vorhanden, so wird das aktuelle event der valuation history hinzugefügt und ein Lernalgorithmus angestoßen. Dieser passt dann lediglich die Wahrscheinlichkeiten der Regel an. Norm-invocation events stellen eine besondere Art von Nachrichten bzw. Ereignissen dar. Durch sie wird ein Lernprozess innerhalb von EmIL-S ausgelöst. Bei norm-invocation events handelt es sich entweder um externe norm-invocations, zum Beispiel bei Rückmeldungen von anderen Agenten zu einem bestimmten Ereignis oder aber um von einem Agenten selbst gemachte Beobachtungen. Zusätzlich hierzu gibt es noch interne norm-invocations. Diese werden durch Prozesse innerhalb von EmIL-S angestoßen und unterliegen keiner direkten Steuerung von außerhalb. Im Verlauf der Simulation unterliegt der normative frame und damit alle Regeln einem ständigen Wandel. So werden zur Laufzeit Regeln variiert und hinzugefügt. Neue Regeln werden durch Zusammenführen von event-action trees gebildet. In Abbildung 2.4 ist ein solcher normative frame inklusive Feldern dargestellt. In EmIL-S ist auch eine Komponente enthalten, mit der es anhand einer grafischen Benutzeroberfläche möglich sein soll, die oben beschriebenen event-action trees zu erstellen und damit die vollständige Konfiguration anzufertigen.

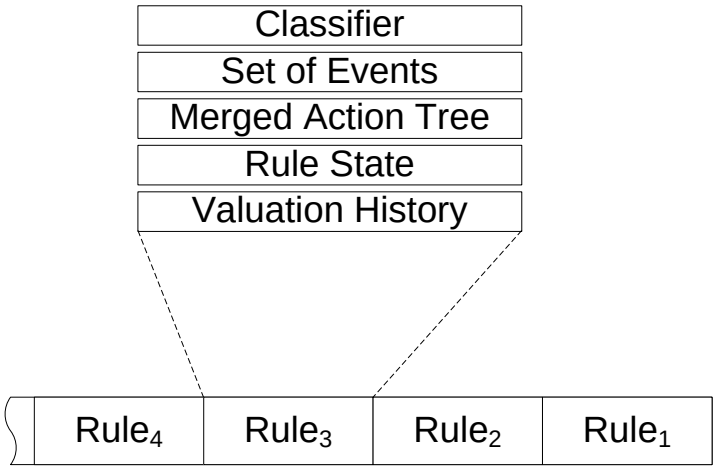

Abbildung 2.4: Normative frame. Vgl. [LM09b, Seite 3]

Auch soll es innerhalb dieser, als Agent-Designer bezeichneten Komponente, zu einem späteren Zeitpunkt möglich sein, die Änderungen der Wahrscheinlichkeiten zu beobachten. Die Benutzeroberfläche des Agent-Designers ist in Abbildung 2.5 zu sehen. Innerhalb des Agent-Designers lassen sich die Agententypen sowie die initialen event-action trees definieren und anzeigen. Innerhalb des Designers ist in der Mitte ein Bereich vorgehalten, in dem während der Laufzeit eintreffende Ereignisse abgebildet werden sollen. Auch können im unteren Bereich Teile des normative boards dargestellt werden. Durch diese Werkzeuge wird es möglich, die Simulation zur Laufzeit zu beobachten.

2.1.2 MEME

Forschung mittels Simulation bedarf meist einer großen Anzahl an Simulationsläufen mit unterschiedlichen Parameterbereichen und einer systematischen Auswertung der gewonnenen Daten. Diese Tätigkeiten lassen sich jedoch durch MEME zum größten Teil automatisieren. Das Akronym MEME steht für: „The Model Exploration Module“. Es ist Teil eines großen Softwaresystems namens „mass“, welches zur Erstellung von Simulationen genutzt werden kann. Mass und die Komponente MEME werden von „AITIA International Inc.“ entwickelt. Die Komponente MEME kann jedoch auch losgelöst von „mass“ genutzt werden und ist daher für das Projekt EmIL zweckdienlich. So ist es mit ihr möglich, eine Reihe von Simulationsläufen zu planen und von MEME durchführen zu lassen. Hierbei kümmert sich MEME um den Ablauf der Simulationsläufe, die Variation der Parameter und die Aufbereitung und Speicherung der gewonnenen Daten. Es ist mit

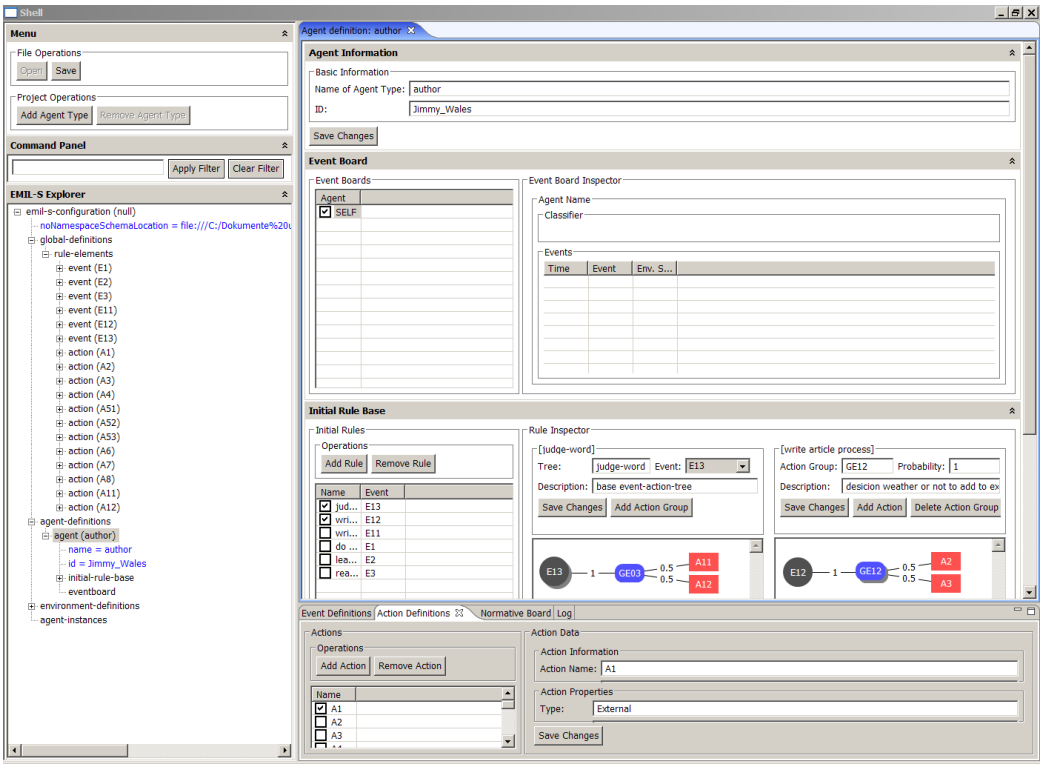

Abbildung 2.5: Agent Designer

ihr sogar möglich, Parameterbereiche in Abhängigkeit von bereits gewonnenen Daten zu definieren. Auch kann MEME die Simulation verteilt auf einem Rechencluster oder einer Vielzahl voneinander unabhängigen Rechnern, die zu einem virtuellen Rechencluster verbunden werden, ablaufen lassen. Hierdurch werden die meist langwierigen Simulationsläufe beschleunigt. Die von MEME aufbereiteten und gespeicherten Daten lassen sich nach Abschluss der Simulationsläufe mit spezieller Software auswerten. Dies kann je nach Forschungsansatz eine aufgestellte Theorie bestätigen, völlig neue Erkenntnisse erbringen oder aber zur Überarbeitung einer Theorie führen. Vgl. [AIT09].

2.1.3 Das Simulationswerkzeug

Die dritte Komponente innerhalb der zu erstellenden Simulationsumgebung ist das Simulationswerkzeug. Hierbei handelt es sich um ein Werkzeug oder Framework, das es dem Benutzer ermöglichen soll, alle notwendigen Strukturen, Methoden und Eigenschaften der Simulation zu entwickeln und zu implementieren. Hierfür muss das Framework Möglichkeiten zur Steuerung des Simulationsablaufes anbieten, auch Visualisierungs- und Analy-

semöglichkeiten sollten vorhanden sein. Es existiert eine Vielzahl von Simulationswerkzeugen, die sich unterschiedlicher Beliebtheit erfreuen und teils auf bestimmte Bereiche spezialisiert sind, zum Beispiel Swarm, Jade, Repast oder Trass, um nur einige zu nennen. Vgl. [Swa07, Tel00, Rep09].

Um mit den drei beschriebenen Komponenten ein Simulationsmodell umsetzen zu können, sind mehrere Schritte erforderlich. So ist es notwendig, innerhalb des Simulationswerkzeuges alle Methoden zu implementieren, die für die Simulation notwendig sind. Hierzu zählen unter anderem Tätigkeiten der Agenten, Eigenschaftsänderungen, Änderungen der Umwelt, Kommunikation mit EmIL-S sowie Kommunikation zwischen Agenten und ihr sonstiges Verhalten. Auch muss eine initiale Konfiguration für die Komponente EmIL-S gefunden werden. Sind diese beiden Schritte vollzogen, so kann mittels MEME eine Reihe von Simulationsdurchläufen geplant und ausgewertet werden.
Diese Arbeit soll eine Implementierung eines Modells in der beschriebenen Art und Weise hervorbringen und damit zeigen, dass die bisher erstellten und konzipierten Komponenten in der gewünschten Art miteinander zusammenwirken können.

2.2 Entwicklung und aktueller Stand

EmIL-S liegt in einer lauffähigen Version vor. Die Komponente, die innerhalb der Simulationsumgebung die kognitiven Entscheidungen der Agenten trifft, funktioniert und beinhaltet bisher einen simplen aber zweckdienlichen Lernalgorithmus. Ein zweiter Teil von EmIL-S, der Agenten-Designer, mit dem es möglich sein soll, die initiale Konfiguration für EmIL-S mittels einer komfortablen grafischen Benutzeroberfläche zu erstellen, liegt in einer beinahe fertigen Version vor. An dieser Stelle wird jedoch noch an weiteren Funktionalitäten gearbeitet. So soll es in Zukunft möglich sein, die Entscheidungsfindung in Echtzeit, also während der Simulation, beobachten zu können. Hierfür wird der Agent-Designer erweitert, um die Konfiguration einzelner Agenten in Form von event-action trees beobachten zu können. Schnittstellen zu Trass und Repast wurden definiert und implementiert. EmIL-S ist also bereits mit diesen beiden Simulationswerkzeugen nutzbar. Für die Zukunft ist die kontinuierliche Erweiterung von EmIL-S geplant. Hierbei stehen zuerst die Entwicklung und Implementierung diverser unterschiedlicher Lernalgorithmen im Vordergrund. Nachfolgend sollen die Nachrichten, die EmIL-S empfangen kann, um weitere Funktionen erweitert werden. Auch sollen für EmIL-S weitere Schnittstellen für zusätzliche Simulationswerkzeuge entwickelt werden, dies jedoch abhängig davon, ob an-

dere Simulationswerkzeuge Verwendung finden sollen und ein begründetes Interesse für die Nutzung dieser Simulationswerkzeuge besteht.

Kapitel 3

Entwurf

Diese Arbeit soll dazu dienen, die bereits beschriebenen Programme und Konzepte zu prüfen. Hierfür wird ein Szenario implementiert, das die beschriebenen Komponenten nutzt und in einer lauffähigen Simulation zur Ausführung kommen soll. Im Zuge der Konzeptionierung und der Implementierung dieses Szenarios sollen Unzulänglichkeiten, Fehler und sinnvolle Ergänzungen an den Komponenten aufgezeigt werden. Dies soll zeigen, ob die entwickelten Komponenten in der ihnen zugedachten Art und Weise Verwendung finden können und ob es Möglichkeiten zur Verbesserung gibt.
Hierzu ist es zuerst notwendig, das vorgegebene Szenario zu analysieren. Dieses liegt sowohl in einer Szenariobeschreibung als auch in Form eines ausführbaren NetLogo-Programms vor. Anschließend wird das Szenario neu implementiert, wobei die beschriebenen Komponenten Verwendung finden sollen. Die Implementierung untergliedert sich in verschiedene Phasen. So muss erst ein konzeptioneller Entwurf erstellt, dann eine Konfiguration für EmIL-S gefunden und darauf aufbauend das Programm mit einem Simulationswerkzeug implementiert werden. Abschließend erfolgt eine Test- und Evaluierungsphase.

3.1 Das Ursprungsszenario

3.1.1 Beschreibung des Wikipedia-Szenarios

Das Wikipedia-Szenario, auch ‘CollaborativeWriting’ genannt, beschreibt eine fiktive Wikipedia, in der die Agenten Artikel schreiben und diese untereinander kommentieren und diskutieren können. Wie in der realen Welt soll es hier möglich sein, Normen zu ent-

wickeln und zu festigen. Hierfür müssen die Agenten in die Lage versetzt werden, diese Tätigkeiten auszuüben. Im vorliegenden Szenario erfolgt die Diskussion mittels Nachrichten, die jedoch nicht wie im normalen Leben frei sind, sondern wohl definiert. So enthält eine Nachricht stets einen Absender, einen Empfänger, einen Modal und einen Nachrichtentext. Der Empfänger kann sowohl ein aktiver Agent als auch ein passiver Beobachter sein. Vgl. [EmI08a, Seiten 4-5]. Der Modal kann einer der folgenden Typen sein. Vgl. [EmI08a, Seite 5]:

- eine beliebige Aussage (assertion) (A)
- eine Handlungsweise (behaviour) (B)
- eine Anfrage (request) (R)
- eine Notwendigkeit (deontic) (D)
- eine Bestätigung (validation) (V)
- eine Sanktion (sanction) (S)

Mit dieser Spezifikation lassen sich eine Vielzahl möglicher Nachrichten erzeugen. Von der allgemeinen nicht zwangsweise richtigen Aussage über mögliche Handlungsempfehlungen, wie zum Beispiel: „Es sollte so verfahren werden.“ bis hin zu Sanktionen, wobei letztere nicht nur negativ sondern auch positiv ausfallen und mit einer Gewichtung versehen werden können. Via Bestätigungen lassen sich gemachte Aussagen von anderen Quellen bekräftigen. Auch sind Anfragen zu bestimmten Themen möglich. Ein einfacher Wikipediaartikel lässt sich hiermit auch erzeugen. Dies ist dann eine einfache Aussage, welche als Modal den Typ „A“ enthält, als Absender die Identität des Autors, als Empfänger alle Agenten und als Inhalt eine beliebige Zeichenkette. Vgl. [EmI08a, Seite 5].

Es obliegt dem jeweiligen Ersteller einer Simulation zu entscheiden, ob die Agenten aus einer vorgegebenen und feststehenden Reihe von Nachrichten auswählen oder, ob die Agenten mittels Algorithmen Nachrichten selbst erzeugen und versenden können. Im hier beschriebenen Szenario können die Agenten Artikel verfassen und in eine fiktive Wikipedia einfügen. Die Artikel bestehen aus Wörtern, welche aus Buchstaben gebildet werden. Die Wortlänge kann beliebig festgelegt werden. Im konkreten Szenario werden Wörter nur aus den Buchstaben „*aeibklsw* “ gebildet, wobei das Leerzeichen dazugehört und jeweils das Wortende bildet. Jedoch ist es durchaus möglich, die Art der Wortbildung zu variieren und eine andere, als die hier beschriebene Sprache, zu definieren. Für das

vorliegende Szenario erscheint diese Definition jedoch ausreichend. Wörter werden von den Agenten mittels der vorgegebenen Buchstaben erzeugt. Es wird eine Regelmenge für die Erstellung von Wörtern zugrunde gelegt, jedoch nicht festgeschrieben. Die Wortbildung soll im Verlauf der Simulation anhand von Normbildung entwickelt werden. Die angesprochene Regelmenge findet in diesem Szenario Verwendung bei der Generierung von Sanktionen, die gegen Agenten ausgesprochen werden, welche sich nicht an diese Regelmenge gehalten haben. Das erste Wort jedes Artikels wird als Schlüsselwort definiert. Vgl. [EmI08a, Seite 5].

Mit Hilfe dieser Definition einer Sprache wird es möglich, die Agenten in die Lage zu versetzten, bestimmte Schlüsselwörter innerhalb der Texte zu suchen. Diese können dann als Verweise (Links) interpretiert werden und somit eine Gewichtung der Artikel ermöglichen. Auch ist es so möglich, dass Agenten Artikel vergleichen. Hierbei können sich verschiedene Aktionen ergeben. Sind zwei Artikel sehr ähnlich, so könnte der Agent dies als ein Plagiat interpretieren und beide Autoren darüber informieren. Dies wäre dann wiederum eine reine Feststellung. Oder aber der Agent könnte den Autor des neueren Artikels dazu auffordern, diesen wieder zu entfernen. Auch Sanktionen sind an dieser Stelle möglich. Denkbar ist auch, dass ein Agent einen Artikel findet, der zwar Ähnlichkeiten zu einem Anderen aufweist, aber kürzer als dieser ist. Dann wäre es möglich, beide Artikel zu einem zusammen zu führen. Auch das Suchen nach Abweichungen von der beschriebenen Grammatik ist möglich und führt zu definierten Maßnahmen. Vgl. [EmI08a, Seiten 5-6].

Die Agenten können Artikel lesen, selbst schreiben, kommentieren sowie ergänzen oder verändern. Im Wikipedia-Szenario sollen die Agenten auch mittels der gelesenen und empfangenen Nachrichten Erkenntnisse über andere Agenten gewinnen und daraus Rückschlüsse ziehen können. Das beschriebene Szenario lässt sich noch um eine Vielzahl von Möglichkeiten erweitern. Es wurde sehr nah an der Realität modelliert. So ist es auch in einer realen Wikipedia möglich, Artikel zu verfassen und andere Artikel zu lesen, zu kommentieren, zu löschen oder zu ergänzen. In einer realen Wikipedia können auch teils angeregte Diskussionen zum Inhalt oder zur Qualität eines Artikel entstehen und so zum Ausbilden von Normen führen. Vgl. [EmI08a, Seiten 5-6].

3.1.2 Beschreibung der Wikipedia-NetLogo-Simulation

Die von Troitzsch im März 2008 erstellte NetLogo-Simulation setzt das beschriebene Wikipedia-Szenario in einer ausführbaren Simulation um.
Aufgrund der Beschränkungen von NetLogo konnte jedoch nicht auf alle Aspekte eingegangen werden. Die Implementierung in NetLogo hat die generelle Umsetzbarkeit des Szenarios gezeigt und sollte als anschauliches Beispiel und Basis für weitere Entwicklungen dienen.

