Funktionelle Erweiterung und Anpassung für das Mikrosimulationsprogramm CoMicS im Bereich der grafischen Ergebnisausgabe

Diplomarbeit

Zu Erlangung des Grades Diplom Informatiker (Dipl. Inf.)
im Studiengang Computervisualistik

vorgelegt von
Sascha Bonnemann

Erstgutachter: Prof. Dr. Klaus G. Troitzsch
Institut für Wirtschafts- und Verwaltungsinformatik
Universität Koblenz-Landau
Zweitgutachter: Dr. Marc Hannappel
Institut für Soziologie
Universität Koblenz-Landau

Koblenz, den 30.11.2016

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 dieser Arbeit in die Bibliothek bin ich einverstanden.	☐	☐

.....

(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

In den vergangen Jahren sind Rechnergeschwindigkeiten und -kapazitäten rapide gestiegen. Neue Chip- und Festplattentechnologien bewirken teilweise Verdopplungen dieser Bereiche in einem zweijährigen Rhythmus. Diese Entwicklung ist auch im Bereich der Mikrosimulation spürbar geworden. Aufgrund der stetig wachsenden technischen Möglichkeiten hat auf der anderen Seite auch die Komplexität von Mikrosimulationsmodellen zugenommen, da entsprechende Software immer schneller und effektiver arbeitet. Gerade in Zeiten von „Big Data“ und „Data Mining“ werden immer größere Datensätze in noch kürzerer Zeit simuliert und berechnet. Die zunehmende Komplexität und Größe von Simulationen ist durchaus als positiv zu bewerten, wirft aber gleichzeitig die Frage auf, wie die Nutzer von Mikrosimulations-Software auch ohne technische Expertisen zukünftig in der Lage sein werden, entsprechende Programme leicht nutzen zu können. Dieser Problemstellung hat sich Christian Tiefenau im Jahr 2014 im Rahmen seiner Master-Thesis angenommen, indem er eine modulare, client-/serverbasierte Version des Mikrosimulationsprogramms „CoMicS“ implementierte, welche es Nutzern unabhängig von vorhandenen oder fehlenden Programmierkenntnissen erlaubt, komplexe Simulationsmodelle umzusetzen und berechnen zu lassen. Das Programm ist durchgehend modular und dynamisch konzipiert, wodurch es weder für die Simulationsmodelle, noch für die Eingangsdatensätze hinderliche Restriktionen gibt.
Komplexere Simulationen erzeugen immer komplexere Ergebnisdaten, woraus sich eine weitere Fragestellung anschließt, wie im Sinne der oben angesprochenen Modularität, eine grafische Ergebnisauswertung in „CoMicS“ zu implementieren ist, die über Text- und Tabellenansichten hinausgeht.
Das Ziel dieser Diplomarbeit ist es daher, eine grafische Ergebnisausgabe in die bestehende Struktur der Software „CoMicS“ zu implementieren. „Grafische Ergebnisausgabe“ bedeutet hierbei: ein Nutzer dieser Mikrosimulationssoftware soll in die Lage versetzt werden, die errechneten Ergebnisdaten visuell explorieren zu können. „Co-

MicS“ ist dynamisch und modular programmiert, damit der Nutzer möglichst viele Freiheiten bei der Benutzung genießt. Dies bedeutet beispielsweise für die grafische Ergebnisausgabe, dass dieses Vorgehen adaptiert werden muss. Die Ausgabe soll so programmiert werden, dass sie alle Events (unabhängig davon welche und wie viele es sind) in Form von Diagrammen unter Betrachtung der zeitlichen Komponente anzeigen wird. Da „CoMicS“ eine Software ist, deren Nutzeroberfläche innerhalb eines Webbrowsers angezeigt wird, wird die Ergebnisausgabe ebenfalls in diesen Rahmen eingebettet. Einen Teil dieser Arbeit bildet folglich die Recherche nach einem passenden Tool/einer Javascript-Bibliothek, wodurch die Grundfunktionen zur Diagrammerstellung zur Verfügung gestellt und in das bestehende Softwarekonstrukt eingearbeitet werden können.
Die Arbeit gliedert sich in vier Hauptkapitel. Im ersten Kapitel „Mikrosimulation“ wird ein Überblick über Begrifflichkeiten und Prinzipien von Mikrosimulationen gegeben. Dieses grundlegende Verständnis ist Voraussetzung dafür, dass Ziel und Sinn der Software „CoMicS“ verstanden, interpretiert und für die eigene Erweiterung adaptiert werden können.
Darauf folgt das Kapitel „Programmanalyse“ in dem bereits vorhandene Software im Hinblick auf die grafische Ergebnisausgabe untersucht wird. Es wird analysiert, welche Möglichkeiten dem Nutzer zur Verfügung gestellt werden und ob/inwiefern diese sinnvoll sind. Der besondere Fokus liegt an dieser Stelle auf den aktuell umfangreichsten und bekanntesten Programmen „LIAM2“ und „Modgen“, welche stellvertretend für eine Vielzahl an existierenden Softwarelösungen betrachtet werden sollen.
Anschließend wird im Kapitel „Datenvisualisierung“ die Wichtigkeit von visuellen Datenrepräsentationen erläutert und in einem eigenen Unterkapitel der Sinn und die gegebene Relevanz im Bereich der Mikrosimulation verdeutlicht. Der „Versuch“ einer Definitionsfindung zu diesem Begriff sowie ein kurzer historischer Überblick zu diesem Forschungsgebiet sind ebenfalls Teil dieses Kapitels.
Das fünfte Kapitel „Grafische Ergebnisausgabe in CoMicS“ bildet den Kern dieser Diplomarbeit. Hier wird die Implementierung der grafischen Ergebnisausgabe vorgestellt und dokumentiert.
Das Ende dieser Diplomarbeit bilden ein Fazit, welches die Zielvorgabe mit dem Endergebnis vergleicht, und ein Ausblick, der zukünftige Erweiterungen und Anpassungen behandeln wird.