Umgesetzte Aspekte

In der NetLogo-Wikipedia-Simulation haben die Agenten die Möglichkeit, verschiedene Aktionen auszuüben. Diese Aktionen gliedern sich wie folgt. Vgl. [EmI08a, Seiten 7-8]:

- Ein Agent kann einen Artikel schreiben (A1) und diesen dann in die Wikipedia einbringen (A2) oder aber seinen Artikel einem Anderen bereits Bestehenden hinzufügen (A3).
- Ein Agent kann ein Plagiat eines existierenden Artikels erstellen und dieses als einen selbst geschriebenen Artikel in die Wikipedia einbringen (A4).
- Den Agenten ist es möglich, die Wikipedia nach doppelten Einträgen, nach Plagiaten und nach Wörtern, die nicht den Konventionen entsprechen, zu durchsuchen (A5) und die involvierten Autoren für diese Vergehen zu maßregeln (A6).
- Auch ist es den Agenten möglich, nach Wörtern in bestehenden Artikeln zu suchen, die einem von ihnen bereits verwendeten Schlüsselwort entsprechen (A7).
- Es ist dem Agenten auch möglich, keine der genannten Aktionen auszuüben und einfach nichts zu tun.

Die Bedienoberfläche

Die NetLogo-Wikipedia-Simulation hat eine grafische Bedienoberfläche, die sich in verschiedene Bereiche untergliedert. Ihre Elemente sind in Abbildung 3.1 dargestellt. In der linken oberen Ecke befinden sich drei Schaltflächen, über die die Simulation selbst gesteuert wird.

- Setup: Die Initialisierung der Simulation wird vorgenommen.

- Go: Ein einzelner Simulationsschritt wird abgearbeitet.
- Go mit Pfeilen: Eine unbegrenzte Anzahl von Simulationsschritten wird abgearbeitet. Wird diese Schaltfläche erneut betätigt, so pausiert die Simulation.

Unterhalb dieser drei Schaltflächen befinden sich Regler, die zur Konfiguration der Simulation dienen. Hiermit lassen sich alle wichtigen Parameter der Simulation einstellen.
Um diese Regler herum sind Ausgabefenster positioniert, in denen während der Simulation die aktuellen Werte bestimmter Parameter angezeigt werden.
Im rechten Teil des Bildschirminhalts befinden sich drei Fenster, in denen sich Parameterveränderungen in Form von Graphen im Verlauf der Simulation ablesen lassen.
Im mittleren Teil befindet sich ein Ausgabefenster, in dem während der Simulation Statusmeldungen aufgelistet werden. Wird die Simulation beendet, so wird in diesem Ausgabefenster eine Zusammenfassung wichtiger Parameter angezeigt. Rechts davon befindet sich ein Fenster, das die Agenten selbst als kleine farbige Symbole darstellt. Im unteren Teil der Bedienoberfläche befindet sich ein Ausgabebereich, in welchem während der Simulation Statusmeldungen und Nachrichten von NetLogo abgebildet werden.

Das Programm

Das NetLogo-Programm beinhaltet sowohl die Komponenten, die zur Kommunikation und zur Interaktion der Agenten benötigt werden, als auch die Komponenten, die die Normentwicklung verarbeiten. Hier besteht ein fundamentaler Unterschied zum später geplanten Entwurf. In diesem sind in EmIL-S die Komponenten vereinigt, die etwas mit Normentstehung zu tun haben. Auf der physikalischen Ebene spielen sich hingegen die restlichen Möglichkeiten der Interaktion ab. Dies ermöglicht es, auf die Komponenten zur Normentwicklung und Verarbeitung gezielter einzugehen und in Folge dessen ein tieferes Verständnis für Normen und deren Entwicklung zu erlangen.

Die Ergebnisse

Hauptziel der Umsetzung des Szenarios in ein NetLogo-Programm war, zu beweisen, dass eine Umsetzung in ein Programm überhaupt möglich ist. Dies wurde durch das vorliegende, lauffähige NetLogo-Programm gezeigt.
Des Weiteren bietet das beschriebene Programm die Möglichkeit, bereits Simulationsläufe durchzuführen und die dabei gewonnenen Daten auszuwerten. Dies ermöglicht einen

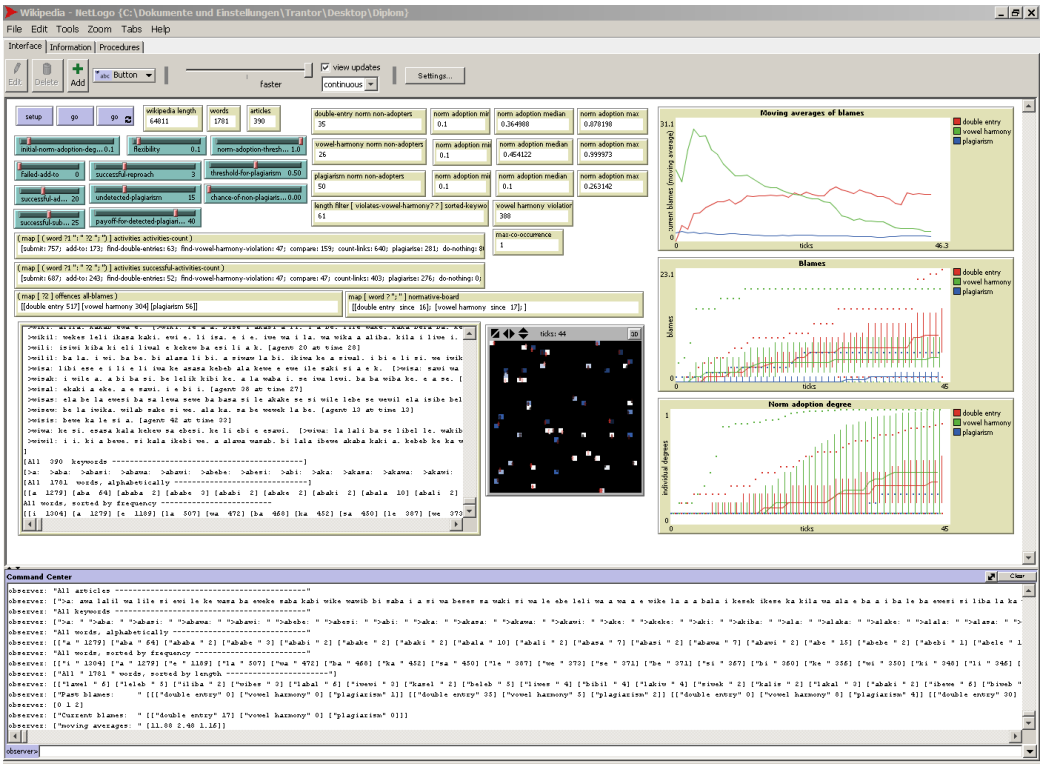

Abbildung 3.1: Benutzeroberfläche [Tro08]

wissenschaftlichen Einstieg. Jedoch zeigt das Programm Mängel auf, die durch das Projekt EmIL behoben werden sollen. So ist es nicht leicht erweiter- oder änderbar, da ein tiefes Einarbeiten in den Programmcode von Nöten ist. Auch ist das Automatisieren von Simulationsabläufen und die automatisierte Aufarbeitung und Auswertung der Daten mittels NetLogo nicht möglich. Da die Komponenten zur Normentstehung im NetLogocode integriert wurden, sind diese für Außenstehende nur sehr schwer änderbar. Eine Trennung dieser Komponenten wäre vorteilhaft. Dies war ein Hauptanliegen bei der Konzeptionierung von EmIL-S. Vgl. [Tro08].

3.1.3 Erweiterung des Szenarios

Das vorliegende Szenario bietet bereits eine Vielzahl interessanter Aspekte. Jedoch agieren die Agenten meist recht ähnlich. Um diesen Umstand zu beheben und einige neue Ideen mit einzubringen, soll das Szenario erweitert werden. Die Erweiterung betrifft den Umgang der Agenten mit der Sprache. An dieser Stelle ergibt sich eventuell die Möglichkeit, Normentstehungsprozesse besser zu verstehen. So gab es im ursprünglichen Szenario

für die Agenten die Möglichkeit, ein Wort entsprechend der sogenannten „vowel harmony“[1] zu schreiben oder aber ein Wort, welches sich nicht an diese Regelmenge hält zu erstellen. Ein Wort, das der vowel harmony entspricht, ist demnach richtig geschrieben. Alle Wörter, die dies nicht tun, sind falsch geschriebene Wörter. Die vowel harmony stellt also eine Art Sprachkonvention dar, welcher die Wörter entsprechen müssen. Das Szenario soll nun um eine Agentenklasse erweitert werden, in der Agenten zwar die vowel harmony kennen, sie jedoch genau umgekehrt nutzen. Bei diesen „Rebellen“ sind alle Wörter, die der vowel harmony entsprechen, falsch geschrieben und alle, die ihr nicht entsprechen, richtig geschrieben. Hierdurch ergibt sich im späteren Verlauf der Simulation eine interessante Neuerung. Da die Agenten durch ihre Analyse der Wikipedia falsche Wörter finden und deren Autor dafür mit einer Sanktion belegt wird, lernen die Agenten, was richtig und was falsch ist. Durch die neuen „Rebellen“ wird die Lernphase entscheidend geändert. Während es im ursprünglichen Szenario innerhalb der Simulation nach einiger Zeit zu einer Stabilisierung der Wortqualität gekommen ist, wird diese durch die zweite Agentenart womöglich verhindert. Mögliche Ergebnisse wären zum Beispiel, dass eine der beiden Gruppen sich auf Dauer durchsetzt oder aber, dass eine stabile Pendelbewegung zwischen den beiden Ideologien zu beobachten sein wird. Auch könnte, je nach Lernalgorithmus, die Art, in der Wörter geschrieben werden, völlig chaotisch sein. Möglich wäre auch, dass Agenten einer Gruppe durch Lernen zu einer anderen Gruppe überwechseln und ein steter Austausch zwischen diesen beiden Gruppen stattfindet. Durch diese Änderung des Szenarios lassen sich eventuell Wege der Normentstehung erkennen. So ist es durchaus möglich, dass die unterschiedlichen Agenten durch Emergenz[2] eine Norm entwickeln, die für beide Agentenklassen gültig ist. In einem solchen Fall wären die Faktoren, die zu dieser Normentstehung führen, herauszuarbeiten und zu analysieren.

[1] Der Begriff „vowel harmony“ beschreibt einen phonetischen Prozess, der in einigen Sprachen (z.B. Ungarisch) Verwendung findet. So beschreibt dieser Prozess das Angleichen von bestimmten Vokalen innerhalb eines Wortes in Abhängigkeit davon, ob andere Vokale innerhalb desselben Wortes auftreten. Hierbei folgen die Vokale nicht direkt aufeinander, sondern sind durch Konsonanten voneinander getrennt. Beispielsweise würde das fiktive Wort „aseka“ nicht der vowel harmony entsprechen. Nach Anwendung des phonetischen Prozesses würde das Wort „asaka“ oder aber „eseke“ lauten. Vgl. [Gol96].

[2] „[Emergenz ist] In Systemen das Auftreten von Merkmalen auf höheren Organisationsebenen, die nicht aufgrund bekannter Komponenten niedrigerer Ebenen hätten vorhergesagt werden können.“ [May00, Seite 403].

3.2 Konzeptioneller Entwurf

3.2.1 Definition der Anforderungen

Nach den Prinzipien der Softwaretechnik unterteilen sich Anforderungen in zwei Kategorien. Funktionale Anforderungen und nicht funktionale Anforderungen. Funktionale Anforderungen sind all jene Anforderungen, die direkt die Funktionen der Software definieren. Im Gegensatz dazu beschreiben die nicht funktionalen Anforderungen all jene Anforderungen, die die Eigenschaften der Software definieren. Vgl. [Rup04, Seite 140ff.]. Das zu implementierende Programm soll möglichst alle durch das Szenario vorgegebenen Funktionen beinhalten. Diesbezüglich wurden in [EmI08a, Seiten 4-7] bereits Anforderungen definiert, welche auch für die in dieser Arbeit implementierte Software gelten sollen. Die wichtigsten funktionalen Anforderungen sind nachfolgend aufgezählt:

1. Die implementierte Software muss Schnittstellen zu EmIL-S sowie zu MEME beinhalten.
2. Agenten müssen enthalten sein,
3. diese müssen Nachrichten empfangen und auch senden können.
4. Neue Wörter können von ihnen gebildet werden.
5. Die Agenten können Artikel schreiben, lesen, ergänzen, löschen, kopieren und verändern.
6. Die von Agenten ausführbaren Aktionen müssen in einer Konfiguration beschrieben sein.
7. Die Simulation muss steuerbar sein.
8. Parameter der Simulation müssen änderbar sein.
9. Die Ergebnisse der Simulation müssen in einer auswertbaren Form vorgehalten und ausgegeben werden.

Nicht funktionale Anforderungen an die Wikipedia-Implementation: Eine möglichst intuitive und einfache Bedienung der Software ist angestrebt. Hierfür sollen entsprechende graphische Benutzeroberflächen, Anzeigeflächen für Graphen sowie Möglichkeiten zur Varianz der Parameter erstellt werden. Die Software soll möglichst frei von Fehlern sein.

Es muss daher möglich sein, die Software zu testen, um mögliche Fehlerquellen aufzuspüren. Da es sich um einen Prototypen handelt, soll die Software in Teilen oder vollständig in anderen Projekten wiederverwendbar sein. Auch eine einfache Erweiterbarkeit und Neugestaltung bestehender Strukturen soll ermöglicht werden. Die Software sollte, wenn möglich, auf andere Betriebssystemplattformen portierbar sein, um ein möglichst breites Anwenderfeld abdecken zu können.
Die funktionalen Anforderungen ergeben sich direkt aus dem Wikipedia-Szenario. Die nicht funktionalen hingegen sind teilweise für jede Software gültig, teilweise sollen sie die Arbeit im Projekt oder aber die Arbeit Projektfremder erleichtern. So ist die Anforderung, dass die Software auch auf anderen Betriebssytemen lauffähig sein soll, schon allein durch den Umstand gegeben, dass sie und auch EmIL-S in Java geschrieben werden und Java auf einer Vielzahl unterschiedlicher Plattformen zur Ausführung gebracht werden kann. Für das Simulationswerkzeug Repast und auch die Steuerungskomponente MEME verhält es sich ähnlich. Auch sie sind auf vielen unterschiedlichen Plattformen nutzbar. Dadurch wird es einem sehr weiten Personenkreis möglich, diese Komponenten zu nutzen. Dadurch, dass die Software in der objektorientierten Programmiersprache Java geschrieben wird, muss sie in sinnvolle Klassen und ihre Methoden gegliedert werden. Hierdurch wird es möglich, die Software leicht zu erweitern, indem Methoden abgeändert oder überschrieben werden. Auch ist es möglich, neue Klassen hinzuzufügen.

3.2.2 Entwurf der Softwarearchitektur

Um Simulationen durchführen zu können, wird die gesamte Simulationsumgebung benötigt. Zur Durchführung einer Simulation ist ein Modell nötig, das auf der physikalischen Schicht implementiert werden muss. Zusätzlich muss für die EmIL-S-Komponente eine initiale Konfiguration erstellt werden. Mit diesen Komponenten ist es möglich, die Simulation innerhalb der Simulationsumgebung zu laden. Hiernach werden entweder händisch Parameter für einen Durchlauf eingestellt oder aber MEME wird für die Steuerung und Auswertung einer Reihe von Durchläufen konfiguriert. Das Zusammenspiel dieser Komponenten zeigt Schaubild 3.2. Dieses stellt sich in drei Ebenen dar. Die oberste Ebene zeigt in einem Anwendungsfalldiagramm, was das System leisten soll. So sind hier die Anwendungsmöglichkeiten aus Sicht eines möglichen Nutzers dargestellt. Die zweite Ebene, welche bereits eingangs erwähnt wurde, zeigt in einem Komponentendiagramm das Zusammenspiel der verschiedenen Teile der Simulationsumgebung. Für das Simulationswerkzeug muss dem Modell entsprechend eine Implementation erstellt werden. Diese

ist im unteren Teil des Schaubildes abgebildet und zeigt in einem Klassendiagramm drei Klassen und ihre wichtigsten Methoden.

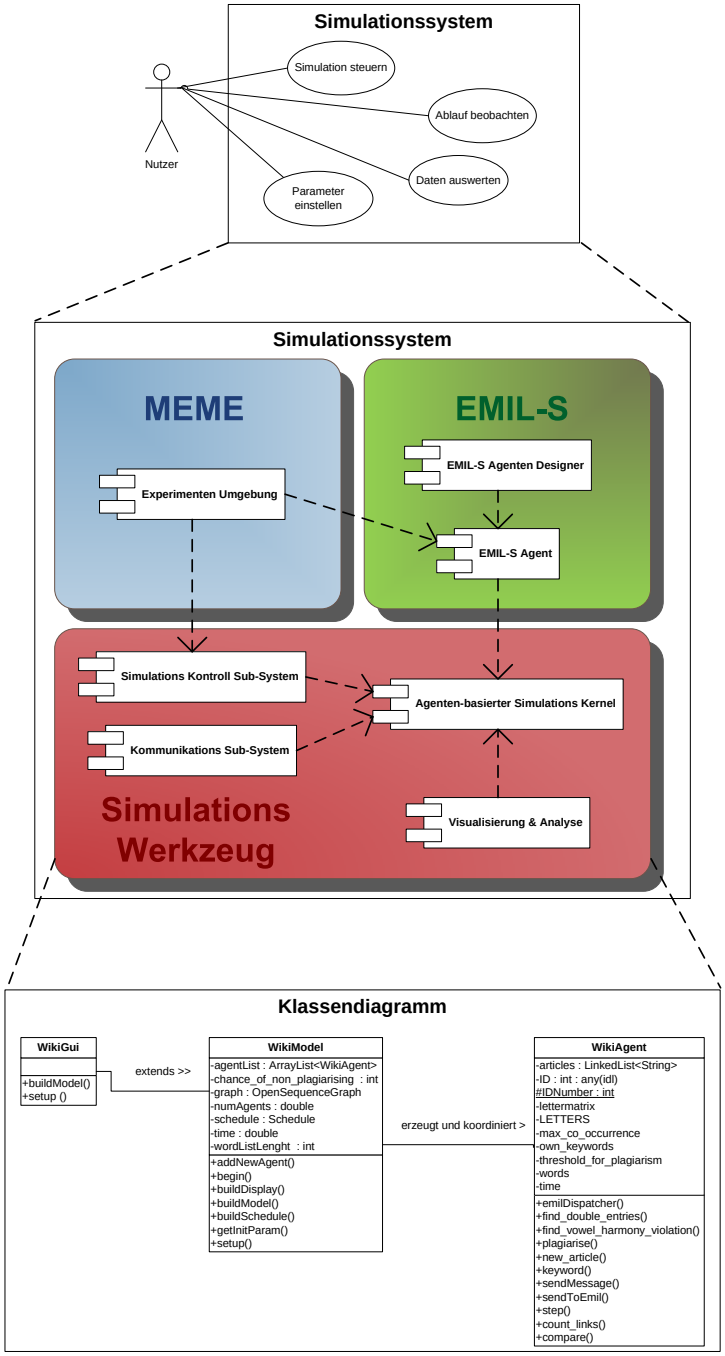

Abbildung 3.2: Überblick

Die drei beschriebenen Klassen stellen die Implementation des Modells und damit die physikalische Schicht der Simulation dar. Diese Implementation kommt innerhalb des Simulationswerkzeuges zur Ausführung und interagiert mit den beiden anderen Komponenten. So wird das Simulationswerkzeug zu einem späteren Zeitpunkt von MEME gesteuert. Die Interaktion mit EmIL-S besteht in Nachrichten, die zwischen diesen beiden Komponenten ausgetauscht werden können. Die Implementation innerhalb der physikalischen Schicht stellt sich nun folgendermaßen dar.
Die drei Klassen bündeln jeweils voneinander trennbare Teile der physikalischen Schicht. So gibt es eine Klasse die alle graphischen Bedienelemente sowie Fenster für Anzeigen und Graphen enthält. Die zweite Klasse, welche den Ablauf der Simulation innerhalb der physikalischen Schicht steuert sowie die Agenten erzeugt, beinhaltet all jene Methoden, die vom Simulationswerkzeug vorgegeben sind. Zusätzlich beinhaltet sie Methoden zur Erzeugung von Agenten, zum Setzen von initialen Parametern und eine Methode, die den zeitlichen Ablauf steuert. Die dritte Klasse beinhaltet all jene Methoden, die das Verhalten der Agenten erzeugen und bestimmen. Ein Simulationslauf gestaltet sich folgendermaßen: Innerhalb der Steuerungsklasse, welche den Namen WikiModel trägt, werden Agenten erzeugt und mit initialen Parametern versehen. Die Simulationsoberfläche wird initialisiert und eine Art Zeitplan für die einzelnen Simulationsschritte wird erstellt. Dieser Zeitplan definiert sich wie folgt:
Es gibt Simulationsschritte, die nacheinander durchlaufen werden; die sogenannten Ticks. Innerhalb eines Ticks wird jeder Agent aufgerufen, seine Tätigkeiten durchzuführen. Das bedeutet, dass alle vorhandenen Agenten innerhalb eines Ticks die Möglichkeit erhalten, Aktionen durchzuführen. Dies ist vergleichbar mit einem Tag in der realen Welt. Jeden Tag haben Personen die Möglichkeit, innerhalb eines Wikis Inhalte zu erstellen, zu ergänzen und andere Tätigkeiten auszuüben. Für den Betreiber des Wikis ist es möglich, die Tätigkeiten zeitlich anhand des Datums einzuordnen. Genauso verhält es sich für den vorliegenden Entwurf.
Innerhalb der Agenten-Klasse sind alle Methoden aufgelistet, die das mögliche Verhalten bestimmen. Sobald ein Agent von der Klasse WikiModel aufgerufen wird, seine Tätigkeiten durchzuführen, folgt dieser Agent einem definierten Ablauf. Das Verhalten der Agenten ergibt sich zum einen durch kognitive Prozesse, zum anderen durch physikalische Abfolgen von Tätigkeiten, die festgelegt sind und keiner Entscheidung bedürfen. Die kognitiven Prozesse jedoch spielen sich ausschließlich innerhalb der Komponente EmIL-S ab. Das Zusammenspiel von Agent und EmIL-S-Gegenstück ist in Abbildung 3.3 als Aktivitätsdiagramm dargestellt. Jeder Agent, der aufgerufen wird seine Tätigkeiten auszuü-

ben, hat aufgrund der initialen Konfiguration mehrere mögliche Tätigkeiten zur Auswahl. Daher ist es hier notwendig, die Entscheidung welche Tätigkeit ausgeübt werden soll, von EmIL-S treffen zu lassen. Jeder Agent prüft also zunächst, ob bereits Nachrichten von EmIL-S eingetroffen sind. Ist dies nicht der Fall, so sendet der Agent eine Nachricht an EmIL-S, die ein event enthält, das EmIL-S dazu veranlasst, die Wahl für den Agenten zu treffen. Hat der Agent nun eine Nachricht von seinem EmIL-S-Gegenstück erhalten, so identifiziert er anhand des Inhaltes die auszuübende Tätigkeit und führt diese aus. Tätigkeiten wiederum können dazu führen, dass weitere Folgeentscheidungen zu treffen sind, welche wieder Interaktion mit EmIL-S erforderlich machen und wiederum zu Nachrichten führen. Ein Beispiel für Kommunikation zwischen einem physikalischen Agenten und

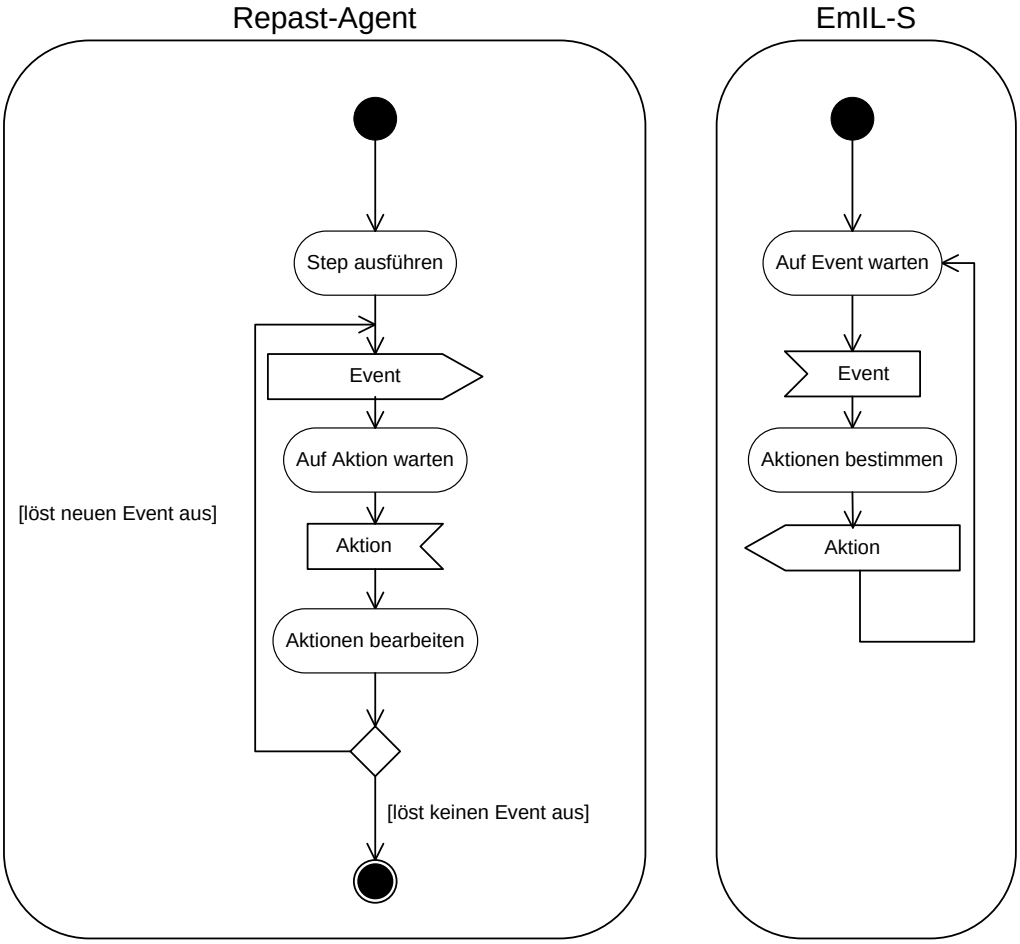

Abbildung 3.3: Agent-EmIL-S-Verhalten

seinem EmIL-S-Gegenstück ist in einem Sequenzdiagramm in Abbildung 3.4 dargestellt. Der in diesem Diagramm beschriebene Agent sendet zu Beginn eine Nachricht an EmIL-S. In dieser Nachricht wird der event „E1“ gesendet. Dieser führt dazu, dass EmIL-S zwischen mehreren diesem event zugeordneten Aktionen wählen kann. Nachdem EmIL-S eine Entscheidung gefällt hat, wird eine Antwort an den Agenten zurückgesendet. Die Aktion ist im beschriebenen Fall „A1“, welche besagt, dass der Agent einen neuen Artikel schreiben soll. Diese Aktion wird nun vom Agenten abgearbeitet und gliedert sich in mehrere Punkte. So muss der Agent, um einen Artikel erstellen zu können, zunächst ein Schlüsselwort erstellen. Nachdem er dieses gebildet hat, prüft er, ob das Schlüsselwort bereits verwendet wurde. Ist dies der Fall, so sendet er einen event „E12“ an EmIL-S, andernfalls sendet er „E11“. Je nach gesendetem event ergibt sich der weitere Handlungsablauf. So kann EmIL-S als Reaktion auf ein bereits vorhandenes Schlüsselwort entscheiden, ob ein neuer Artikel mit dem selben Schlüsselwort zusätzlich zum bereits in der Wikipedia enthaltenen hinzugefügt werden soll oder, ob der bereits bestehende Artikel mit dem selben Schlüsselwort um weitere Informationen ergänzt werden soll.

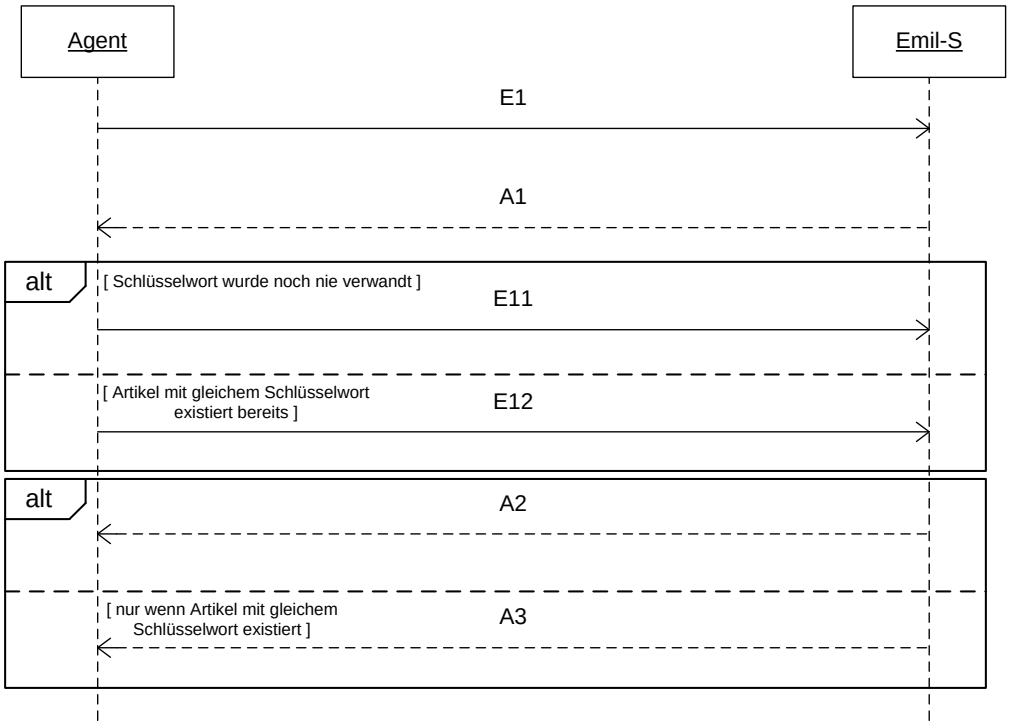

Abbildung 3.4: Kommunikation

Die von Agenten durchführbaren Tätigkeiten sind immer Aktionen. Diese sind in einer Konfiguration für EmIL-S festzuhalten. Jede Aktion kann von spezifischen events ausgelöst werden und ergibt so das Verhalten der Agenten. Die Zusammenhänge zwischen events und actions, die aus dem Wikipedia-Szenario abgeleitet wurden, finden sich in Abbildung 3.5. In dieser Abbildung sind die events jeweils beginnend mit dem Buchstaben „E“, gefolgt von einer Nummer, gekennzeichnet. Die zugehörigen Aktionen sind jeweils unterhalb der events in einer Baumstruktur annotiert. Die möglichen Abfolgen von events und Aktionen ergeben sich einerseits durch die Baumstruktur, andererseits durch Pfeile, welche mögliche Abfolgen beschreiben. So löst ein event jeweils eine oder mehrere actions aus, die direkt zu diesem event gehören. Diese Aktionen können wiederum, durch Prozesse auf der physikalischen Ebene, events auslösen. Diese Verknüpfungen sind jeweils durch Pfeile gekennzeichnet. Der Übersichtlichkeit halber wurden physikalische Ereignisse und Tätigkeiten in der Farbe grün unterlegt, während Entscheidungen und events, die EmIL-S betreffen, in Blau eingefärbt wurden. Analog hierzu sind die Pfeile unterschiedlich eingefärbt. Nachdem nun alle möglichen events, actions und Abläufe beschrieben wurden, ist es möglich, eine Konfiguration für EmIL-S zu erstellen. Diese wird in Kapitel 4.1 beschrieben. Die Implementation der Methoden, welche auf der physikalischen Ebene benötigt werden, ist nunmehr anhand der Beschreibung und der gesammelten Erkenntnisse möglich.

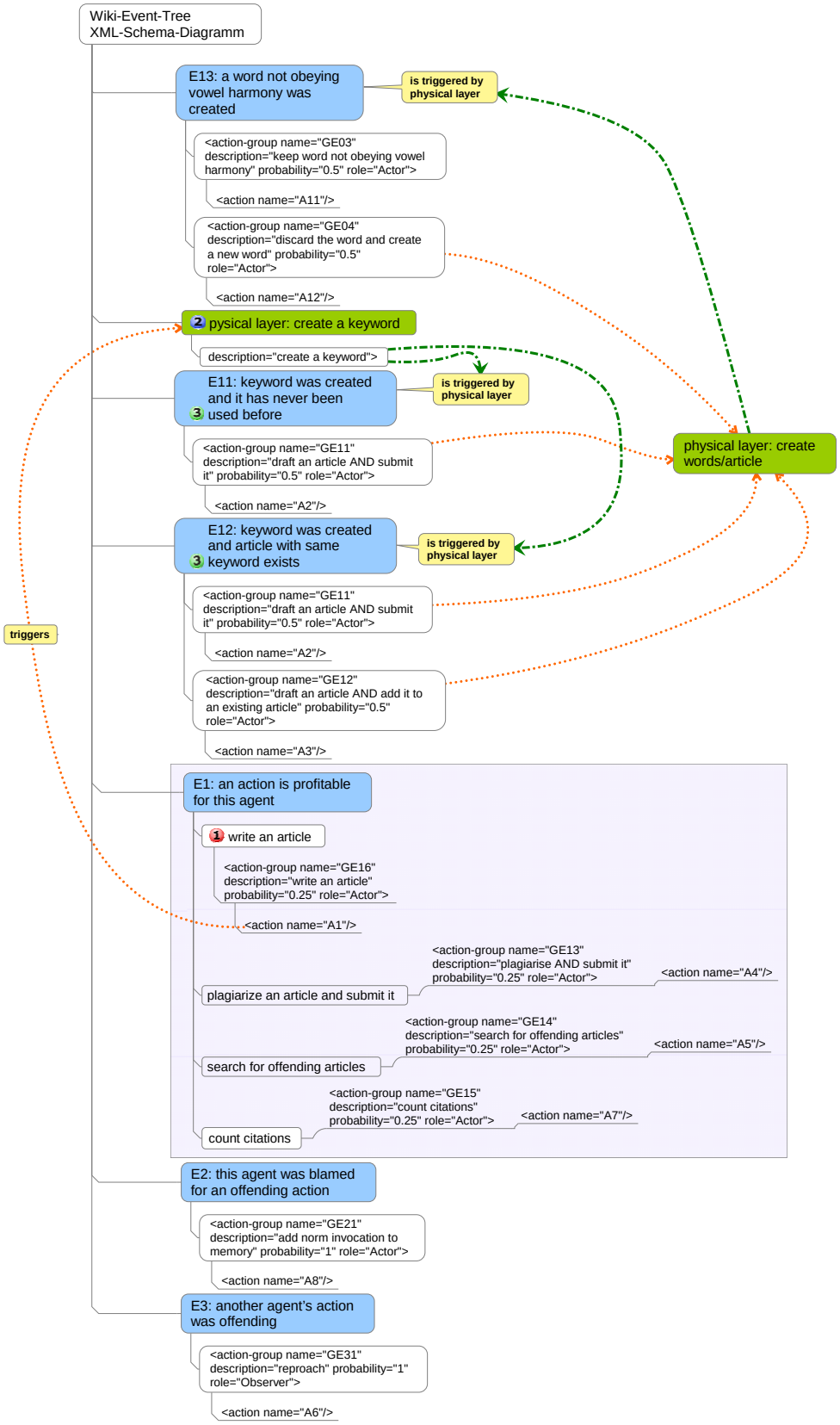

Abbildung 3.5: Diagramm der Abläufe

Kapitel 4

Implementierung

4.1 Die Konfiguration

Zur Beschreibung der Konfiguration von EmIL-S wurden verschiedene Beschreibungsarten in Betracht gezogen. Die Wahl fiel auf die„eXtensible Markup Language“ (XML) und in diesem speziellen Fall, auf XML-Schema. Die Datenbeschreibungssprache XML ist für diesen Zweck sehr gut geeignet, da sich Daten und ihre Struktur damit sehr einfach modellieren lassen und das Ergebnis in Form einer XML-Datei von Menschen trotzdem noch recht einfach gelesen und verstanden werden kann. XML-Schema ist im Vergleich mit Dokumententyp-Definitionen (DTD)[1] deutlich mächtiger und für den hier angestrebten Zweck besser geeignet. So lassen sich mittels XML-Schema Kardinalitätsangaben sowie differenziertere Beschreibungen der Daten vornehmen als dies mittels DTD möglich wäre. Vgl. [EE04, Seite 83]. Diese Beschreibungen sind in einer separaten XML-Schema-Definitions-Datei (kurz XSD-Datei) abgelegt. Mit dieser Datei wird eine erstellte XML-Datei auf Konformität zum definierten Schema hin überprüft.