2 Mikrosimulation

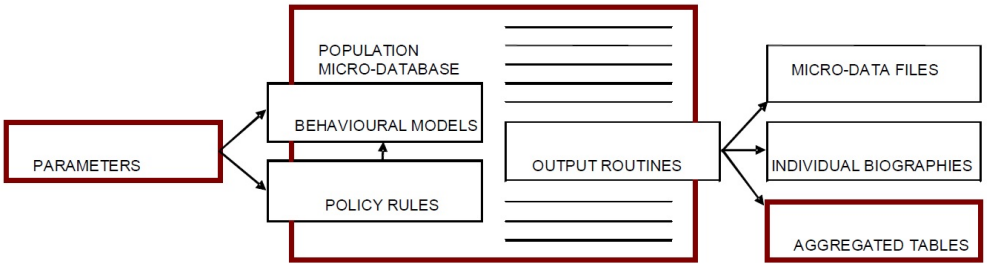
Mikrosimulationen werden heutzutage immer häufiger und intensiver als Werkzeug im Bereich der Forschung eingesetzt. Besonders gut geeignet, und deshalb dort besonders häufig vertreten, sind sie für sozio-ökonomische und politische Anwendungsbereiche. Generell dienen Mikrosimulationen dazu, dass basierend auf einem heterogenen Datensatz ein bestimmtes Verhalten für die im Datensatz enthaltenen Individuen für die Zukunft vorausgesagt werden kann. Im Bereich der Politik wäre daher ein möglicher Anwendungsfall zu prüfen, in welcher Höhe finanzielle Mehreinnahmen für den Staat nach einer Mehrwertsteuererhöhung zu erwarten wären. Um derartige und andere Fragestellungen zu simulieren bedarf es detaillierter Datensätze und komplexer Simulationsmodelle, welche die entsprechenden Daten in Abhängigkeit von verschiedenen Ausgangsparametern logisch repräsentieren. Diese sogenannten Verhaltensregeln können dabei wiederum selbst auf empirischen Auswertungen von Quer- oder Längsschnittdatensätzen basieren.
Seit der Einführung des Begriffs und der Technik *Mikrosimulation* (Orcutt, 1957) vor über einem halben Jahrhundert haben sich die Ansätze, Techniken und vorallem auch die verfügbaren Datensätze extrem gewandelt und bieten somit Wissenschaftlern und Forschern mehr denn je eine sehr gute Möglichkeit bestimmte gesellschaftliche Probleme und Fragestellungen mittels Mikrosimulationen zu analysieren und/oder vorherzusagen. [SPIELAUER, 2009]

2.1 Das Mikrosimulationsmodell

Damit eine Mikrosimulation durchgeführt werden kann, benötigt man ein ihr zugrundeliegendes Modell, welches die Simulations beschreibt - das sogenannte Mikrosimulationsmodell. In der folgenden Grafik kann man die einzelnen Elemente eines solchen Modells erkennen.

Das zentrale Element innerhalb eines solchen Modells ist ein repräsentativer Da-

Abbildung 2.1: Mikrosimulationsmodell nach Spielauer



Quelle: [SPIELAUER, 2009]

tensatz, auf dessen Grundlage die Simulation einmal laufen wird. Dieser enthält alle relevanten Charakteristika und Variablen (bspw. Geschlecht, Geburtsjahr, Bildungsstand, etc.) die notwendig sind, um einen einzelnen Datensatzeintrag (einzelne Person, Haushalt, etc.) innerhalb der Simulation über einen bestimmten Zeitraum beschreiben zu können. Der Datensatz stellt somit den Ausgangspunkt bzw. die Basis einer jeden Simulation dar, weshalb es besonders wichtig ist, dass dieser alle wichtigen Informationen in einer derartigen Form bereitstellt, dass das Mikrosimulationsprogramm diesen entsprechend verarbeiten kann. Verarbeiten bedeutet hierbei, dass vordefinierte Funktionen, Verhaltensmodelle und Regeln auf die Einträge im Datensatz angewandt werden und diese somit verarbeiten und verändern können. Am Ende eines Simulationslauf kann der veränderte Datensatz nun zu der Analyse einer Ausgangsfragestellung genutzt werden. Die gerade angesprochenen Funktionen werden in sogenannten Modulen organisiert. Diese Simulationsmodule beinhalten biographische Ereignisse (Geburt, schulische Laufbahn, Heirat, Familiengründung, etc.) und bestimmen deren Eintreten für einen bestimmten Eintrag des Datensatzes, sprich für ein in ihm enthaltenes Individuum. Die Module regulieren somit den Fortbeschreibungsprozess einer Simulation und sind immer ereignisspezifisch. Fortbeschreibungsprozess bedeutet hierbei: in einem Simulationsprozess wird der zugrunde liegende Datensatz in Abhängigkeit zuvor implementierter Verhaltensregeln bis zu einem bestimmten Zeitpunkt fortgeschrieben und entsprechende Merkmale des Datensatzes verändert. [SPIELAUER, 2009]

Das Kernstück jeden Moduls ist ein Simulationsalgorithmus. Diese Simulationsalgorithmen sind Programmcodes, die den Ablauf innerhalb eines Moduls beschreiben und steuern. An diesen Stellen wird entschieden, wie und nach welchen Regeln der Fortschreibungsprozess auszusehen hat. Konkret entschieden wird, welche Akteure in

ein Modul eingelesen werden und anhand welcher Funktionen der Fortschreibungsprozesse erfolgt. Um diese Entscheidung innerhalb eines Moduls treffen zu können, bedient man sich einer Kombination aus empirisch ermittelten Eintrittswahrscheinlichkeiten (Fortschreibungsparameter) und einem Zufallsexperiment (Monte-Carlo-Experiment[1]), dessen Basis das *Gesetz der großen Zahlen*[2] darstellt.
Der Simulationsalgorithmus operiert auf Basis von einem simulationsinternen Zufallsgenerator, welcher eine gleichverteilte Zufallszahl aus dem Intervall [0,1] zieht und diese mit der zuvor ermittelten bedingten Wahrscheinlichkeit vergleicht. Für den Fall, dass der gezogene Zufallswert kleiner oder gleich dem der bedingten Wahrscheinlichkeit für den Eintritt eines Ereignisses ist, findet das entsprechende Ereignis für eine Person statt, andernfalls wird es nicht angewandt. Ist das im Modul enthaltende Ereignis für eine Person eingetreten, werden in einem nächsten Schritt die Merkmale der betroffenen Person angepasst. Hierbei ist programmiertechnisch darauf zu achten, dass Änderungen, die während einer Simulation den Datensatz einer Person verändern, auch Auswirkungen auf andere Personen haben können. Dies ist zum Beispiel bei einem Ereignis der Art „Geburt eines Kindes“ der Fall. Ein großer Vorteil von Mikrosimulationen ist daher, dass sie eine Verbindung zwischen den Personen ermöglichen, welche durch Vektoren oder Zeiger realisiert werden können. [HANNAPPEL UND TROITZSCH, 2014]

2.2 Mikrosimulationsansätze

Mikrosimulationen basieren hauptsächlich auf stochastischen Berechnungsprozessen, dem zuvor erwähnten Monte-Carlo-Experiment, weshalb in den meisten Fällen häufig von einer Monte-Carlo-Simulation gesprochen wird. Trotzdem gibt es unterschiedliche Herangehensweisen bei den Mikrosimulationsmodellen, welche sich hauptsächlich durch drei Aspekte unterscheiden lassen: der Modellierung des individuellen Verhaltens, der Modellierung der Zeit (perioden- und ereignisorientierte Mikrosimulation) und den unterschiedlichen Arten der zugrundeliegenden Bevölkerung, also die Art der genutzten Datensätze innerhalb der Simulation.

[1] stochastisches Verfahren, bei dem eine sehr große Anzahl gleichartiger Zufallsexperimente die Basis darstellt

[2] das Gesetz der großen Zahlen besagt, dass sich die relative Häufigkeit eines Zufallsergebnisses immer weiter an die theoretische Wahrscheinlichkeit für dieses Ergebnis annähert, je häufiger das Zufallsexperiment durchgeführt wird

2.2.1 Verhaltensmodellierung

Da die komplexen Strukturen menschlichen Denkens, und im Speziellen der Entscheidungsfindungsprozess realer Menschen, noch nicht ohne weiteres mit Hilfe von Algorithmen abgebildet werden können, geschieht dies im Bereich der Mikrosimulation mit Hilfe von Wahrscheinlichkeiten, auch wenn es außer Frage steht, dass menschliche Handlungen und Entscheidungen keineswegs Zufallsergebnisse sind. Da es nicht möglich ist Charaktereigenschaften oder Neigungen eines realen Menschen empirisch zu erfassen, verwendet man in der Mikrosimulation das sich wiederholende Ziehen einer Zufallszahl. Als Annäherung hierzu bietet sich nun allerdings die empirisch messbare Häufigkeit an, mit der Menschen in bestimmten Situationen bestimmte Entscheidungen getroffen und somit bestimmte Ereignisse hervorgerufen haben - bspw. die Häufigkeit, mit der Frauen mit einem bestimmten Bildungsstand in einem bestimmten Alter ein Kind geboren haben, oder die Häufigkeit, mit der junge Menschen mit einem bestimmten Bildungshintergrund in einem bestimmten Alter den Übergang von einer Bildungsinstitution in eine andere vollzogen haben. Diese relativen Häufigkeiten werden dann als bedingte Wahrscheinlichkeiten verstanden, mit denen bei Vorliegen bestimmter Bedingungen bestimmte Entscheidungen getroffen und umgesetzt werden. [HANNAPPEL UND TROITZSCH, 2014]

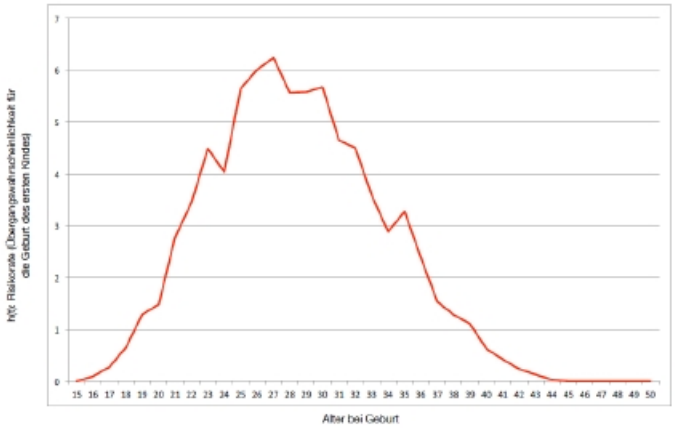
2.2.2 Zeitverlauf

Periodenorientierte Mikrosimulation

Innerhalb periodenorientierter Mikrosimulation wird die Zeit als Distanz zwischen vordefinierten Zeitintervallen $(t_0, t_1, ..., t_T)$ operationalisiert, welche meist ein Jahr oder einen Monat umfassen. Innerhalb eines jeden Zeitintervalls durchlaufen die im Datensatz existierenden Individuen verschiedene Module, in denen individuelle Merkmale fortgeschrieben werden. Die modulspezifischen Simulationsalgorithmen entscheiden darüber, welche Personen das Modul durchlaufen und welche Ereignisse jeweils angestoßen und berechnet werden. Zusätzlich sind an dieser Stelle die entsprechenden Verhaltensregeln implementiert, welche die Eintrittswahrscheinlichkeit für ein konkretes Ereignis festlegen. Neben Ereignissen, die alle Personen betreffen, gibt es solche, die nur von einer bestimmten Subpopulation durchlaufen werden können, wie bspw. „Geburt eines Kindes“ - dieses Ereignis wird logischerweise nur von weiblichen Ak-