4.1.1 Beschreibung der XML-Sprache

Die initiale Konfiguration für EmIL-S wird in Form von event-action trees beschrieben, welche als XML-Datei an EmIL-S übergeben werden. Diese beinhalten events, welche von irgendetwas oder irgendjemandem ausgelöst werden. Dies kann sowohl auf der physikalischen Ebene der Simulation, als auch innerhalb von EmIL-S selbst geschehen. Hier-

[1] Dokumententyp-Definitionen kurz DTD werden in XML zum Festlegen der Struktur der Dokumenteninstanz genutzt. Vgl. [EE04, Seite 19].

bei stellt ein event ein besonderes, von allen anderen events unterscheidbares, Ereignis dar. Events bilden die Wurzel eines jeden event-action trees. Von ihnen ausgehend folgen die sogenannten „action-groups“, in welchen bestimmte zusammengehörige actions gebündelt werden. Diese actions stellen Aktionen dar, die abgearbeitet werden und in Summe den Ablauf der eigentlichen Simulation bilden. Bei der vorgestellten Struktur haben sowohl die action-groups, als auch die actions, Wahrscheinlichkeiten. So ist es möglich, dass mehrere action-groups von einem Event ausgelöst werden können und zwar mit unterschiedlicher Wahrscheinlichkeit. Innerhalb einer jeden action-group hingegen wird immer nur eine einzige von mehreren vorhandenen actions ausgelöst, dies allerdings wiederum auch mit einer bestimmten Wahrscheinlichkeit. Die Wahrscheinlichkeiten, welche die actions bestimmen, dürfen daher in Summe niemals 1.0 überschreiten. Eine schematische Darstellung für einen event-action tree wurde bereits in Abbildung 2.2 beschrieben. Die Konfigurationsdatei für EmIL-S beinhaltet jedoch noch mehr. Sie ist in drei Abschnitte gegliedert:

1. „Global definitions“, hier sind alle *events* definiert und mit einer verständlichen Beschreibung versehen. Darauf folgen die Definitionen der *actions*, jedoch noch ohne Zuordnung zu einer *action-group* und ohne Wahrscheinlichkeiten. Zu den jeweiligen *actions* ist neben der Beschreibung noch ein *type* zu definieren sowie eine *expression*. Der *type* stellt lediglich dar, ob sich die beschriebene *action* intern oder extern auswirkt. Die *expression* dient der weiteren Verarbeitung und stellt in den meisten Fällen eine Nachricht dar, die an die jeweilige Schicht gesendet wird.

2. „Agent definitions“, in diesem Abschnitt werden *actions* und *events* verknüpft, die direkt die Agenten betreffen. Hier werden nach einer oder mehreren Agentendefinitionen alle *events*, *action-groups* und *actions* aufgelistet und definiert, die sich direkt den Agenten zuordnen lassen. An dieser Stelle ist es demnach möglich, mehrere unterschiedliche Agentenarten zu definieren. Die *events* und die *actions*, die im globalen Teil definiert wurden, finden hier ihre Verwendung. Es ist nicht möglich, *actions* oder *events* zu verwenden, die nicht im globalen Teil deklariert wurden.

3. „Environmental definitions“, hier folgen die *event-action trees* und Regelsätze, welche die Agentenumgebung betreffen.

Mittels dieser drei Abschnitte lassen sich alle notwendigen Ereignisse, Aktionen und Regeln vorgeben, die zum Ablauf der Simulation notwendig sind.

4.1.2 Überführung der Konfiguration in XML

Um eine XML-Konfiguration für EmIL-S zu finden, ist es notwendig, das zu modellierende Szenario vollständig zu verstehen. Besonderes Augenmerk muss auf alle Normen beeinflussende Faktoren gelegt werden. Für das vorher beschriebene Wikipediaszenario wurden die Normen beeinflussenden Faktoren wie folgt definiert:

- Alle Entscheidungen, die Agentenverhalten betreffen, sind von EmIL-S zu treffen.
- Alle Reaktionen auf getroffene Entscheidungen werden an EmIL-S weitergereicht und EmIL-S ermittelt hieraus Normen.

So entscheiden die Agenten im übertragenen Sinne, welche Tätigkeit sie ausüben wollen, anhand von in der Vergangenheit gemachter Erfahrungen. Zu Beginn einer Simulation existieren keine gemachten Erfahrungen. Es wird auf die initiale Konfiguration zurückgegriffen.
Natürlich gibt es auch Aktionen, die innerhalb der physikalischen Schicht ganz ohne Interaktion mit EmIL-S ablaufen. Dies sind allerdings durch das Szenario definierte Tätigkeiten, die keinerlei wichtige Entscheidungen beinhalten und auch für das gegebene Szenario keinerlei Normentwicklung nötig machen. So ist zum Beispiel die Entscheidung einen neuen Artikel zu schreiben eine Entscheidung, die von EmIL-S getroffen wird. EmIL-S entscheidet jedoch nicht, wie lang dieser Artikel sein soll. Die Länge eines Artikels spielt im beschriebenen Szenario keinerlei Rolle. Denkbar wäre es aber, die Länge in einer abgewandelten Form des Szenarios doch eine Rolle spielen zu lassen, zum Beispiel bei der Bewertung der Reputation des Autors oder Ähnlichem. In einem solchen Fall wäre es notwendig, das zugrundeliegende Konzept abzuändern und die Entscheidung über die Länge eines Artikels an EmIL-S zu übertragen. Dafür wären sowohl Änderungen an der Konfiguration wie auch am Programm der physikalischen Ebene notwendig.

4.1.3 Vorstellung der XML-Konfiguration

Die Konfiguration für das zugrunde liegende Wikipedia-Szenario ist in einer XML-Datei festgeschrieben. Die Struktur dieser XML-Konfiguration ist in Schaubild 3.5 schematisch aufgezeichnet.
Sie unterteilt sich in drei größere Bereiche, auf die im Folgenden eingegangen wird.
Der erste Teil ist der globale Definitionsteil. Dieser beinhaltet alle events und actions, die für die Simulation benötigt werden. Er ist im Listing 4.1 dargestellt. Die Zeilen 1-3 dieses Listings definieren das XML-Format und die XML-Schemadatei, welche zur

Validierung der XML-Datei benötigt wird. Nachfolgend finden sich hier von Zeile 6-11 alle events, die auch in Schaubild 3.5 abgebildet sind. Jeder event hat einen Namen und eine Beschreibung. Zusätzlich zu den events sind alle verwendeten actions von Zeile 13-60 aufgelistet. Auch hier hat jede action einen Namen und eine Beschreibung, zusätzlich enthalten die actions eine „expression“ und einen „type“. Das Feld „type“ bestimmt, wie sich eine action auswirkt. Diese kann sich entweder auf den agierenden Agenten selbst oder aber auf andere auswirken. Das Feld „expression“ beinhaltet den Wert, der an die jeweilige Schicht gesendet wird. Dieser ist sehr wichtig, um die action innerhalb von Repast oder EmIL-S eindeutig bestimmen zu können. Es ist zu beachten, dass in diesem globalen Definitionsteil noch keinerlei Zuordnung von actions zu events definiert ist.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <emil-s-configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:noNamespaceSchemaLocation="file:///C:/Path/emil-s-configuration.xsd">
4    <global-definitions>
5      <rule-elements>
6        <event name="E1" description="an action is profitable for this agent"/>
7        <event name="E2" description="this agent was blamed for an offending action
             "/>
8        <event name="E3" description="another agents action was offending"/>
9        <event name="E11" description="keyword was created and it has never been
             used bevor"/>
10       <event name="E12" description="keyword was created and article with same
             keyword exists"/>
11       <event name="E13" description="a word not obeing vowel harmony was created
             "/>
12
13       <action name="A1" description="create a keyword">
14         <type>External</type>
15         <expression>1</expression>
16       </action>
17       <action name="A2" description="write an article and submit it">
18         <type>External</type>
19         <expression>2</expression>
20       </action>
21       <action name="A3" description="write an article and add an it to existing
             keyword">
22         <type>External</type>
23         <expression>3</expression>
24       </action>
25       <action name="A4" description="plagiarise an article and submit it">
26         <type>External</type>
27         <expression>4</expression>
28       </action>
29       <action name="A51" description="search the current state of the
```

```
            encyclopaedia for double entries">
30          <type>Internal</type>
31          <expression>51</expression>
32        </action>
33        <action name="A52" description="search the current state of the
              encyclopaedia for words that do not obey the vowel harmony (they are
              bad style)">
34          <type>Internal</type>
35          <expression>52</expression>
36        </action>
37        <action name="A53" description="search the current state of the
              encyclopaedia for plagiarisms">
38          <type>Internal</type>
39          <expression>53</expression>
40        </action>
41        <action name="A6" description="reproach the respective author or authors">
42          <type>External</type>
43          <expression>6</expression>
44        </action>
45        <action name="A7" description="count articles that contain a word about
              which they wrote an article">
46          <type>Internal</type>
47          <expression>7</expression>
48        </action>
49        <action name="A8" description="add norm invocation to memory">
50          <type>Internal</type>
51          <expression>8</expression>
52        </action>
53        <action name="A11" description="keep word">
54          <type>External</type>
55          <expression>11</expression>
56        </action>
57        <action name="A12" description="discard word and create a new word">
58          <type>External</type>
59          <expression>12</expression>
60        </action>
61
62      </rule-elements>
63    </global-definitions>
```

Listing 4.1: XML-Global-Definitions

Diese Zuordnung folgt im Agenten-Definitionsteil, welcher dem globalen Definitionsteil nachfolgt und in Listing 4.2 dargestellt ist. Innerhalb dieses Abschnittes erfolgt die Zuordnung von actions zu events für den Agententyp „Autor“. Dieser ist in dem beschriebenen Szenario der einzige Agententyp (Zeile 2). Es ist jedoch ohne weiteres möglich, mehr als einen Agententyp zu definieren. Hierfür wäre es lediglich notwendig, den Agenten-Definitionsteil um einen neuen Agententyp zu erweitern und für diesen wieder eine spe-

zifische Zuordnung von actions zu events zu definieren. In der vorliegenden Definition finden sich nun die Zuordnungen von actions zu events von Zeile 3-56. Diese Zuordnungen finden in Form von event-action trees statt. Mittels XML wird eine Schachtelung vorgenommen, welche die bereits beschriebene Baumstruktur ergibt. Als Beispiel sei hier der event E1 beschrieben. Die Definition reicht für E1 von Zeile 28-44. Der hier definierte event-action tree hat einen Namen: „do something“ und eine Beschreibung. Diesem event-action tree ist in Zeile 29 das event E1 zugeordnet. Unterhalb dieser Zeile folgen vier action groups G13, G14, G15, G16, welche jeweils eine oder mehrere Aktionen A1, A4, A51, A52, A53, A7 beinhalten. Zu beachten ist, dass ein event nur in jeweils einem event-action tree auftauchen darf, da ein event ein eindeutiges Ereignis darstellen soll. Die zugeordneten Gruppen und Aktionen haben jeweils Wahrscheinlichkeiten. Die Wahrscheinlichkeitswerte der Gruppen definieren, mit welcher Wahrscheinlichkeit EmIL-S bei einer Entscheidung diese Gruppe auswählen wird. Es kann mehr als eine Gruppe ausgewählt werden, daher darf die Wahrscheinlichkeit an dieser Stelle frei zwischen 0-1 gewählt werden. Anders verhält es sich für die Wahrscheinlichkeiten an den Aktionen. Diese dürfen innerhalb einer Gruppe in Summe 1.0 nicht überschreiten. Ist in einer Gruppe nur eine Aktion aufgelistet, so ist es ratsam, die Wahrscheinlichkeit für diese Aktion auf 1 zu setzen, da es sonst innerhalb von EmIL-S zu Fehlinterpretationen kommen kann.

```
1  <agent-definitions>
2    <agent name="author" id="Jimmy_Wales">
3      <initial-rule-base>
4
5        <event-action-tree name="judge-word" description="base event-action-tree
             ">
6          <event name="E13"/>
7          <action-group name="GE03" description="judge-word" probability="1" role
               ="Actor">
8            <action name="A11" probability="0.5"/>
9            <action name="A12" probability="0.5"/>
10         </action-group>
11       </event-action-tree>
12
13       <event-action-tree name="write article process" description="desicion
             weather or not to add to existing article">
14         <event name="E12"/>
15         <action-group name="GE12" description="desicion weather or not to add
             to existing article" probability="1" role="Actor">
16           <action name="A2" probability="0.5"/>
17           <action name="A3" probability="0.5"/>
18         </action-group>
19       </event-action-tree>
```

```
20
21          <event-action-tree name="write article process" description="add article
                to wiki">
22            <event name="E11"/>
23            <action-group name="GE11" description="draft an article AND submit it"
                  probability="1" role="Actor">
24              <action name="A2" probability="1"/>
25            </action-group>
26          </event-action-tree>
27
28          <event-action-tree name="do something" description="following event-
                action-tree">
29            <event name="E1"/>
30            <action-group name="GE16" description="write an article" probability
                  ="0.8" role="Actor">
31              <action name="A1" probability="1"/>
32            </action-group>
33            <action-group name="GE13" description="plagiarise AND submit it"
                  probability="0.5" role="Actor">
34              <action name="A4" probability="1"/>
35            </action-group>
36            <action-group name="GE14" description="search for offending articles"
                  probability="0.3" role="Actor">
37              <action name="A51" probability="0.2"/>
38              <action name="A52" probability="0.1"/>
39              <action name="A53" probability="0.7"/>
40            </action-group>
41            <action-group name="GE15" description="count citations" probability
                  ="0.3" role="Actor">
42              <action name="A7" probability="1"/>
43            </action-group>
44          </event-action-tree>
45
46           <event-action-tree name="learning process" description="this agent was
                blamed for an offending action and learns from that">
47           <event name="E2"/>
48           <action-group name="GE21" description="add norm invocation to memory"
                probability="1" role="Actor">
49             <action name="A8" probability="1"/>
50           </action-group>
51           </event-action-tree>
52
53           <event-action-tree name="reaction" description="another agents action
                was offending, this is the reaction">
54           <event name="E3"/>
55           <action-group name="GE31" description="reproach" probability="1" role="
                Observer">
56             <action name="A6" probability="1"/>
57           </action-group>
58           </event-action-tree>
```

```
59
60          </initial-rule-base>
61        </agent>
62      </agent-definitions>
```

Listing 4.2: XML-Agent-Definitions

Im abschließenden Teil, dem Teil für die Definition der Umgebung, welcher nicht extra aufgelistet wurde, finden sich keine Definitionen. Er ist leer, da die Verwendung bisher noch nicht spezifiziert wurde. Für das hier vorliegende Szenario ist er nicht notwendig.

4.1.4 Erweiterungen der XML-Sprache

Zielsetzung bei der Konzeption der XML-Sprache war es, diese so einfach wie möglich zu halten, jedoch mittels der XML-Eigenschaften sehr präzise und maschinenverständliche Formulierungen zu ermöglichen. XML erlaubt es bereits sehr früh, Fehler in der Konfiguration zu vermeiden, indem Vorbedingungen geprüft werden. Im Verlauf dieser Arbeit wurden zu diesem Zweck Integritätsbedingungen erarbeitet und zur ursprünglichen XSD-Datei hinzugefügt. Dies wurde durch die XML-Elemente key, keyref und selector möglich. Mit diesen wird eine Fremdschlüsselbeziehung zwischen den im globalen Teil definierten und den im späteren Teil verwendeten events und actions erzeugt. Dies ermöglicht einem XML-Prozessor zu prüfen, ob alle verwendeten events und actions auch im globalen Teil definiert wurden. Ist dies nicht der Fall, ist die vorgelegte XML-Datei nicht Schema konform. Auf diese Weise lassen sich bereits im Vorhinein mittels XML-Datei und XML-Schema einige Fehler vermeiden. Die folgenden Zeilen wurden zur XSD-Datei hinzugefügt:

```
1   <!-- ab hier Bedingungen für referentielle Integritaet /begin of integrity
        constraints -->
2   <!-- fuer die events / for the events-->
3   <xsd:key name="my-event-Id">
4       <xsd:selector xpath="./global-definitions/rule-elements/event"/>
5       <xsd:field xpath="@name"/>
6     </xsd:key>
7     <xsd:keyref name="my-event-Id-ref" refer="my-event-Id">
8       <xsd:selector xpath="./agent-definitions/agent/initial-rule-base/event-
            action-tree/event"/>
9       <xsd:field xpath="@name"/>
10    </xsd:keyref>
11    <!--und fuer die actions/ for the actions -->
12    <xsd:key name="my-action-Id">
13      <xsd:selector xpath="./global-definitions/rule-elements/action"/>
```