teuren innerhalb der Simulation durchlaufen. Konkret bedeutet dies, dass zwar initial auch die männlichen Akteure in ein solches Modul aufgenommen werden, aber direkt nach Feststellung des Geschlechts wieder ausgeschlossen werden, da dieses Ereignis für sie irrelevant ist.
Man benötigt Übergangswahrscheinlichkeiten[3] (auch Risikoraten oder Hazard-Rates genannt) für jeden Akteur um den Eintritt eines Ereignisses berechnen zu können. Diese Übergangswahrscheinlichkeiten definieren für eine konkrete Merkmalsgruppe (abhängig von bspw. Alter, Geschlecht, Bildungsstand, etc.) die bedingte Wahrscheinlichkeit, dass ein Ereignis innerhalb von einem definierten Zeitintervall stattfinden wird oder nicht. [HANNAPPEL UND TROITZSCH, 2014] Abbildung 2.2 zeigt die

Abbildung 2.2: Übergangswahrscheinlichkeit für die Geburt des ersten Kindes
*Übergangswahrscheinlichkeiten (Hazard-Rates) zum Zeitpunkt t



Alter	h(t)*
0	0
(...)	(...)
28	5,56
29	5,57
30	5,67
31	4,64
32	4,49
33	3,58
34	2,88
35	3,27
36	2,38
(...)	(...)
50	0.0

Quelle: [HANNAPPEL UND TROITZSCH, 2014]

fiktiven Übergangswahrscheinlichkeiten für die Geburt eines ersten Kindes. Für bisher noch kinderlose Frauen wird im Simulationsmodul „Geburt“ berechnet, ob sie innerhalb des nächsten Simulationsintervalls ein Kind bekommen oder nicht. Wie bereits zuvor an anderer Stelle erwähnt, vergleicht der Simulationsalgorithmus innerhalb des Moduls den Wert der Übergangswahrscheinlichkeit für eine bestimmte Person mit der vom Zufallsgenerator ermittelten Zufallszahl. Dieser Vergleich bestimmt, ob für eine Person ein Ereignis eintritt oder nicht. Ist es der Fall, dass ein Ereignis für eine eingelesene Person stattgefunden hat, wird die entsprechende Merkmalsausprägung im

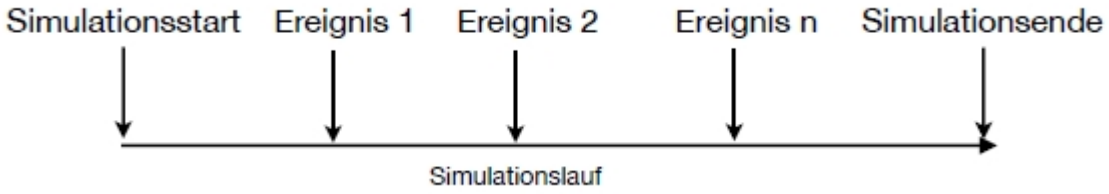
[3] allgemein versteht man hierunter die Wahrscheinlichkeit, mit der ein System pro Zeiteinheit von einem Zustand Z_t in einen Zustand Z_{t+1} wechselt

Datensatz angepasst und die Person verlässt das Modul.

Ereignisorientierte Mikrosimulation

„In ereignisorientierten Mikrosimulationsmodellen sind die Ereignisse die Fortschreibungsparameter. Die Zeit wird dabei als Zufallsvariable zwischen zwei Ereignissen modelliert, so dass die Simulation nicht wie oben beschrieben von Zeit t_s zu Zeit t_{s+1} fortschreitet, sondern von Ereignis e_i zu Ereignis e_{ti+1}“. [HANNAPPEL UND TROITZSCH, 2014]

Abbildung 2.3: Ablauf einer ereignisorientierten Mikrosimulation



Quelle: [HANNAPPEL UND TROITZSCH, 2014]

Anhand dieser Abbildung ist zu erkennen, wie ereignisorientierte Mikrosimulationen im Vergleich zu den vorher beschriebenen periodenorientierten Mikrosimulationen aufgebaut sind. Da sich der Fokus nun auf die Ereignisse selbst verlagert hat, werden zwei grundsätzliche Veränderungen zu dem vorherigen Modell der Mikrosimulation deutlich. Anstelle der in periodenorientierten Simulationsmodellen genutzten Übergangswahrscheinlichkeiten werden in ereignisorientierten Mikrosimulationsmodellen nun Überlebenswahrscheinlichkeiten (auch: Survivalrates) benutzt. Diese berechnen die Wahrscheinlichkeit des Auftreten eines Ereignisses bis zu einem bestimmten Zeitpunkt t_s.
Der zweite elementare Unterschied zur Struktur einer periodenorientierten Mikrosimulation ist, dass ereignisorientierte Modelle eine zusätzliche Datenbank benötigen. Diese Datenbank, oftmals auch als Ereigniskalender bezeichnet, beinhaltet alle Ereignisse für eine Person in chronologischer Reihenfolge. Anstelle einer Datenbank könnte man prinzipiell aber auch mit einer einfachen Liste oder binären Bäumen arbeiten, in denen die entsprechende Struktur samt Abhängikeiten hinterlegt ist. Es können zwar bereits einige Ereignisse vor Simulationsstart berechnet werden, allerdings gibt

es bestimmte Ereignisse, denen ein konkretes anderes Ereignis vorausgegangen sein muss, damit es überhaupt eintreten kann. Beispiele für diese Abhängigkeiten sind Ereignisabfolgen der Art „Geburt des ersten Kindes → Geburt des zweiten Kindes“ oder auch „Heirat → Scheidung“. [HANNAPPEL UND TROITZSCH, 2014]

2.2.3 Population

Die bisher genannten Modelle haben sich im Besonderen darin unterschieden, wie sie die Zeit innerhalb der Simulation verstehen und repräsentieren. Ein anderer Ansatz besteht darin, die Betrachtungsweise in erster Linie auf die Population zu richten. Es lassen sich Kohorten- von Populations-, Querschnitts- von synthetischen und offene von geschlossenen Modellen unterscheiden. [HANNAPPEL UND TROITZSCH, 2014]

Populations- und Kohortenmodelle

Bei Populationsmodellen wird entweder der komplette Datensatz, welcher alle Personen umfasst, oder eine Stichprobe aus selbigem entnommen. Der Fokus der Fortbeschreibung liegt bei diesen Modellen auf ausgewählten Parametern der Akteure. Im Gegensatz hierzu arbeiten Kohortenmodelle mit einer konkreten Teilmenge des Datensatzes und konzentrieren sich beispielsweise nur auf Personen aus einem bestimmten Geburtsjahrgang. Diese Konkretisierung ermöglicht zum Beispiel eine Analyse der Auswirkungen fiskalpolitischer[4] Entscheidungen auf die lebenslaufspezifische Einkommensentwicklung.

Querschnitts- und synthetische Modelle

Die Basis für Querschnittsmodelle bildet ein repräsentativer Datensatz, welcher die Merkmale und Eigenschaften einer bestimmten Bevölkerung zu einem bestimmten Zeitpunkt darstellt. Ein solcher Datensatz ist das Ergebnis einer Bevölkerungsumfrage und beinhaltet folglich Angaben über reale Akteure und ihre Merkmale. Während einer Simulation werden alle vorhandenen Akteure eines Datensatzes eingelesen und in Abhängigkeit verschiedener Merkmale oder Merkmalskombinationen fortgeschrieben.

[4] die Fiskalpolitik ist ein wirtschaftliches Instrument des Staates, welches mittels der Beeinflussung von Steuern und Stattsausgaben die konjunkturellen Schwankungen auszugleichen versucht

Bei synthetischen Modellen basieren die individuellen Daten nicht auf realen Befragungen, sondern werden künstlich durch sogenannte Imputationsverfahren[5] erzeugt und den Akteuren zugeordnet.

Offene und geschlossene Modelle

Die Differenzierung zwischen offenen und geschlossenen Modellen bezieht sich darauf, inwiefern ein vorhandener Datensatz an Personen erweitert oder verkleinert werden kann. Bei offenen Modellen ist eine Beeinflussung des Datensatzes von Äußen" beispielsweise durch Ein- und Auswanderungen denkbar. Zusammen mit Fertilität und Mortalität bilden die Migrationsprozesse alle möglichen demographischen Entwicklungsmöglichkeiten ab. Bei geschlossenen Modellen ist die einzige Möglichkeit der Erweiterung eines Datensatzes die Geburt einer neuen Person. Ansonsten gibt es keine weitere Möglichkeit, dass ein neuer Fall dort aufgenommen wird. Die Entscheidung für ein offenes oder geschlossenes Modell ist daher meist vom Umfang der zur Verfügung stehenden Informationen abhängig.

2.2.4 Ergebnisse

Ergebnis eines Mikrosimulationsmodells ist ein errechneter Datensatz, welcher in seiner Struktur der des zu Beginn genutzten Ausgangsdatensatzes entspricht. Allerdings enthält dieser Datensatz auch alle Abbilder der neu hinzugekommenen Akteure mit ihren unterschiedlichen Merkmalen, wodurch letztendlich ein fiktiver Datensatz entstanden ist, dessen Basis vordefinierte Regeln und Verteilungswahrscheinlichkeiten bilden. In anschließenden Analysen sind diese Ergebnisse daher stets kritisch zu beobachten, da sie allesamt auf Monte-Carlo-Experimenten basieren. Jedes Ergebnis ist daher ein individuelles Resultat, welches von denen aus anderen Durchläufen losgelöst ist, solange nicht eine hinreichend große Anzahl an Durchläufen vollzogen wurde, um dadurch Auskunft über die Gesamtvarianz des Modells zu erhalten. [HANNAPPEL UND TROITZSCH, 2014]

[5] Imputation gehört zu den sogenannten Missing-Data-Techniken - dies sind Verfahren, die fehlende Daten in statistischen Erhebungen in den Datensätzen vervollständigen

3 Programmanalyse

Bevor konkrete Änderungen oder Erweiterungen der Software „CoMicS“ angedacht oder umgesetzt werden können, ist es im wissenschaftlichen Sinne wichtig zu erfassen, was andere Programme aktuell bereits leisten können beziehungsweise in welchen Bereichen Verbesserungsmöglichkeiten oder gar Innovationen möglich sind. Dieses Kapitel befasst sich daher mit dem Status Quo der aktuell zur Verfügung stehenden Programme im Bereich der dynamischen Mikrosimulation. Eine zu diesem Themenbereich sehr umfangreich verfasste Zusammenstellung datiert aus dem Jahre 2013: „A Survey Of Dynamic Microsimulation Models: Uses, Model Structure And Methodology“ von Jinjing Li und Cathal O‘Donoghue beinhaltet eine zum damaligen Zeitpunkt breitgefächerte Dokumentation und Analyse der international relevanten Mikrosimulationsprogramme. [LI UND O‘DONOGHUE, 2013] Die Gesamtheit der dort aufgelisteten Modelle und Programme zu testen wäre zeitlich im Rahmen einer Diplomarbeit nicht realisierbar. Deshalb werden stellvertretend zwei Mikrosimulationsprogramme ausgewählt und einer weiteren Betrachtung unterzogen. Neben grundlegenden Analysekriterien, technischen Voraussetzungen und Usability sollen die beiden Programme hinsichtlich der Tools und Möglichkeiten zur visuellen Ausgabe/Darstellung der Simulationsergebnisse genauer analysiert werden werden.