```
14        <xsd:field xpath="@name"/>
15      </xsd:key>
16      <xsd:keyref name="my-action-Id-ref" refer="my-action-Id">
17        <xsd:selector xpath="./agent-definitions/agent/initial-rule-base/event-
              action-tree/action-group/action"/>
18        <xsd:field xpath="@name"/>
19      </xsd:keyref>
20      <!-- ende der Integritaetsbedingungen / end of integrity constraints -->
```

Listing 4.3: constraints

Eine weitere Frage, die im Projekt diskutiert wurde war, ob es von Vorteil wäre, die optionale Angabe der Wahrscheinlichkeiten an den actions und action-groups zu erzwingen. Mit den Möglichkeiten von XML und XML-Schema wäre es hier durchaus möglich, die Angabe verpflichtend zu machen. Zum Beispiel mit der Zeile: „<xsd:attribute name=“probability“ use required>“. Auch könnte man den default-Wert unter Zuhilfenahme von „default=1.0“ schon voreinstellen. Jedoch bieten XML und XML-Schema leider keine Möglichkeit zu prüfen, ob die an den actions annotierten Wahrscheinlichkeiten in Summe kleiner 1.0 sind, wodurch es zu Problemen kommen kann, wenn mehr als eine Aktion definiert wird und die Wahrscheinlichkeiten nicht abgeändert werden. Vgl. [EE04, Seite 137].

4.2 Das Simulationswerkzeug

4.2.1 Auswahl des Simulationswerkzeuges

Für die Entscheidungsfindung stellten sich einige grundlegende Anforderungen:

1. Das zu findende Simulationswerkzeug muss mit EmIL-S zusammenarbeiten können und die Anpassung an EmIL-S sollte mit einem handhabbaren Aufwand erfolgen können.

2. Das Simulationswerkzeug sollte auch mit MEME zusammenarbeiten können.

3. Die Entwicklung der physikalischen Schicht jeden Modells sollte möglichst leicht zu erstellen, zu warten und anzupassen sein. Auch sollte es selbst einem fachfremden Personenkreis möglich sein, in kurzer Zeit die zur Erstellung eines Modells nötigen Kenntnisse zu erwerben.

Als Simulationswerkzeug wurde das quelloffene und frei zugängliche Repast-J gewählt. Dieses lässt sich in der Version 3.1 in Java programmieren und ist sehr einfach mittels einer Schnittstelle mit EmIL-S verknüpfbar. Des Weiteren besteht die Möglichkeit, mit MEME in Repast ablaufende Simulationsläufe zu steuern und zu überwachen sowie die dabei entstehenden Daten auszuwerten. Da eine Verknüpfung seitens MEME zu Repast bereits entwickelt wurde und auch eine Verknüpfung zu EmIL-S entwickelt werden soll, fiel die Wahl auf Repast-J 3.1, da es auch den oben genannten dritten Punkt erfüllt. Repast liegt auf der Entwicklerseite auch in einer sehr viel neueren Version, „Repast Simphony“, vor. Diese wurde für die vorliegende Arbeit allerdings nicht gewählt. So ist Repast Simphony bislang noch nicht mittels MEME steuerbar. Auch ist die Programmierung für Repast Simphony komplexer und fehlerträchtiger als bei der verwendeten Version 3.1, da in Repast Simphony eine weitere Skriptsprache namens „Groovy“ eingebaut wurde. Das mehrfache Umwandeln von einer grafischen Notation in eine Skriptsprache und dann in die eigentliche Programmiersprache Java führt zu einer Vielzahl möglicher Fehler.

4.2.2 Beschreibung des Simulationswerkzeuges

Repast ist ein Simulationswerkzeug zur Modellierung von Agenten basierten Simulationen. Der Name Repast ist ein Akronym für: „Recursive Porous Agent Simulation Toolkit“. Es wurde ursprünglich von Sallach, Collier, Howe, North und anderen an der Universität von Chicago entwickelt. Später wurde die Weiterentwicklung und Pflege von Repast

von diversen Organisationen durchgeführt, wie zum Beispiel vom Argonne National Laboratory. Zum Zeitpunkt der Erstellung dieser Arbeit wurde Repast von einer non-profit-Organisation gepflegt und weiterentwickelt, die sich „Repast Organization for Architecture and Development“ (ROAD) nennt. Die Software Repast ist im Internet frei erhältlich. Repast ermöglicht es, mit Hilfe von Java ein Modell zu implementieren. Hierfür bietet es eine Reihe von Methoden an, welche die Implementierung sehr einfach machen. So ist es möglich, auf Methoden zur Steuerung der Simulation zurückzugreifen. Auch sind Gerüste für die Agenten bereits vorhanden und müssen nur noch um die modellspezifischen Funktionen ergänzt werden. Die Darstellung grafischer Elemente, wie zum Beispiel Graphen, einer virtuellen Umgebung oder sogar Geoinformationen sind mittels Repast möglich. Parameteranzeigen und die Möglichkeit, Parameter über eine grafische Benutzeroberfläche zu setzen und zu verändern, sind in Repast ebenso vorhanden, wie ein Panel zur Steuerung der Simulation. Vgl. [Rep09].

4.3 Die Implementierung und Integration

Die zu implementierenden Komponenten ergeben sich direkt aus dem vorgegebenen Szenario. So sollen möglichst alle in diesem Szenario beschriebenen Fähigkeiten der Agenten sowie Kommunikationswege und Umgebungszustände in der Implementation Verwendung finden. Eine enge Anlehnung an das NetLogo-Beispielprogramm wird angestrebt. Dies soll später ermöglichen, Ergebnisse von Simulationsdurchläufen beider Implementationen miteinander zu vergleichen. So sollen Rückschlüsse auf Abweichungen vom vorgegebenen Szenario sowie das Aufdecken implementationsbedingter Unterschiede der Programme möglich werden. Zusätzlich zur eigentlichen Implementation soll eine Dokumentation erstellt werden, die es ermöglicht, die Struktur und den Ablauf des Programms zu verstehen. In der vorliegenden Arbeit wurde diese Programmbeschreibung in Form eines JavaDoc[2] erstellt und liegt nebst dem Programm dieser Arbeit auf einer Compact Disc bei. Sowohl das Programm als auch die Beschreibung sind zusätzlich im Internet[3] verfügbar. Im nachfolgenden Abschnitt werden die zu implementierenden Komponenten kurz erläutert, um einen Überblick zu geben. Nachfolgend wird in späteren Abschnitten auf wichtige Komponenten und ihre Funktionsweisen näher eingegangen.

4.3.1 Zu implementierende Komponenten

Um die Simulation steuern zu können, ist es notwendig, eine grafische Benutzeroberfläche vorzuhalten. Diese ermöglicht, die Simulation zu starten, zu pausieren oder zu stoppen. Auch soll sie die Möglichkeit bieten, Parameter zu setzen oder aber abzuändern. Die Benutzeroberfläche wird im Programm vom Simulationswerkzeug Repast zur Verfügung gestellt. Die in Repast vorgehaltene Komponente ermöglicht nicht nur, eine Simulation vollständig zu steuern, sondern es ist sogar möglich, Parameter einzelner Agenten während der Simulation zu inspizieren. Die für die Simulation notwendigen Agenten müssen die eingangs beschriebenen Eigenschaften erfüllen. Diese Agenten müssen zum Beginn der Simulation erzeugt werden. Eine Kommunikation mit EmIL-S muss ermöglicht werden, hierfür sind Methoden zu programmieren. Eine grafische Ausgabe sowohl in Form von Statusmeldungen als auch in Form von Graphen soll erzeugt werden. Eine spätere Ver-

[2] Bei JavaDoc handelt es sich um ein Werkzeug zur Generierung von Programmbeschreibungen, welches dem Java-SDK beiliegt. Vgl. [Sun09].

[3] Hyperlink:
`http://userpages.uni-koblenz.de/~emil/code_examples/CollaborativeWriting/`

knüpfung mit MEME ist angestrebt, jedoch nicht Bestandteil dieser Arbeit. MEME soll zu einem späteren Zeitpunkt die Simulation selbst steuern und überwachen können. Es ist notwendig, die Programmierung derart zu gestalten, dass die Parameter, die innerhalb der Simulation Verwendung finden, mittels geeigneter Methoden abgeändert und ausgelesen werden können. Zusätzlich zur grafischen Benutzeroberfläche soll eine Ausgabe wichtiger Statusmeldungen implementiert werden, die es ermöglichen soll, den Simulationsverlauf verfolgen zu können.

4.3.2 Entwicklung und Implementation der Komponenten

Da die zu entwickelnden Komponenten innerhalb des Simulationswerkzeuges Repast-J zur Ausführung kommen, müssen alle Komponenten den Konventionen von Repast-J entsprechen. Eine Simulation in Repast-J besteht aus zwei oder mehr Klassen. Vgl. [Rep08]:

- Die Modell-Klasse: In ihr werden die Simulationsparameter und der Ablauf der Simulation implementiert. Sie steuert während der Simulation den gesamten Ablauf.
- Die Agenten-Klasse: Sie bündelt all jene Tätigkeiten in Methoden, die das Verhalten der Agenten definieren.

Zusätzlich zu diesen beiden Klassen, lassen sich weitere implementieren, welche zum Beispiel Methoden und Eigenschaften einer Agenten-Umgebung oder aber Elemente der Benutzeroberfläche beinhalten. Die beiden Hauptklassen für das Modell und die Agenten beinhalten zwingend besondere Methoden, abhängig davon, welche Art von Modell erstellt wird. So enthält die Modell-Klasse, wenn sie eine Spezialisierung von „SimModelImpl“ ist, zwingend einige Operationen, die sie von der Klasse SimModelImpl erbt. Die wichtigsten sind:

- `setup()`, diese Operation wird aufgerufen, wenn der Setup-Knopf auf der Repast Benutzeroberfläche betätigt wird. Innerhalb dieser Operation sollten alle Initalisierungstätigkeiten abgearbeitet werden.
- `buildModel()`, diese Operation baut das Modell auf. Hier finden alle diejenigen Tätigkeiten statt, die für den späteren Ablauf der Simulation notwendig sind, zum Beispiel das Erzeugen der Agenten, das Initialisieren mit Werten oder ähnliches.
- `buildSchedule()`, hier ist es möglich, einen Zeitplan für den Ablauf der Simulation zu entwickeln.

Die Agenten-Klasse beinhaltet alle Elemente, die die Agenten direkt betreffen. Darin sind alle Tätigkeiten als Methoden definiert, die von den Agenten ausgeübt werden können. Hierbei gibt es keine genaue Definition, was diese Agentenklasse beinhalten muss, da dies immer abhängig vom jeweiligen Modell ist. Jedoch ist es ratsam, innerhalb der Agentenklasse eine Methode vorzuhalten, welche jedes Mal durchlaufen wird, wenn ein neuer Zeitschritt erfolgt. Auch sind bei der späteren Auswertung der Simulation Variablen zur individuellen Bestimmung einer Agenteninstanz von Vorteil. Das Simulationswerkzeug Repast lässt hier einen sehr breiten Spielraum für die Gestaltung der Agenten.
Die Kommunikation der Agenteninstanzen mit ihrem jeweiligen EmIL-S-Gegenstück erfolgt durch Nachrichten. Zu diesem Zweck wird eine Schnittstelle genutzt, die für Repast erstellt wurde.

4.3.3 Beschreibung des Programms

Das Programm wurde mit Hilfe der Entwicklungsumgebung Eclipse erstellt. Diese ermöglicht, das Repast-Simulationswerkzeug in der Art einzubinden, dass eine Ausführung der Simulation direkt aus der Entwicklungsumgebung heraus möglich ist. Das Programm kann jedoch auch von Repast direkt gestartet werden. Für diesen Zweck verfügt es über eine grafische Benutzeroberfläche zur Steuerung der Simulation. Über diese ist es auch möglich, Parameterwerte zu setzen und Parameter zu beobachten. In einem Ausgabefenster werden während der Simulation Ereignismeldungen ausgegeben. Hiermit lässt sich der Verlauf der Simulation verfolgen. Zusätzlich gibt es Graphen, in welchen ausgesprochene Sanktionen aufgezeichnet werden. Die graphischen Elemente sind in Abbildung 4.1 dargestellt. Das Programm beinhaltet die Klassen: WikiModel, WikiAgent und WikiGui. Die Klasse WikiModel steuert die Simulation und beinhaltet folgende Methoden:

- `setup()`, diese Methode setzt wichtige Variablen auf initale Nullwerte und initialisiert `agentlist` und `schedule`. Bei `agentlist` handelt es sich um eine `Arraylist`, in der Referenzen der Agenteninstanzen gespeichert werden. Dies ist notwendig, um die einzelnen Instanzen der Agenten im Verlauf der Simulation direkt ansprechen zu können. Auch wird innerhalb der Setupmethode die für EmIL-S notwendige XML-Konfigurationsdatei geladen.

- `begin()`, hier werden lediglich die Methoden `buildModel()` und `buildSchedule()` aufgerufen.

- `buildModel()`, die Agenten werden in dieser Methode erzeugt und Referen-

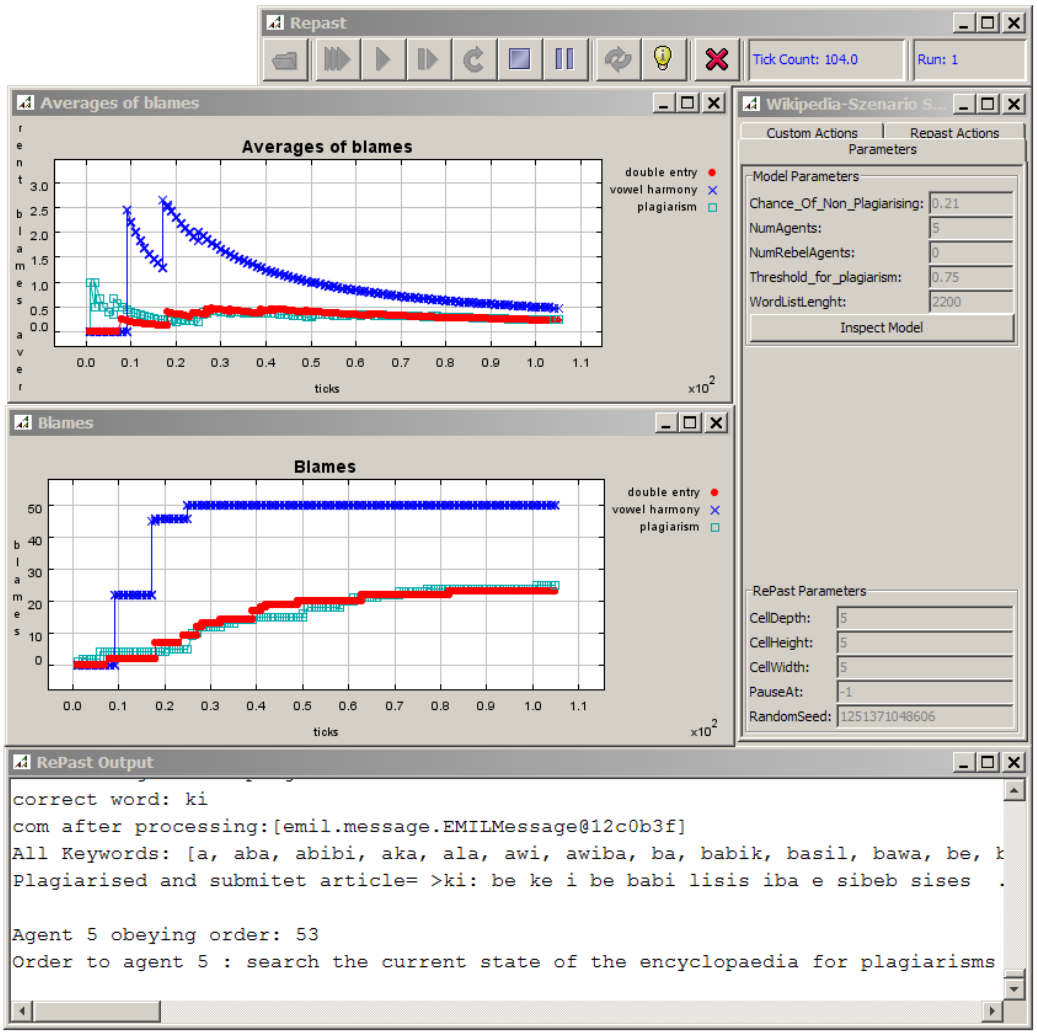

Abbildung 4.1: Repast Oberfläche

zen auf diese Agenteninstanzen werden in `agentlist` abgelegt. Die Anzahl der Agenten wird über eine Variable gesteuert. Nachdem die Agenten erzeugt wurden, wird eine Reportmethode innerhalb jedes Agenten aufgerufen. Diese soll dazu dienen zu überprüfen, ob die Agenten korrekt erzeugt wurden. Auch ließen sich hier eventuell vergebene initiale Werte der Agenten ausgeben.

- `buildSchedule()`, die Methode `buildSchedule` erzeugt eine Art Zeitplan, in welchem die Reihenfolge der Agenten aufgestellt wird. Die Agenten werden an dieser Stelle der Reihe nach aufgefordert, ihre Methode `step()` auszuführen. Zusätzlich werden jeweils die Graphen aktualisiert. Dies erfolgt innerhalb der Klasse

WikiStep. Diese wird jeweils zum Beginn eines jeden Repast-internen Zeitschrittes instanziiert. Die Methode buildSchedule beinhaltet hierzu die Zeile `schedule.scheduleActionBeginning(0, new WikiStep())`, welche bei Eintreten des Ereignisses `ActionBeginning` eine neue Instanz der Klasse WikiStep erzeugt. Das führt zu einem erneuten Durchlauf durch die Agenten. Hierdurch entsteht ein Kreislauf, der erst endet, wenn das Ereignis `ActionBeginning` nicht mehr ausgelöst wird. Zudem sind die Ereignisse `AtEnd` und `AtPause` zu nennen. `AtPause` ist das Ereignis, das zwischen jedem Zeitschritt ausgelöst wird, hier wird durch eine zusätzliche Methode nur eine Textzeile ausgegeben. Dies soll es einfacher machen, innerhalb der Programmausgabe die einzelnen Zeitschritte voneinander trennen zu können. Das Ereignis `AtEnd` wird ausgelöst, sobald der Nutzer der Simulation diese über die Schaltfläche Stop beendet. Hier wird über eine zusätzliche Methode dann eine Ausgabe innerhalb des Ausgabefensters erzeugt. Alle relevanten Daten werden in einer für den Leser ansprechenderen Form ausgegeben, als dies während des Verlaufes der Simulation möglich wäre.

Die Klasse WikiModel beinhaltet noch eine Reihe zusätzlicher Methoden, die notwendig sind, um Parameter über die grafische Benutzerschnittstelle festlegen zu können. Auch beinhaltet sie eine Reihe von sogenannten „getter“- und „setter“-Methoden, die benötigt werden, um Variablenwerte auszulesen und zu setzen.

Die zweite sehr wichtige Klasse ist WikiAgent. In dieser sind alle Methoden und Eigenschaften gebündelt, welche die Agenten betreffen. Sie beinhaltet eine Vielzahl von Methoden. Nachfolgend werden die Wichtigsten erläutert.

- Von zentraler Bedeutung ist die Methode `step()`. Sie wird von der Klasse WikiModel für jeden einzelnen Agenten in jedem Zeitschritt aufgerufen. Über diese Methode wird zentral der Ablauf der Tätigkeiten der Agenten gesteuert.
- `emilDispatcher()`, stellt eine Methode dar, in der die von EmIL-S empfangenen Nachrichten für den jeweiligen Agenten ausgewertet werden. Hierbei wird die Nachricht in ihre Bestandteile zerlegt; abhängig vom Inhalt werden dann weitere Methoden aufgerufen. An dieser Stelle wird die Verknüpfung zur zuvor beschriebenen initialen Konfiguration klar, denn die Nachrichten beinhalten die in dieser Konfiguration definierten Aktionen. Diese werden in Form eines Strings übergeben. So stellt der Nachrichteninhalt: „1“ die Aktion „A1“ dar, die den Agenten

auffordert ein neues Schlüsselwort zu erzeugen. Bei Erhalt der Nachricht „1“ wird daher innerhalb der Methode `emilDispatcher()` die Methode `keyword()` aufgerufen, welche ein neues Schlüsselwort für die spätere Verwendung erzeugt. Nachdem dieses neue Schlüsselwort erzeugt wurde, wird ein Event an EmIL-S gesendet, welches entweder Event „E11“ oder Event „E12“ ist, abhängig davon, ob das Schlüsselwort bereits in der Wikipedia als Schlüsselwort Verwendung gefunden hat, oder nicht. Dieses veranlasst wiederum EmIL-S erneut, Aktionen in Form von Nachrichten an den Agenten zu senden. In diesem Fall wären dies zum Beispiel Aktionen, die bestimmen, ob das erstellte Schlüsselwort als Schlüsselwort für einen neuen Artikel Verwendung finden soll oder ob ein zu erstellender Artikel zusätzlich zu einem bereits bestehenden Artikel in die Wikipedia eingestellt werden soll. Auch ist hier eine Aktion möglich, in der ein bestehender Artikel um neue Inhalte ergänzt wird. Die Methode `emilDispatcher()` stellt daher eine Art Schaltzentrale dar, in welcher der Ablauf der Methodenaufrufe abhängig von empfangenen Nachrichten spezifiziert ist. Nachdem eine Nachricht ausgewertet wurde, wird diese gelöscht. Dies wird durch die Datenstruktur „Queue“, in der eintreffende Nachrichten zwischengespeichert werden, in einer sehr einfachen Weise möglich. Die jeweils älteste Nachricht wird entnommen und die „Queue“ somit um eine Nachricht kürzer. In diesem Zusammenhang macht sich der Vorteil der Datenstruktur „Queue“ gegenüber anderen Listenformen besonders bemerkbar, denn Nachrichten von EmIL-S treffen zeitlich unbestimmt ein. Eine neue Nachricht wird einfach an die „Queue“ angehängt und somit werden die Nachrichten immer in der Reihenfolge abgearbeitet in der sie eingetroffen sind.

- eine sehr ähnliche Methode ist `AgentMsgComProcessing()`. Innerhalb dieser Methode werden Nachrichten in ähnlicher Form ausgewertet, wie dies in `emilDispatcher()` der Fall ist. Jedoch handelt es sich bei diesen Nachrichten um solche, die von anderen Agenten stammen. Das Wikipedia-Szenario beinhaltet nur eine Art von Nachrichten, die von Agent zu Agent gesendet werden können: „Sanktionen“. Diese werden in der NetLogo-Implementation als Reaktion der Agenten auf ein bestimmtes Ereignis versendet. So sendet ein Agent, der ein Plagiat innerhalb der Wikipedia entdeckt hat, an den Autor dieses Plagiates eine Nachricht vom Typ Sanktion. Der Agent, der diese Nachricht empfängt und auswertet, wird aufgrund dieser Nachricht lernen. Diese Form der Kommunikation wurde auch in der hier beschriebenen Repast-Umsetzung implementiert. Jedoch wurden diese Nach-