3.1 LIAM2

Das erste Programm, welches in diesem Zusammenhang untersucht werden soll, ist LIAM2. LIAM2 ist eine Software, mit der verschiedene Arten von Mikrosimulationsmodellen entwickelt werden können. Bei der Entwicklung dieser Software wurde bewusst darauf geachtet, dass die Toolbox so generisch wie möglich erstellt wurde, damit alle möglichen Varianten von Modellen mittels LIAM2 umgesetzt und getestet werden können. Das Programm wird unter der „GNU General Publice License (GPL) version 3“ [GNU General Public License Version 3, 2016] zur Verfügung ge-

stellt, was zusammengefasst bedeutet, dass man LIAM2 kopieren, modifizieren und weitervertreiben darf, wenn man sich an ein paar generelle Grundregeln hält. Mittels dieser Vorgehensweise erhoffen sich die Entwickler, dass die Entwicklungskosten für Mikrosimulationsmodelle hinsichtlich Zeit und Geld reduziert werden können [LIAM2 HOMEPAGE, 2016]. Die zum Zeitpunkt der Erstellung dieser Diplomarbeit aktuellste Version ist 0.11, laut Homepage datiert auf den 24. November 2015 [LIAM2 0.11, 2015].

3.1.1 Allgemeines

Zugang

LIAM2 ist im Internet als kostenloser Download verfügbar und für jeden ohne Registrierung oder Ähnliches zugänglich. Die Webseite stellt zwei unterschiedliche Bundles der Programmversion 0.11 zur Verfügung. Beide sind für das Windows-Betriebssystem geschrieben, zum Einen für die 32-bit- und zum Anderen für die 64-bit-Version. Beide Varianten sind über einen direkten Download-Link erreichbar. Die Pakete enthalten jeweils beide die folgenden Bestandteile innerhalb einer .zip-Datei:

- eine ausführbare .exe-Datei für Windows
- den Texteditor „Notepad++“ [NOTEPAD++, 2016] in der Version 5.8.6, der bereits für das Arbeiten mit LIAM2-Modellen vorkonfiguriert wurde
- eine Dokumentation im .chm-Format[1] (.pdf- und .html-Versionen der Dokumentation sind als separate Downloads auf der Homepage verfügbar [LIAM2 DOKUMENTATION, 2016])
- ein Demo-Modell und ein synthetischer[2] Datensatz

[1] .chm-Dateien sind komprimierte Archive, die wiederum verschiedene Dateien wie Hilfetexte im HTML-Format, Bilder oder auch Javascript in kompilierter bzw. gepackter Form enthalten. Es ist ein Dateiformat der Firma Microsoft und wurde mit dem Betriebssystem Windows 98 eingeführt.

[2] synthetisch bedeutet an dieser Stelle, dass keine Daten verwendet wurden, die auf realen Personen basieren.

Desweiteren ist der offene Quellcode für LIAM2 ebenfalls auf der Homepage als .zip-Archiv zum Download vorhanden oder alternativ als GitHub-Projekt verfügbar [LIAM2 GITHUB, 2016].

Technische Voraussetzungen

LIAM2 wird aktuell nur für Windows-Betriebssysteme angeboten. Eine Version für Mac-User ist zum Stand dieser Diplomarbeit nicht vorhanden. Für Windows gibt es eine Variante für das 32-bit-System, und eine für das 64-bit-System. Der Unterschied dieser beiden Versionen besteht ausschließlich darin, dass die 64-bit-Version den Nutzer in die Lage versetzt, größere Datensätze für eine Simulation zu verwenden, da hierbei mehr als 2 Gb an Arbeitsspeicher genutzt werden können, was bei der 32-bit-Version nicht der Fall ist. Abgesehen davon unterscheiden sich die bereitgestellten Bundles nicht. Die Dateigrößen der .zip-Archive liegen bei 29.65 Mb (32-bit) und 51.33 Mb (64-bit) - somit sollten sie im Normalfall bei den heutigen Festplattengrößen keinerlei Probleme verursachen. Da das entpackte .zip-Archiv bereits alle notwendigen Dateien, Verzeichnisse und den Notepad++-Editor enthält, ist die Installationsroutine nach dem Entpacken unmittelbar abgeschlossen und es sind keine weiteren technischen Notwendigkeiten oder Limitierungen erkennbar. Generell ist natürlich die Größe des Arbeitsspeichers und die allgemeine Leistung des Rechners dafür verantwortlich, wie schnell eine Simulation berechnet werden kann, da LIAM2 die Simulationsberechnungen lokal auf dem Rechner durchführt.

Notwendige Programmierkenntnisse

Wie bereits erwähnt, liefert LIAM2 in beiden Bundles direkt den „Notepad++“-Editor mit. Dieser wird benötigt, um die Simulationen/Modelle definieren und bearbeiten zu können. Die zugrundeliegende Programmiersprache hierfür ist YAML[3], weshalb die entsprechenden Demo-Modelle und die eigenen in .yml-Dateien gespeichert werden. Aus dem Editor heraus können die dort definierten Modelle mit einem Druck auf die Taste „F6“ direkt gestartet werden. Der Editor ist so vordefiniert, dass

[3] YAML ist eine vereinfachte Auszeichnungssprache zur Datenserialisierung, angelehnt an XML und an die Datenstrukturen von Programmiersprachen wie Perl oder Python.

dieser Tastendruck bewirkt, dass die Datei „main.exe“[4] automatisch mittels Konsole aufgerufen wird und alle notwendigen Funktionen und Unterprogramme startet um die Simulation zu berechnen. Ist ein solcher Simulationslauf erfolgreich abgeschlossen worden, wird dem Benutzer des Programms eine interaktive Python-Konsole zur Verfügung gestellt. Mittels dieser Konsole ist es möglich, den zuvor erstellten Ergebnisdatensatz mit Hilfe von verschiedenen Python-Befehlen zu analysieren und ausgeben zu lassen. Hierfür werden bestimmte Methoden und simulationsspezifische Schlüsselwörter zur Verfügung gestellt.

Flexibilität

Die Software „LIAM2“ bietet dem Benutzer einen einfachen und guten Ansatz um mit eigenen Ausgangsdatensätzen arbeiten zu können, sofern diese gewissen strukturellen Grundregeln folgen. Die entsprechenden Datensätze werde im.csv-Dateiformat hinterlegt, benötigen in der ersten Zeile die Spaltennamen und darauf sukzessive folgend die jeweiligen Werte der Agenten, welche mittels Semikolon oder Komma getrennt werden können. Damit eigene Daten tatsächlich genutzt werden können, ist es notwendig, dass der Datensatz die Felder „id“ und „period“ enthält. Dies sind Schlüsselfelder für die interne Simulationsberechnung und Zuordnung von LIAM2 - alle weiteren Felder können vom Nutzer beliebig gewählt und benannt werden.

3.1.2 Modellierungsansatz

Zeitverlauf

LIAM2 ist eine Mikrosimulationssoftware, die nur periodenorientierte Simulationen erstellen und verarbeiten kann. Wie bereits im Kapitel 2.2.2 Zeitverlauf erläutert, wird hierbei die Zeit innerhalb der Simulation in der Art verstanden, dass sie die Distanz zwischen vordefinierten Zeitintervallen beschreibt, welche in der Regel einen Monat oder ein Jahr umfassen. Im Umkehrschluss bedeutet dies eine generelle Limi-

[4] nach dem Entpacken des Installations-zip-Archivs zu finden unter ¡Installationspfad¿/liam2/

tierung für das Erstellen von Mikrosimulationen mit LIAM2, da beispielsweise keine ereignisorientierten Simulationen möglich sind.

Population

Hinsichtlich der Wahl zwischen Populations-/Kohorten oder Querschnittsmodellen, gibt LIAM2 dem Benutzer die freie Wahl. Eigene Datensätze können wie bereits vorher beschrieben für die Simulationen genutzt werden und somit liegt auch die Entscheidung zwischen offenen und geschlossenen Modellen alleinig beim Entwickler. Um eigene Daten verwenden zu können, wird eine .csv-Datei für den Import benötigt, die die in der Dokumentation genauer beschriebene Struktur aufweisen muss. [LIAM2 DATA-IMPORT, 2016]

Zufallsfunktionen

LIAM2 liefert verschiedene Möglichkeiten für die Berechnung von Zufallszahlen und somit für das zufällige Verhalten der Agenten innerhalb einer Simulation. Wie bereits an anderer Stelle erwähnt, werden diese generierten Zufallszahlen mit den Werten der Wahrscheinlichkeitstabellen verglichen um zu entscheiden, ob beispielsweise ein konkretes Ereignis eintritt oder nicht. Die folgende Tabelle zeigt eine kurze Übersicht dieser Funktionen:

Tabelle 3.1: Funktionen zur Berechnung von Zufallszahlen in LIAM2

Funktionsname	Rückgabewert
uniform	Zufallszahl im Intervall [0,1) mit stetiger Gleichverteilung
normal	Zufallszahl im Intervall [0,1) mit Normalverteilung
logistic	logistischer Regressionswert berechnet anhand von $f(x) = \frac{1}{1+e^x}$

Neben den oben aufgezeigten Möglichkeiten bietet LIAM2 noch die Nutzung einer

Gumbelverteilung an und die Generierung einer Zufallszahl zwischen zwei vom Nutzer festgelegten Grenzen. Ebenfalls bereits in LIAM2 integriert ist die folgende Methode: $choice([option\_1, ..., option\_n], [prob\_option\_1, ..., prob\_option\_n])$
Sie entscheidet für eine der vordefinierten Optionen mit der jeweiligen Wahrscheinlichkeit und liefert diese Option entsprechend zurück. Ein sehr einfaches Beispiel zur Veranschaulichung ist die Ermittlung des Geschlechts einer Person. Mit einer Wahrscheinlichkeit von 51% bekommt man ein männliches Kind, die Wahrscheinlichkeit für ein weibliches liegt dementsprechend bei 49%. Unter der Voraussetzung, dass an anderer Stelle im Programm männlich als „true“ und weiblich als „false“ definiert wurde, sieht das konkrete Beispiel der choice-Funktion in LIAM2 folgendermaßen aus:

Listing 3.1: Bespiel für die Methode „choice“

```
1 gender: choice([True, False], [0.51, 0.49])
```

[LIAM2 DOKUMENTATION, 2016]

Datenmanagement

In LIAM2 werden Simulationsmodelle in YAML definiert und beschrieben. Um dies zu erreichen, müssen die Entitäten samt ihrer entsprechenden Felder im Modell definiert werden. Ein Beispiel für ein triviales Modell in LIAM2 wird in der folgenden Abbildung dargestellt und wurde der LIAM2-Dokumentation ([LIAM2 DOKUMENTATION, 2016]) entnommen:

Listing 3.2: Beispielcode für eine Simulationsbeschreibung in LIAM2

```
1  entities:
2      person:
3          fields:
4              # period and id are implicit
5              - age: int
6
7          processes:
8              ageing:
9                  - age: age + 1
10
```

```
simulation:
    processes:
        - person: [ageing]
    input:
        file: input.h5
    output:
        file: output.h5
    start_period: 2015
    periods: 10
```