richten durch eine Erweiterung von EmIL-S obsolet. Die Sanktionen werden nun in Form von speziellen Nachrichten direkt über EmIL-S versendet. Es bedarf daher keiner direkten Agentenkommunikation auf Repast-Ebene mehr. Diese Methode wurde jedoch für andere Implementierungen und Erweiterungen im Programmcode belassen. Die mit dieser Methode versendeten Nachrichten enthalten den Zeitpunkt, zu dem eine anstößige Aktion ausgeübt wurde und die Information, um welche Aktion es sich handelt.

Die beschriebenen Methoden bilden ein Gerüst, durch welches der Aufruf weiterer Methoden definiert wird. Durch sie wird die sinnvolle Abfolge von Ereignissen und Aktionen erst möglich.
Zu ihnen gehören unter anderem folgende Methoden:

- `sendMessage`: Diese Methode wird von EmIL-S aufgerufen, um Nachrichten an den Agenten zu übergeben.
- Mittels `sendToEmil` ist es dem Agenten möglich, Nachrichten an EmIL-S zu senden.
- Auch sind mit `sendToAgent` Nachrichtensendungen an andere Agenten möglich. Auch diese Methode ist wie bereits beschrieben durch eine Erweiterung von EmIL-S überflüssig geworden.
- `keyword()` erzeugt ein neues Schlüsselwort,
- welches mittels `new_article(String my_keyword)` in einem neuen Artikel Verwendung finden kann. Die Agenten erzeugen hierfür eine Reihe neuer Wörter, welche sie zu einem oder mehreren Sätzen zusammensetzen. Das Schlüsselwort wird dann an den Anfang des Artikels gesetzt. Auch wird das Schlüsselwort in einer Agenten eigenen Liste abgelegt, um später identifizieren zu können, wie häufig eigene Artikel zitiert bzw. verlinkt wurden.
- Mit der Methode `find_double_entries()` ist es den Agenten möglich, die Wikipedia nach Artikeln mit gleichem Schlüsselwort zu durchsuchen.
- `find_vowel_harmony_violation()` ermöglicht es, Artikel zu entdecken, die nicht der so genannten „vowel-harmony“ entsprechen. Hierbei handelt es sich um eine eingangs erwähnte Sprachvorgabe, der Wörter entsprechen sollten. Werden Wörter innerhalb eines Artikels gefunden, die nicht dieser Vorgabe entspre-

chen, so werden sie in der Art umgeformt, dass sie der Sprachvorgabe entsprechen. Zusätzlich wird der Autor, welcher ein oder mehrere nicht konforme Wörter verwendet hat, für diese Tat mit einer Sanktion belegt. Zu beachten ist hier, dass durch die oben erwähnte Erweiterung des Szenarios alle Agenten solche Wörter, welche nicht ihrer eigenen „vowel-harmony“ entsprechen, korrigieren. Diese Korrektur wird durch Agenten der jeweils anderen Klasse wieder umgekehrt. Auch die Agenten der Klasse „Rebellen“ belegen die Autoren vermeintlich falsch geschriebener Wörter mit einer Sanktion. An dieser Stelle könnte es zu Problemen kommen, da ein Autor für ein Wort sanktioniert werden kann, das er zwar richtig geschrieben hat, das jedoch von einem Agenten der anderen Klasse umgeschrieben wurde. In einem solchen Fall würde er von einem Agenten seiner eigenen Klasse dafür bestraft werden, dass eines seiner richtig geschriebenen Wörter von einem Agenten der anderen Klasse „korrigiert“ wurde.

- Die Agenten können die Wikipedia nach Artikeln durchsuchen, in denen ein Wort auftaucht, das sie selbst als Schlüsselwort benutzt haben. Dies ermöglicht es, eine Art Verweis auf einen bestehenden Artikel zu simulieren. Autoren deren Artikel zitiert beziehungsweise verlinkt werden, können mittels `count_links()` die Anzahl der Artikel ermitteln, die auf einen ihrer Artikel verweisen.

- Plagiate können mit der Methode `plagiarise()` erstellt werden.

- Sie können jedoch unter Umständen mit Hilfe von `compare()` wieder aufgefunden werden. Die Methode `compare()` vergleicht hierfür einen beliebigen Artikel mit allen anderen innerhalb der Wikipedia. Ist die Ähnlichkeit zweier Artikel höher als ein durch die Variable `threshold_for_plagiarism` vorgegebener Wert, so wird der jüngere der verglichenen Artikel aus der Wikipedia entfernt und der Autor dieses Artikels erhält eine Nachricht vom Typ Sanktion.

Die Klasse WikiAgent beinhaltet zudem noch eine Vielzahl weiterer Methoden, die zum einen von den beschriebenen Methoden benötigt werden, zum anderen rudimentäre Aufgaben erfüllen.

Als dritte Klasse ist noch WikiGui zu nennen. Sie ergänzt die Klasse WikiModel um Elemente der grafischen Benutzeroberfläche. Sie wurde erstellt, um diese Elemente von der eigentlichen Steuerung der Simulation zu trennen und die Übersichtlichkeit innerhalb der Klasse WikiModel zu erhalten. WikiGui enthält Methoden zur Erzeugung von Fenstern,

in denen Graphen abgebildet werden. Denkbar wäre es, auch weitere grafische Elemente zu erzeugen, zum Beispiel Fenster, in denen der aktuelle Stand der Wikipedia dargestellt wird oder die über den Zustand bestimmter Variablen informieren. Dies wurde für die hier vorliegende Arbeit jedoch nicht umgesetzt, da eine automatisierte Auswertung der Daten zu einem späteren Zeitpunkt mittels MEME erfolgen soll.

4.3.4 Kommunikation mit EmIL-S

Die Kommunikation mit EmIL-S geschieht mit einer in EmIL-S definierten Schnittstelle namens `emil.agentIEMILAgentWrapper`, die innerhalb der Agenten-Klasse genutzt wird. Durch diese Schnittstelle wird innerhalb des Konstruktors der Agentenklasse zunächst ein EmIL-S-Agent als Pendant zum Repast-Agenten erzeugt und eine Referenz zu diesem innerhalb der jeweiligen Agenteninstanz gespeichert. Dies geschieht mit der Befehlszeile: `emil.Controller.getInstance().addAgent(agentType, agentID, refAgentWrapper)`. Der `agentType` ist zuvor innerhalb der XML-Konfigurationsdatei zu definieren. Die `agentID` ist eine eindeutige Bezeichnung für die Agenten. Der `refAgentWrapper` ist eine Referenz auf das aktive Agentenobjekt. Im Programm, welches während dieser Arbeit implementiert wurde, lautet diese Zeile daher: `emIlRef = Controller.getInstance().addAgent(„author", ID, this);`. Nach diesem Schritt ist es nun dem Repast-Agenten möglich, Nachrichten an sein EmIL-S-Pendant zu senden und auch Nachrichten von diesem zu erhalten. Das geschieht mit zwei weiteren Methoden. Zum Senden an EmIL-S wird die Methode `processMessage` verwendet, die als Argument eine Nachricht vom Typ `IEMILMessage` erwartet. Nachrichten von EmIL-S hingegen werden über die Methode `sendMessage`, die wiederum ein Argument vom Typ `IEMILMessage` erwartet, von EmIL-S an sein Repast-Pendant gesendet. An diesem Punkt ist darauf zu achten, dass die `sendMessage`-Methode von EmIL-S aufgerufen wird und nicht vom Repast-Agenten. An dieser Stelle müssen dann sinnvolle Konstrukte vorgehalten werden, um diese Nachrichten zu speichern oder weiterzuverarbeiten. In der Wikipedia-Implementierung geschieht dies mittels einer `Queue`, die die eintreffenden Nachrichten von EmIL-S zwischenspeichert. Der betreffende Agent kann dann diese Liste zu gegebener Zeit abarbeiten. Die Nachrichten, die ausgetauscht werden, sind vom Typ IEMILMessage. Dieser lag in einer frühen EmIL-S-Version in folgender Form vor: `EMILMessage(Object sender, Object recipients, Modal modal, IContent<?>content)`. Hierbei sind der Sender und der Empfänger jeweils Referenzen auf die Instanz des jeweiligen Agenten. In einer

späteren Version von EmIL-S wurden hier noch Felder geschaffen, um sogenannte „norm invocation messages“ versenden zu können. Diese erlauben es unter Anderem, einem anderen Agenten direkt Nachrichten zukommen zu lassen. Das dritte Feld innerhalb der Nachricht ist der Modal. Hierbei handelt es sich um einen der Modale, die in Kapitel 3.1 beschrieben wurden. Zum Zeitpunkt der Implementierung des Wikipedia-Programms war es EmIL-S jedoch noch nicht möglich, alle Modale zu interpretieren. Das letzte Feld, welches von der `EMILMessage`-Klasse erwartet wird, ist ein Inhalt. Dieser kann in unterschiedlichen Formen vorliegen, zum Beispiel als `StringContent`, welcher sowohl einen Text als auch eine Zeitangabe erwartet. Der Text ist hier ein Event, welches von EmIL-S mit Hilfe der XML-Konfiguration interpretiert wird. Die mitgegebene Zeitangabe wird von EmIL-S benötigt, um eine Zuordnung zu einem früher ausgelösten Event erstellen zu können. Ein Ablauf für eine solche Kommunikation könnte sich folgendermaßen präsentieren:

1. Zum Zeitpunkt t=3 trifft EmIL-S aufgrund eines vorher ausgelösten events E1 die Entscheidung, action „A4“ auszulösen.

2. Agent A wird daher im selben Zeitschritt t=3 von EmIL-S angewiesen, ein Plagiat zu erstellen (A4).

3. Agent A arbeitet diesen Befehl ab, erstellt ein Plagiat und setzt als Erstellungszeitpunkt den Zeitwert t=3.

4. Agent B wird zum Zeitpunkt t=27 von EmIL-S angewiesen, nach Plagiaten zu suchen.

5. Agent B arbeitet diesen Befehl ab, findet das von Agent A erstellte Plagiat und sendet eine Nachricht an sein EmIL-S-Gegenstück. Diese Nachricht enthält das event, welches das Auffinden eines Plagiates definiert. Zusätzlich wird im Content-Teil das event mitgeliefert, welches die Erstellung des Plagiates ausgelöst hat (E1). Dieses wird ergänzt durch den im Plagiat gesetzten Erstellungszeitpunkt (t=3) und den Agenten, der das Plagiat erstellt hat. Diese Nachricht wird von EmIL-S ausgewertet und führt dazu, dass eine Aktion vom Typ „norm-invocation“ ausgelöst wird.

6. EmIL-S wertet innerhalb dieser Aktion die Nachricht aus. Es ist EmIL-S somit möglich, festzustellen, dass die Entscheidung, welche zum Zeitpunkt t=3 für den Agenten A getroffen wurde, von anderen Agenten missbilligt wurde. Daraufhin ändert EmIL-S die Wahrscheinlichkeiten innerhalb des event-action trees von Agent

A entsprechend ab. Der Grad dieser Änderung und auch die Richtung hängen vom verwendeten Lernalgorithmus ab.

Im Verlauf der Implementation wurden verschiedene Punkte, welche die Kommunikation betreffen, rege im Projektteam diskutiert. So wurde angedacht, die Trennung von physikalischer Ebene und kognitiver Ebene strikt beizubehalten. Hierfür wurde in die Implementierung die Möglichkeit eingebaut, dass Agenten untereinander kommunizieren können. Denn ohne diesen Kommunikationskanal wäre es ihnen sonst nicht möglich, einander Sanktionen zu senden. Dieser Grundsatz wurde jedoch im späteren Verlauf aufgebrochen. So gibt es innerhalb von EmIL-S drei Arten von „norm invocation“ Nachrichten.

- Ein Agent der physikalischen Schicht sendet via EmIL-S eine Nachricht an einen anderen Agenten. Dies geschieht mittels einer „norm invocation“ Nachricht, die einen anderen Agenten als Ziel beinhaltet. Bei den erwähnten Sanktionen handelt es sich um einen solchen Fall.

- Ein Agent befindet sich in einer Beobachterrolle und liest Nachrichten an einen anderen Agenten mit. Als Reaktion auf diese mitgelesenen Nachricht sendet der Agent eine „norm invocation“ Nachricht an den anderen Agenten. Hier sind unterschiedliche Szenarien denkbar. So können hier dem Agenten zum Beispiel Verhaltensvorschläge unterbreitet werden.

- Eine dritte Möglichkeit für „norm invocation“ Nachrichten ist, diese in eine action zu integrieren. Diese action wiederum wird durch eine Nachricht mit Modal Assertion ausgelöst, welche von der physikalischen Schicht an EmIL-S gesendet wird und dann innerhalb von EmIL-S zum Versenden der „norm invocation“ Nachrichten führt.

Damit ist es einem Agenten möglich, an sein EmIL-S-Gegenstück zu berichten, dass er ein Ereignis wahrgenommen hat, welches einen anderen Agenten betrifft. Hierdurch werden innerhalb von EmIL-S Aktionen ausgelöst, die Auswirkungen auf andere Agenten haben. Mittels dieser Konstrukte wird eine größere Funktionalität erreicht. Durch eine zusätzliche Möglichkeit innerhalb von EmIL-S, Nachrichten direkt an andere Agenten zu senden, wird eine große Menge an Kommunikation auf der physikalischen Ebene eingespart. Es zeigt sich eine andere Struktur der Kommunikation; es ergeben sich zwei Klassen von Nachrichten. Die erste Nachrichtenart wird von aktiv agierenden Agenten ausgelöst. Diese Nachrichten führen im weiteren Verlauf wiederum zu Aktionen auf der physikalischen

Ebene. Die zweite Art der Nachrichten wird von Agenten in einer Beobachterrolle versendet. Sie führen dann innerhalb von EmIL-S zu „norm-invocation“-Aktionen, welche sich nicht direkt in Aktionen auf der physikalischen Ebene auswirken.
Die Frage, welche Nachrichtentypen jeweils Verwendung finden sollen, ist nicht für jedes Szenario leicht zu beantworten. So gibt es im Wikipedia-Szenario nur eine Art von Agenten und diese haben auch nur eine Rolle. Agenten sind hier immer aktive Agenten. Während einer (aktiven) Tätigkeit können sie allerdings Feststellungen machen, die andere Agenten betreffen. So soll sich die Erkenntnis, dass ein anderer Agent ein Plagiat erstellt hat, in Form einer Sanktion widerspiegeln, aber auch in Form einer positiven Auswirkung für die Tatsache des Auffindens. Der betroffene Agent soll sein Verhalten anpassen können. All dies wird durch das neue Konstrukt gewährleistet. Hierbei übt im Wikipedia-Szenario ein Agent zwar eine aktive Tätigkeit aus, nämlich das Suchen nach Plagiaten. In dem Augenblick jedoch, in dem er ein solches entdeckt, ist er für diesen Moment ein Beobachter seiner Umgebung. In anderen Szenarien ist es durchaus möglich, Agenten zu implementieren, die nur eine der beiden Rollen einnehmen.

4.3.5 Einspielen der Konfiguration

Die initiale Konfiguration, die in der vorliegenden Arbeit in Form einer XML-Datei erstellt wurde, wird bei Ausführung der Simulation geladen und ausgewertet. Dies geschieht mittels der Programmzeile: `Controller.getInstance();`, auf welche: `Controller.initializeController(new File("C:\\Pfad\\Konfiguration.xml"))` folgt. Damit wird EmIL-S die initiale Konfiguration übergeben. EmIL-S wertet diese aus und führt einige Schritte zur Bereinigung dieser Konfiguration durch. So werden fehlende Wahrscheinlichkeitswerte an Aktionen ergänzt, wenn diese alleine in einer action-group auftreten. Eventuelle Fehler in der XML-Datei werden im selben Schritt aufgezeigt. Nach dem Einspielen der Simulation ist EmIL-S bereit, auf Nachrichten aus der physikalischen Schicht zu reagieren.

4.3.6 Testläufe und Korrekturen

Die erstellten Methoden wurden bereits während des Implementationsprozesses zahlreichen Tests unterzogen. So wurden sie auf funktionale Korrektheit hin überprüft und die auftretenden Fehler korrigiert. Hierzu wurden die Methoden mit entsprechenden Eingabeparametern aufgerufen und die gelieferten Ergebnisse mit den Anforderungen verglichen. Diese Komponententests ermöglichten eine Vielzahl von Fehlern zu identifizieren und zu

korrigieren. Jedoch garantieren sie nicht, dass alle Fehler im Programmcode gefunden wurden. Nachdem EmIL-S und das implementierte Programm zum ersten Mal miteinander in einer Simulation zum Ablauf gebracht wurden, zeigten sich andere Fehler. So interpretierte EmIL-S das Fehlen von Wahrscheinlichkeiten an den Aktionen nicht korrekt und sendete keine Antworten zu empfangenen events. Hier wurde eine schnelle Problemlösung in Form der oben genannten Bereinigung der Konfigurationsdatei innerhalb von EmIL-S implementiert. Jedoch wurde durch diesen Fehler erneut die Frage aufgeworfen, ob die optionale Notation von Wahrscheinlichkeiten nicht durch entsprechende Änderungen am XML-Schema ersetzt werden sollte, beispielsweise durch eine verpflichtende Angabe der Wahrscheinlichkeiten. Hierdurch ergäben sich jedoch Einschränkungen, die in zukünftigen Weiterentwicklungen zu Problemen führen könnten. Daher wurde die Lösung dieser Frage auf einen späteren Zeitpunkt vertagt.
Weitere Integrationstests offenbarten weitere Fehler, zum Beispiel in der Nachrichtenübermittlung oder der Berechnung der Aktionen, welche jedoch schnell behoben werden konnten. Nachdem EmIL-S um die Funktionalität erweitert wurde, „norm-invocation“-Nachrichten auszuwerten und auch ein simpler Lernalgorithmus Verwendung findet, wurde es möglich, das Zusammenspiel aller Komponenten zu testen. So reagiert EmIL-S auf den Empfang einer „norm-invocation“-Nachricht (kurz NI-Nachricht) durch Anpassen von Wahrscheinlichkeiten innerhalb der event-action trees. Diese Anpassungen werden als Statusmeldungen innerhalb des Repast-Ausgabefensters angezeigt und erlauben es somit, die getätigten Änderungen während der Simulation zu verfolgen. Zuerst wurde nach Durchführung einiger Simulationsläufe die von EmIL-S erstellte Log-Datei zusammen mit den Daten des Repast-Ausgabefensters ausgewertet. Besonderes Augenmerk lag hier auf den von Agenten an EmIL-S gesendeten NI-Nachrichten. Diese wurden von EmIL-S ausgewertet und bewirkten dann einen Lernprozess. Dieser Prozess wird durch das nachfolgende Beispiel verdeutlicht: Zum Zeitpunkt T=33.0 erhält Agent 3 den Auftrag, die Wikipedia nach doppelten Artikeln zu durchsuchen. Dieser Auftrag wird von dem Agenten abgearbeitet und es werden zwei Artikel zum selben Schlüsselwort in der Wikipedia gefunden. Agent 3 identifiziert nun den jüngeren der beiden Artikel und sendet über EmIL-S eine NI-Nachricht an den Autor des Artikels. Dies wird innerhalb des Repast-Ausgabefensters mit der Zeile:

```
Time: 33.0 NI-event (Sanktion for A3) to agent: 2
for action at time: 22.0
```

angezeigt. Hierbei zeigt die Zahl 33.0 den aktuellen Zeitpunkt an, zu dem der doppelte Artikel gefunden wurde. Der betroffene Agent ist Agent 2 und dieser hat zum Zeitpunkt 22.0 den doppelten Artikel erstellt. Die hier gesendete NI-Nachricht enthält noch die action, welche zum Zeitpunkt 22.0 den anstößigen Vorgang ausgelöst hat. Dies ist notwendig, da Agent 2 zum Zeitpunkt 22.0 mehr als eine action ausgeführt haben kann. Die mitgesendete action innerhalb der NI-Nachricht ermöglicht es EmIL-S, zielgenauer den entsprechenden event-action tree anzupassen. Dies wird von EmIL-S innerhalb einer Log-Datei dokumentiert und zeigt sich in folgenden Zeilen:

```
-------- NORM INVOCATION FOR AGENT 2 AT TIME 33.0--------
EVENT: C=Explicit NI M=SANCTION P=-1.0
ENTRY=E1 VAL_T=22.0 VAL_CONT=A3
RULE=emil.agent.rule.EventActionTree@17494c8
LEARN ACTION=A3 OLD_PROB=0.5 NEW_PROB=0.25
```

EmIL-S hat also die NI-Nachricht empfangen und sie einem bestimmten Agenten (Agent 2) und einer bestimmten Regel, welche im Feld „Rule“ angezeigt wird, zugeordnet. Der auslösende event war E1. Dies wurde von EmIL-S anhand der betroffenen action A3, welche mitgesendet wurde, ermittelt. Es handelt sich um eine Sanktion und diese ist mit „-1.0“ gewichtet. EmIL-S hat als Reaktion auf diese NI-Nachricht die Regel angepasst indem der Wahrscheinlichkeitswert von ursprünglich „0.5“ auf nun „0.25“ angepasst wurde. Die Wahrscheinlichkeit, dass die anstößige action in selber Form wieder durchgeführt wird, ist also um die Hälfte vermindert worden. Analog zu diesem Beispiel verhält es sich mit allen anderen sanktionierbaren Aktionen.
Während der Testläufe offenbarten sich noch einige Fehler innerhalb der Repast-Implementierung. So wurden Zeitpunkte falsch übertragen und führten auf Seiten von EmIL-S zu Fehlern. Dies konnte jedoch schnell behoben werden. Auch wurde eine Schwierigkeit aufgezeigt, die auftritt, wenn ein Agent eine NI-Nachricht an sich selbst sendet. Dass ein Lernprozess stattfindet, lässt sich daran erkennen, dass bei einer Vielzahl von Testläufen wiederkehrende Muster innerhalb der Graphen erkennbar sind. So flacht die Kurve, welche die Anzahl von negativen Sanktionen für das Verwenden von Wörtern anzeigt, welche nicht der vowel-harmony entsprechen, im Verlauf der Simulation erkennbar ab. Im späteren Simulationsverlauf werden keine Wörter mehr verwendet, die nicht der vowel-harmony entsprechen. Dieser Umstand kann jedoch nur durch eine Anpassung der Wahrscheinlichkeiten innerhalb der event-action trees der Agenten bewirkt worden sein. An dieser Stelle hat also Lernen stattgefunden. Eine weitere Möglichkeit, die diesen

Sachverhalt erklären würde, wäre das verminderte Überprüfen der Wikipedia auf falsch geschriebene Wörter. Auch dies würde auf einen Lernprozess hindeuten. Bei genauerer Auswertung der Daten zeigt sich, dass das Überprüfen der Wikipedia nicht der Auslöser für die charakteristischen Graphen ist.

4.4 Ergebnisse

Nachdem alle Testläufe abgeschlossen sind, ist es nun möglich, gezielt Simulationsläufe durchzuführen. Da es bisher noch nicht möglich war, die Komponente MEME mit in die Simulationsumgebung zu integrieren, wurden die Simulationsläufe einzeln durchgeführt. Nachdem nun durch Analyse des Ausgabefensters und der Log-Datei von EmIL-S nachgewiesen wurde, dass Lernen in der gewünschten Form stattfindet, ist es möglich, eine Vielzahl von Simulationsläufen durchzuführen und anhand der Graphen Rückschlüsse auf den Lern- und Normentstehungsprozess zu ziehen.
Bereits bei den ersten Simulationsläufen zeigen sich Auffälligkeiten innerhalb der abgebildeten Graphen. Diese sind in Abbildungen 4.2 und 4.3 dargestellt. So flachte die Kurve, welche den Durchschnitt der sanktionierten Plagiate anzeigt (hier in der Farbe türkis) erkennbar ab. Auch zeigte sich innerhalb des Graphen, der die Anzahl der jeweiligen Sanktionen anzeigt (Abbildung 4.3) nach einer gewissen Zeit eine Glättung inerhalb des jeweiligen Graphen. Es kamen also keine oder nur noch vereinzelte Sanktionen hinzu. Der Graph, der die Sanktionen für das Auffinden von Wörtern, die nicht der vowel-haromny entsprechen, anzeigt und in Abbildung 4.3 in blau dargestellt ist, steigt zunächst recht schnell an, um dann in eine gleichbleibende Grade überzugehen. Dies bedeutet, dass keine neuen Sanktionen hinzukommen. Diese Abflachung bleibt auch erhalten, wenn die Anzahl der Agenten erhöht wird. Werden mehr Agenten eingestellt, flacht die Kurve später ab. Bei sehr wenigen Agenten flachen die Kurven hingegen sehr schnell ab. Sind wenige Agenten in der Simulation eingestellt, scheint die Wahrscheinlichkeit höher, dass die von ihnen begangenen sanktionierbaren Aktionen entdeckt werden. Sind mehr Agenten vorhanden, so gehen ihre Taten in der Masse der begangenen Delikte eher unter. Somit geht bei wenig eingestellten Agenten das Lernen jedes Einzelnen schneller vonstatten als bei einer großen Zahl von Agenten. Der Graph, der die Sanktionen für missachtete vowel-harmony anzeigt, verhält sich ein wenig anders als die beiden anderen. So geht dieser Graph bereits nach kurzer Zeit in eine Gerade über und ändert sich im weiteren Verlauf nicht mehr. Dies erscheint zunächst überraschend, wird jedoch bei Analyse des Logfiles und der Repast-Ausgabe erklärbar. So wird bei der Suche nach falsch geschriebenen Wör-

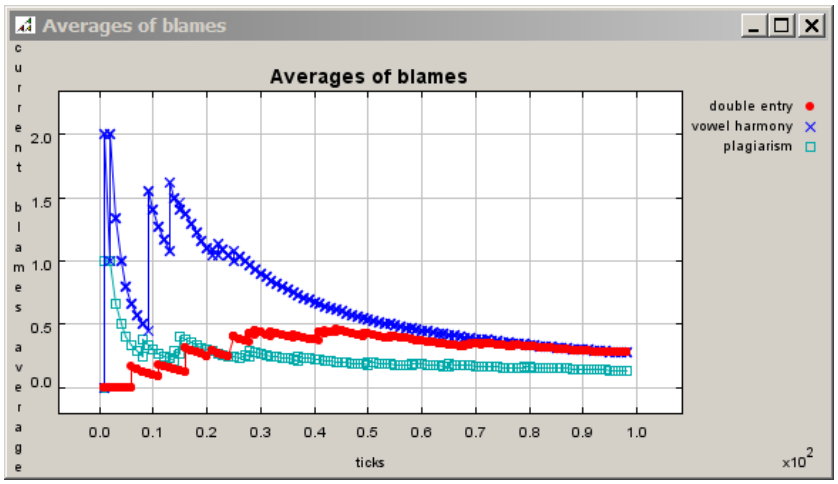

Abbildung 4.2: Durchschnitt

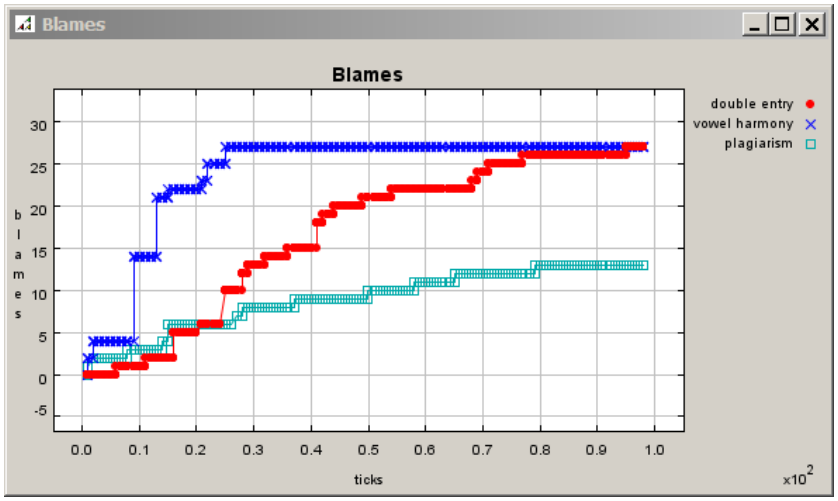

Abbildung 4.3: Sanktionen

tern für jedes falsch geschriebene Wort eine Sanktion ausgesprochen. Da die Agenten zu Beginn der Simulation noch sehr häufig Wörter falsch schreiben, kommt es bei Entdeckung dieser Wörter zu einer sehr großen Anzahl von Sanktionen. Durch diese Masse an Sanktionen passen die Agenten die Wahrscheinlichkeiten innerhalb der event-action trees für diesen Fall sehr stark an. Die Wahrscheinlichkeit sinkt daher beinahe auf 0. Somit erzeugen die Agenten während der Simulation keine falsch geschriebenen Wörter mehr. Ein wenig anders verhält es sich für die beiden anderen Graphen. Die abgebildete rote Kurve zeigt die Anzahl der ausgesprochenen Sanktionen für das doppelte Einstellen eines Arti-

kels. Diese Kurve zeigt im Vergleich zu den beiden anderen keine so starke charakteristische Abflachung im Verlauf der Simulation. Sie steigt zunächst eher stetig an, um dann später leicht abzuflachen. Dies mag im ersten Augenblick auffällig erscheinen. In diesem Fall handelt es sich jedoch nicht um einen Fehler. So werden zumeist alle von den Agenten doppelt eingestellten Artikel im Verlauf der Simulation gefunden und sanktioniert. Die Agenten erzeugen nach diesen Sanktionen deutlich weniger doppelte Artikel. Allerdings tauchen auch danach noch Sanktionen für doppelt eingestellte Artikel auf. Diese Artikel wurden zum Teil jedoch nicht bewusst als doppelte Artikel in die Wikipedia gestellt. Vielmehr können unter bestimmten Umständen auch Plagiate als doppelte Artikel in der Wikipedia auftauchen. Die hierbei ausgesprochenen Sanktionen können allerdings keiner Aktion zugeordnet werden, in solch einem Fall findet kein Lernen statt. Agenten, die nach doppelten Einträgen suchen, treffen daher immer wieder auf Plagiate, die auch als doppelte Artikel in der Wikipedia stehen. Dieser Umstand könnte durch eine Erweiterung des Szenarios behoben werden. So könnte der Ersteller eines Plagiates noch prüfen, ob zum verwendeten Schlüsselwort schon ein Artikel vorhanden ist und das Plagiat oder das verwendete Schlüsselwort verwerfen. Dies wurde in einer abgeschwächten Form auch im Programm umgesetzt. Jedoch war es nicht möglich, diesen Fall komplett zu verhindern, da dies die Performance des Programms zu stark beeinträchtigen würde. Die dritte Kurve, in Abbildung 4.3 in türkis dargestellt, zeigt die Anzahl der Sanktionen für aufgefundene Plagiate. Auch diese steigt zunächst an, um dann leicht abzuflachen. Dies lässt sich nur durch eine Analyse der Repast-Ausgabe erklären. So werden im Verlauf der Simulation eine Vielzahl von Plagiaten erzeugt. Von diesen Plagiaten wird jedoch nur ein Bruchteil bei späteren Suchläufen aufgefunden und sanktioniert. Dies ist dem Umstand zuzuschreiben, dass Agenten bei der Suche nach Plagiaten stets einen Artikel auswählen und diesen mit allen anderen innerhalb der Wikipedia vergleichen, hierbei werden zwangsläufig nicht alle in der Wikipedia enthaltenen Plagiate aufgefunden. Zwar findet auch hier ein recht langsam verlaufender Lernprozess statt, der dazu führt, dass im Verlauf der Simulation deutlich weniger Plagiate erstellt werden, jedoch werden im späteren Teil der Simulation immer wieder Plagiate aus früheren Phasen des Simulationslaufes gefunden, welche bislang in der großen Menge von Artikeln unentdeckt geblieben sind. Da diese nicht oder erst spät gefundenen Plagiate zum Teil auch doppelte Artikel sind, verfälschen sie auch den Graph für doppelt eingestellte Artikel über einen längeren Zeitraum. Bei Simulationsläufen, in welchen für die Sanktionen Parameterwerte gewählt wurden, die auf den Lernprozess sehr große Auswirkungen hatten, wurde beobachtet, dass die Graphen sich früher stabilen Geraden annähern, als in den oben beschriebenen Simulationsläufen.

4.4.1 Ergebnisse der Szenario-Erweiterung

Die in Abschnitt 3.1.3 beschriebene Erweiterung des Szenarios sollte zeigen ob Rebellen-Agenten, welche die vowel-harmony in umgekehrter Weise verwenden als die normalen Agenten, dazu führen, dass andere Normen entstehen. Hierbei gab es, in der Projektgruppe, unterschiedliche Theorien darüber, wie diese Rebellen sich in der Simulation auswirken würden. So wurde vermutet, dass es eventuell zu zyklischen Bewegungen innerhalb des Graphen kommen könnte. Oder, dass eine der beiden Gruppen zur anderen überwechselt und sich wieder eine stabile Kurve ergibt. Auch andere Ergebnisse waren denkbar. Nach Durchführung einiger Simulationsläufe mit gleicher Anzahl Rebellen-Agenten und normalen Agenten zeigt sich jedoch, dass der Graph für die vowel-harmony ein beinahe chaotisches Verhalten zeigt. Der Graph ist in Abbildung 4.4 in blau abgebildet. Er steigt

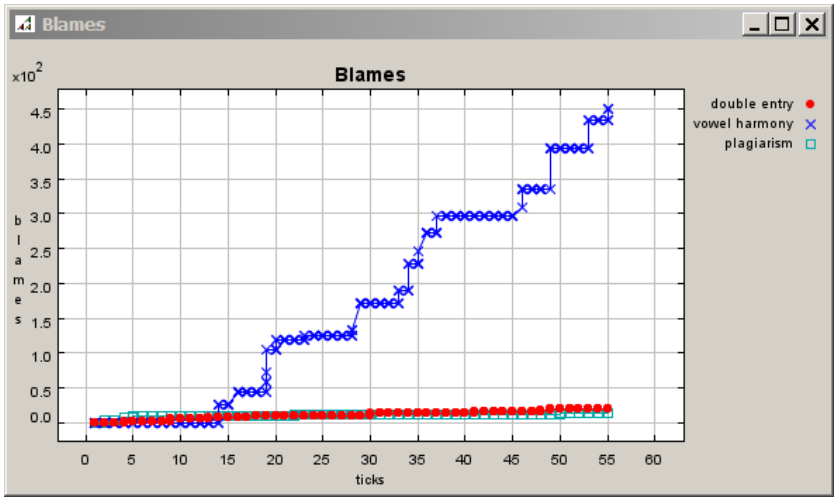

Abbildung 4.4: Sanktionen / Lauf mit Rebellen und normalen Agenten

stetig an, zeigt hierbei aber auch leichte stufenartige Strukturen. Dies lässt sich durch das Verhalten der Agenten erklären. So sanktionieren normale Agenten die jeweils von der Rebellengruppe verwendeten falsch geschriebenen Wörter und korrigieren diese. Finden nun Agenten der Rebellengruppe diese korrigierten Wörter, so senden sie Sanktionen an die ursprünglichen Ersteller dieser Wörter und korrigieren die Wörter wieder in umgekehrter Weise. Hierbei entstehen ständig Wörter, welche für die jeweils andere Gruppe falsch geschrieben sind. Die ausgesprochenen Sanktionen wiederum wirken sich nicht so aus, dass eine der beiden Gruppen nach einer Lernphase die Philosophie der anderen Gruppe übernehmen würde. Dies ist dem Umstand zuzurechnen, dass die Sanktionen immer einer bestimmten Aktion zugerechnet werden, im Falle von falsch geschriebenen

Wörtern der Aktion A2. Diese ist sowohl bei den normalen Agenten als auch bei den Rebellen identisch und führt dazu, dass Wörter, welche als falsch erkannt wurden, nicht mehr in die Wikipedia eingefügt werden. Stattdessen werden neue Wörter erzeugt, welche der jeweiligen Philosophie entsprechen. Den Agenten ist es also einfach nicht möglich, die Philosophie der anderen Gruppe anzunehmen. Dies zeigt sich auch, wenn nur ein Rebell und eine Vielzahl normaler Agenten in einem Simulationslauf verwendet werden. Auch hier zeigt sich ein chaotisches Verhalten des Graphen. Jedoch gibt es mehr stabile Stellen, da der Rebell nicht ständig neue Artikel erzeugt und es daher auch nicht ständig zu falsch geschriebenen Wörtern kommt. Würde den Agenten die Möglichkeit eingeräumt, sich ganz bewusst dafür zu entscheiden, falsche Wörter zu erzeugen, so wäre es ihnen möglich, zur anderen Gruppe überzuwechseln. Durch diese Programmänderung würde sich dann eventuell ein stabiles Verhalten herausbilden. Hierfür wären jedoch grundlegende Änderungen sowohl an der Konfiguration als auch am Programm notwendig.

4.4.2 Vergleich NetLogo- und Repastimplementation

Vergleicht man nun die Ergebnisse von Simulationsläufen aus der NetLogo-Implementierung mit der Repast-Implementierung, so finden sich hier eine Menge Ähnlichkeiten. Augenfällig wird dies bei einem direkten Vergleich der Graphen, wie dies mittels der Abbildungen 4.5, 4.6 und 4.7 möglich ist. Auffällig hierbei ist, dass die Graphen, die die vowel-harmony anzeigen, sich sehr ähnlich verhalten. Die vowel-harmony wird in Abbildung 4.7 in blau und in Abbildung 4.6 in der Farbe grün dargestellt. So treten die hierfür ausgesprochenen Sanktionen zu Beginn des Simulationslaufes in beiden Implementierungen sehr häufig auf, um dann stark nachzulassen und später kaum bis gar nicht mehr aufzutreten. Die Kurven, die Sanktionen für Plagiate anzeigen, sind nicht sehr ähnlich. Sie sind in Abbildung 4.7 in türkis und in Abbildung 4.6 in blau dargestellt. So steigt diese in der Repast-Implementierung viel stärker und länger an als dies in NetLogo der Fall ist. Hier scheint es deutliche Auswirkungen des unterschiedlichen Lernverhaltens zu geben. Die dritte Kurve, jeweils in rot dargestellt, welche die Anzahl der Sanktionen für entdeckte doppelte Artikel anzeigt, ist wiederum recht ähnlich in beiden Implementationen. So tritt in beiden Programmen der Fall auf, dass ein Plagiat auch ein doppelter Artikel sein kann. Auch gibt es hier in der NetLogo-Implementierung einen deutlichen Unterschied zur Repast-Implementierung. Im NetLogo-Programm werden gefundene doppelte Artikel nicht gelöscht. Dies führt dazu, dass im späteren Verlauf bereits sanktionierte Artikel für neu ausgesprochene Sanktionen sorgen. Allerdings schwächt sich die Kurve trotzdem

leicht ab, da der Lernalgorithmus hier für ein verstärktes Lernen sorgt und dadurch wie in der Repast-Implementierung keine neuen Artikel mehr doppelt in die Wikipedia eingestellt werden.

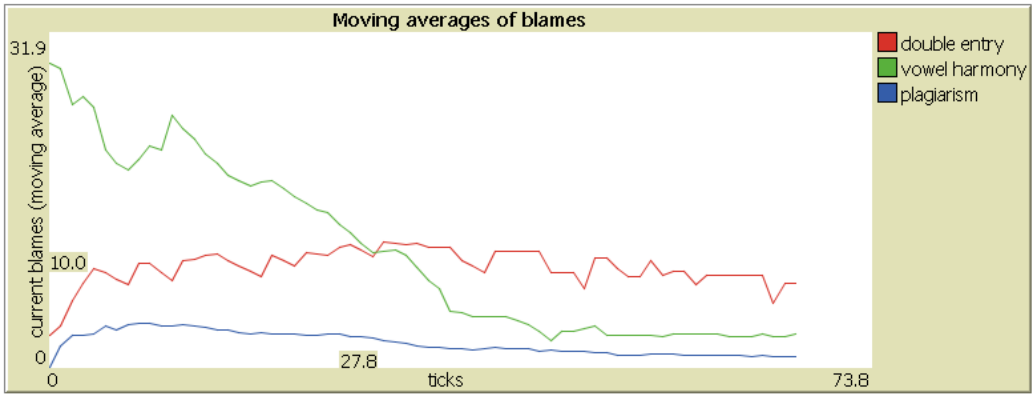

Abbildung 4.5: NetLogo Gleitender Durchschnitt

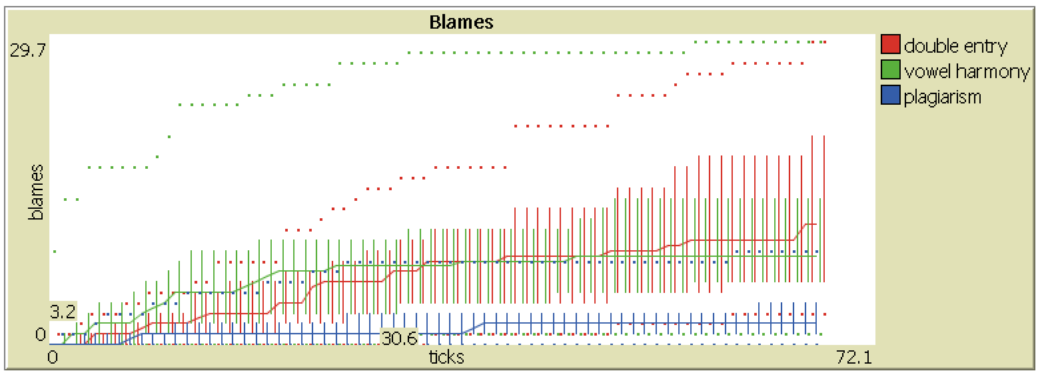

Abbildung 4.6: NetLogo Sanktionen

Die während der Simulationsläufe ermittelten Daten zeigen, dass das angestrebte Ziel, eine Simulationsumgebung als Kombination von EmIL-S und Repast zu erstellen, erfolgreich war. Auch werden nachvollziehbare Ergebnisse erzielt. Die Interpretation dieser Ergebnisse bedarf jedoch eines profunden Wissens über das Modell und die Implementierung. Auch gibt es noch einige Parameter und Eigenschaften an EmIL-S, die nachgebessert werden könnten. Speziell fehlt hier noch die Möglichkeit, events oder actions explizit von Lernprozessen auszuschließen. Eine Reihe von Fragen bleiben an dieser Stelle noch unbeantwortet. So wäre es sehr interessant zu sehen, wie die Agenten in weiteren Si-

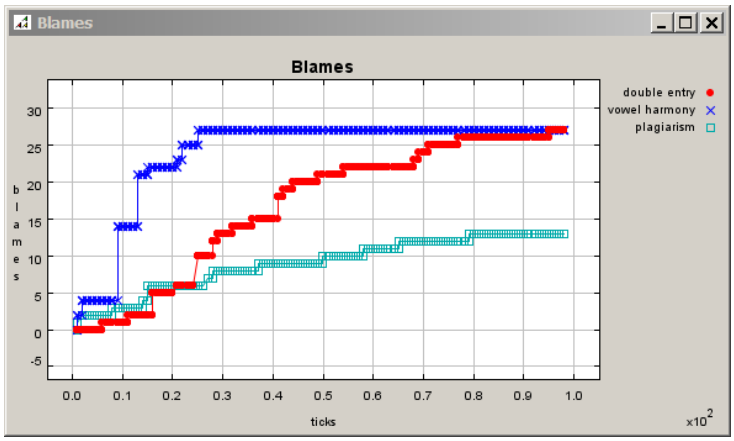

Abbildung 4.7: Repast Sanktionen

mulationsläufen mit unterschiedlichen Lernalgorithmen agieren. Auch der Vergleich der hierbei entstehenden Kurven und Daten verspricht eine Vielzahl von neuen Erkenntnissen.

Kapitel 5

Ausblick

Die Erforschung von Normentstehung mittels Simulationen wird in Zukunft weiter vorangetrieben werden. Im Verlauf des Projekts EmIL wird die Entwicklung der Simulationsumgebung zu einer lauffähigen und erweiterbaren Version bis zur Einstellung des Projektes weiter geführt werden. Im Zuge dieser Entwicklung werden Lernalgorithmen ausgearbeitet, die dann in EmIL integriert werden. Hiermit wird es möglich, unterschiedliche Simulationsmodelle zu implementieren und innerhalb der Simulationsumgebung zur Ausführung zu bringen. Im Hinblick auf die Vielzahl unterschiedlichster Modelle und Szenarien werden sich weitere Erweiterungsmöglichkeiten für EmIL-S ergeben. So sind bereits einige Erweiterungen in der Diskussion, die es ermöglichen sollen, parametrisierbare Nachrichten an EmIL-S zu senden. Auch sind Erweiterungen geplant, die eine Art Skriptsprache innerhalb der Nachrichten vorsehen. Mit dieser wäre es dann möglich, Konfigurationen zur Laufzeit der Simulation zu verändern. Die Entwicklung und Implementation diverser Lernalgorithmen wird sicher anderen Erweiterungen vorangestellt werden. Eine flexible und einfache Einbindung von Lernalgorithmen könnte die Entwicklung auch von projektfremden Entwicklern fördern und so zu einer größeren Zahl von Lernalgorithmen führen. Durch die Vielfalt von Lernalgorithmen, die mit unterschiedlichen Szenarien und Modellen in Simulationen Verwendung finden können, ergibt sich die Notwendigkeit, die entstehenden Daten schnell und präzise auswerten zu können. Daher wird die Anbindung von MEME ein weiteres vorrangiges Ziel sein. Hier ist sowohl Entwicklung beim MEME-Team als auch von Seiten der EmIL-Projektgruppe der Universität-Koblenz-Landau notwendig. Da bislang erst wenige Szenarien mit Hilfe der Simulationsumgebung zur Ausführung gebracht wurden, ist es an dieser Stelle schwer abzuschätzen, welchen Platz dieser völlig neue Ansatz in der Simulationsforschung einnehmen wird. Hierbei

werden sowohl die Benutzerfreundlichkeit und der leichte Umgang mit der Simulationsumgebung als auch das Potenzial für neue Erkenntnisse eine entscheidende Rolle spielen. Die Abbildung menschlichen Verhaltens ist ein Kernelement der Entwicklung. Dadurch ergeben sich sehr viele Einsatzgebiete. So wird es in Zukunft durch diese Software möglich, nicht nur Normentstehung zu erforschen, sondern auch andere Bereiche, in denen komplexes menschliches Verhalten abgebildet werden soll, zu ergründen. Die hier vorliegende Arbeit hat gezeigt, dass es möglich ist, einen lauffähigen Prototypen zu erzeugen. Dieser Prototyp kann als Grundlage für weitere Entwicklung genutzt werden. Auch können sich andere Entwickler an ihm orientieren, um eigene Projekte zu realisieren. Das verwendete Wikipediaszenario bietet noch viel Raum für Erweiterungen. So ist der in dieser Arbeit beschriebene Ansatz nur einer von vielen Möglichkeiten, um die Simulation der Realität anzunähern. Es sind auch Erweiterungen oder Abänderungen denkbar, die nur bestimmte Aspekte der Normentstehung berühren und nichts mit der realen Welt gemeinsam haben.

Kapitel 6

Fazit

Die vorliegende Arbeit hat gezeigt, dass die Simulationsumgebung in der bestehenden Konzeption und in der bestehenden Fassung funktionstüchtig ist. So ließ sich die Entwicklung und Implementation eines Szenarios realisieren und innerhalb der Simulationsumgebung zur Ausführung bringen. Durch diese Arbeit wurden einige Fehler und Unzulänglichkeiten innerhalb der Komponente EmIL-S aufgezeigt, wodurch sich diese erst erkennen und beheben ließen. Hierdurch half diese Arbeit dabei, EmIL-S signifikant zu verbessern. Schwerwiegende konzeptionelle Unzulänglichkeiten wurden jedoch nicht aufgedeckt. Dies war aufgrund der umfangreichen Planung auch nicht zu erwarten. Um in Zukunft Wissenschaftler bei der Nutzung von EmIL-S zu unterstützen, sollten jedoch noch Leitfäden zu diesem Thema erstellt werden. Die Erstellung einer Simulation sollte in Zukunft durch eine umfangreiche und leicht verständliche Programmspezifikation und durch eine oder mehrere Anleitungen begleitet werden. Diese Arbeit kann, zusammen mit dem gleichzeitig entwickelten Prototypen, eine erste Anleitung für zukünftige Entwickler darstellen.

Die bisherige Entwicklung der Projektgruppe der Universität-Koblenz und auch des gesamten Projektes EmIL zeigt ein zukunftsfähiges Konzept, welches interessante und weitreichende Möglichkeiten zur Erforschung von Normentstehung bieten. Es bleibt abzuwarten, ob das Projekt EmIL das Potential hat, einen eigenen neuen Simulationsstandard innerhalb der Simulationsfamilie zu etablieren. Das Konzept, die kognitiven Komponenten innerhalb einer eigenen getrennt von anderen Teilen ablaufenden Software zu bündeln, fand bei einer großen Anzahl unterschiedlicher Fachrichtungen großen Anklang. Sollte sich hier eine größere Interessentengruppe bilden, so kann dem Konzept und den Ideen, die hinter EmIL stehen, noch eine große Zukunft bevorstehen.

Literaturverzeichnis

[ACCC08] ANDRIGHETTO GIULIA, CONTE ROSARIA, CAMPENNÌ MARCO und CECCONI FEDERICO: *Normal = Normative? The Role of Intelligent Agents in Norm Innovation*. In: *The Fifth Conference of the European Social Simulation Association*, Brescia, 1. bis 5. September 2008. University of Brescia.

[ACTP07] ANDRIGHETTO GIULIA, CONTE ROSARIA, TURRINI PAOLO und PAOLUCCI MARIO: *Emergence In the Loop: Simulating the two way dynamics of norm innovation*. In: BOELLA GUIDO, VAN DER TORRE LEON und VERHAGEN HARKO (Herausgeber): *Normative Multi-agent Systems*, Nummer 07122 in *Dagstuhl Seminar Proceedings*, Dagstuhl, Germany, 2007. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[AIT09] AITIA INTERNATIONAL INC.: *MEME*. Internetquelle, `http://mass.aitia.ai/intro/meme`, April 2009. Abgerufen am 25.02.2009.

[DAvG08] DEFERRARI LUNA, AITKEN STUART, VAN HEMERT JANO und GORYANIN IGOR: *WikiSim: simulating knowledge collection and curation in structured wikis*. In: *WikiSym '08: Proceedings of the 2008 international symposium on Wikis*, New York, NY, USA, 2008. ACM.

[EE04] ECKSTEIN RAINER und ECKSTEIN SILKE: *XML und Datenmodellierung*. Dpunkt-Verl., Heidelberg, 2004.

[EmI07] EMIL PROJECT TEAM: *EMIL D3.1: Requirement analysis: Requirements that EmIL-S must meet*, August 2007.

[EmI08a] EMIL PROJECT TEAM: *EMIL D3.2: Report of simulator and agent design*, August 2008.

[EmI08b] EMIL PROJECT TEAM: *EMIL D3.3: EmIL-S: The simulation platform*, August 2008.

[EmI09] EMIL PROJECT TEAM: *EmIL Project*. Internetquelle, `http://emil.istc.cnr.it/`, Januar 2009. Abgerufen am 20.01.2009.

[GC95] GILBERT NIGEL und CONTE ROSARIA: *Artificial societies*. UCL Press, London, 1995.

[Gol96] GOLDSMITH JOHN A.: *The handbook of phonological theory*. Blackwell, Oxford, 1. Auflage, 1996.

[Hor01] HORTY JOHN F.: *Agency and Deontic Logic*. Oxford University Press, 2001.

[JS93] JONES ANDREW J. I. und SERGOT MAREK J.: *Deontic Logic in Computer Science - Normative System Specification*. John Wiley and Sons, 1993.

[Krü01] KRÜGER GUIDO: *GoTo Java 2*. Addison-Wesley, München, 2. Auflage, 2001.

[LM09a] LOTZMANN ULF und MÖHRING MICHAEL: *PowerPoint-Präsentation, EmIL-Meeting Koblenz*. Technischer Bericht, Universität Koblenz-Landau, 2009.

[LM09b] LOTZMANN ULF und MÖHRING MICHAEL: *Simulating Normative Behaviour and Norm Formation Processes*. In: OTAMENDI JAVIER, BARGIELA ANDRZEJ, MONTES JOSE LUIS und PEDRERA LUIS MIGUEL DONCEL (Herausgeber): *23nd European Conference on Modelling and Simulation*, Seiten 187–193, Madrid, 9. bis 12. Juli 2009.

[LM09c] LOTZMANN ULF und MÖHRING MICHAEL: *Simulating Norm Formation - An Operational Approach*. In: *Proc. of the 8th Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS 2009)*, Seiten 1323–1324. Budapest, Mai 2009.

[LMT08] LOTZMANN ULF, MÖHRING MICHAEL und TROITZSCH KLAUS G.: *Simulating Norm Formation in a Traffic Scenario.* In: SQUAZZONI FLAMINIO (Herausgeber): *The Fifth Conference of the European Social Simulation Association*, Brescia, 1. bis 5. September 2008.

[May00] MAYR ERNST: *Das ist Biologie: Die Wissenschaft des Lebens*. Spektrum Verlag, Heidelberg - Berlin, 2000.

[OW02] O'CONNOR TIMOTHY und WONG YU HONG: *Emergent Properties*. Internetquelle, `http://plato.stanford.edu/entries/properties-emergent/`, September 2002. Abgerufen am 20.01.2009.

[Rep08] REPAST DEVELOPMENT TEAM: *How to Build a Repast Model-1*. Internetquelle, `http://repast.sourceforge.net/repast_3/how-to/simplemodel.html`, Juli 2008. Abgerufen am 17.12.2008.

[Rep09] REPAST DEVELOPMENT TEAM: *Repast Recursive Porus Agent Simulation Toolkit*. Internetquelle, `http://repast.sourceforge.net/repast_3/index.html`, Januar 2009. Abgerufen am 20.01.2009.

[RHQ+05] RUPP CHRIS, HAHN JÜRGEN, QUEINS STEFAN, JECKLE MARIO und ZENGLER BARBERA: *UML 2 glasklar*. Hanser Fachbuchverlag, München, 2. Auflage, 2005.

[Rup04] RUPP CHRIS: *Requirements-Engineering und -Management*. Hanser Fachbuchverlag, München, 3. Auflage, 2004.

[Sar05] SARTOR GIOVANNI: *Legal Reasoning, A Cognitive Approach to the Law*. Springer, 2005.

[Sun09] SUN MICROSYSTEMS, INC. : *Javadoc Tool*. Internetquelle, `http://java.sun.com/j2se/javadoc/`, August 2009. Abgerufen am 13.03.2009.

[Swa07] SWARM DEVELOPMENT GROUP: *A resource for agent- and individual-based modelers and the home of Swarm*. Internetquelle, `http://www.swarm.org/index.php/Main_Page`, Mai 2007. Abgerufen am 24.04.2009.

[Tel00] TELECOM ITALIA LAB: *Java Agent Development Framework*. Internetquelle, `http://jade.tilab.com/`, Februar 2000. Abgerufen am 24.04.2009.

[Tro08] TROITZSCH KLAUS G.: *Wikipedia*. NetLogo-Programm, `http://www.uni-koblenz.de/~kgt/Pub/Wikipedia.nlogo`, März 2008. Abgerufen am 20.01.2009.

[Tro09a] TROITZSCH KLAUS G.: *EMIL*. Internetquelle, `http://www.uni-koblenz-landau.de/koblenz/fb4/institute/iwvi/agtroitzsch/projekte/emil-1/?set_language=en`, Januar 2009. Abgerufen am 20.01.2009.

[Tro09b] TROITZSCH KLAUS G.: *Perspectives and Challenges of Agent-Based Simulation as a Tool for Economics and Other Social Sciences*. In: DECKER, SICHMAN, SIERRA und CASTELFRANCHI (Herausgeber): *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Brescia, 10. bis 15. Mai 2009. Copyright 2008, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

[Wil99] WILENSKY URI: *NetLogo*. Internetquelle, `http://ccl.northwestern.edu/netlogo/`, 1999. Abgerufen am 27.01.2009.

[Wri05] WRIGHT GEORG HENRIK: *Norm and Action. A Logical Inquiry*. Routledge and Paul Kegan, London, 2005.