Dieser Beispielcode enthält eine komplett lauffähige Beschreibung einer Simulation in LIAM2. „person“ ist eine Entität, welcher ein Geschlecht und ein Alter zugewiesen werden. Unter „processes“ wurde ein Alterungsprozess („ageing“) definiert, welcher das aktuelle Alter einer eingelesenen Person um den Faktor 1 erhöht. Mittels „start_period“ und „periods“ wird die Laufzeit der Simulation genau festgelegt: sie beginnt in diesem Fall im Jahr 2015 und läuft zehn Jahre in die Zukunft, folgich bis einschließlich 2025. Zusätzlich werden die Dateien des Ausgangsdatensatzes und die des Ausgabedatensatz festgelegt. „simulation“ legt fest, welche Prozesse für welche Entitäten verwendet werden sollen. In diesem sehr einfachen Bespiel ist es nur der Prozess „aging“ für die Entität „Person“.
LIAM bietet Entwicklern weiterhin die Möglichkeit an, mit Agenten zu arbeiten. Um dies innerhalb einer Simulation zu realisieren, wird das Schlüsselwort „path“ innerhalb der Entität „person“ verwendet um eine entsprechende .csv-Datei einzulesen, welche die zugehörigen Werte pro Agent beinhaltet. Diese externe Datei wird durch die Simulationsengine automatisch in ein Agentenmodell transformiert Für die .csv-Datei gilt zu beachten, dass der Datensatz in der ersten Zeile die Namen der Eigenschaften auflistet und in den folgenden Zeilen die jeweiligen Werte pro Agent definiert. Andernfalls kann die Datei nicht korrekt verarbeitet bzw. eingelesen werden. Der korrekte Importierungs-Code kann daher beispielsweise folgendermaßen aussehen:

Listing 3.3: Import einer externen .csv-Datei in LIAM2

```
entities:
    person:
        path:persons.csv
        fields:
```

```
5
6               - age: int
7               - gender: bool
8 ...
```

Dieses Agentenmodell zieh sich die Werte der Agenten aus der angegebenen persons.csv-Datei. Durch die Spezifikationen in „fields“ wird allerdings eine Einschränkung mit eingebaut - aus dem Datensatz in der Datei sollen pro Agent jeweils nur die Werte für „age“ und „gender“ ausgelesen werden. Damit alle vorhandenen Werte ausgelesen werden können, müsste man entsprechend die Codezeilen 4 bis 7 aus dem Programmcode entfernen.
Da der Datensatz selbst keine Informationen zu Beziehungen zwischen verschiedenen Agenten beinhaltet, muss dies, falls gewünscht oder notwendig, in LIAM2 über sogenannte „links“ realisiert werden. Eine verallgemeinerte Form einer solchen Verbindung zeigt die nachfolgende Abbildung:

Listing 3.4: Beispielcode für Linkbeziehungen in LIAM2

```
1 name: {type: <type>, target: <entity>, field: <name of link
     field>}
```

Für Verlinkungen innerhalb einer oder zwischen verschiedenen Entitäten stellt das Programm grundlegend zwei Möglichkeiten zur Verfügung: die „many2one“- und die „one2many“-Beziehung.

3.1.3 Ergebnisausgabe und -darstellung

Auch wenn LIAM2 auf den ersten Blick wie ein sehr schlankes und reduziertes Programm erscheinen mag, bietet es gerade im Bereich der Ergebnisausgabe/-darstellung überraschend viele Möglichkeiten. Dies reicht von der einfachen tabellarischen Ansicht in der Python-Konsole bis hin zu verschiedenen Darstellungsmöglichkeiten der Ergebnisse in Diagrammform. Die Standardinstallation liefert mit „ViTables“ außerdem ein weiteres kleines Programm, mittels diesem man sich die Ausgangs- sowie die Ergebnisdatensätze tabellarisch anzeigen lassen und nach eigenen Bedürfnissen filtern

kann. Die simulierten Datensätz werden standardmäßig im .hdf5-Format[5] ausgeliefert - im Anschluß ist es möglich mit Drittanbieter-Software diese Ergebnisse auslesen und darstellen zu lassen. Sämtliche Wege sind in der LIAM2-Dokumentation ausführlich beschrieben und werden in diesem Abschnitt in jeweils separaten Themen behandelt und veranschaulicht.

Python-Konsole

Wie bereits erwähnt, funktioniert LIAM2 derart, dass ein entsprechendes Modell in der portablen „Notepad++“-Version gestartet und ausgeführt wird, indem man mit Betätigen der Taste „F6“ das vordefinierte Makro „LIAM2: run model (with console)“ auslöst. Hierdurch wird die Simulation gestartet und komplett durchlaufen. Das Drücken von „F6“ bewirkt außerdem, dass im unteren Teil von „Notepad++“ die Python-Konsole geöffnet wird und anzeigt, ob und wie die Simulation erfolgreich durchgelaufen ist. Wie anhand der Abbildung 3.1 zu erkennen, ist die erste Ergebnisdarstellung sehr knapp gehalten und bitten erstmal nicht sonderlich viele Informationen. Dem Nutzer werden an dieser Stelle ein paar wenige grundlegende Möglichkeiten an die Hand gegeben, um den soeben erstellen Ergebnisdatensatz auszuwerten.

Im Folgenden werden einige dieser Basisfunktionen aufgezeigt und jeweils kurz erläutert.

Listing 3.5: die show-Funktion in LIAM2

```
1 # allgemeine Form der show-Funktion
2 show(expr1[, expr2, expr3, ...])
3
4 # Beispiel 1
5 show(count(age >= 18))
```

[5] hdf5 ist ein spezielles Dateiformat, welches für das Speichern und Verwalten von Datensätzen durch die „HDF Group“ entwickelt wurde [HDF GROUP-2016]

Abbildung 3.1: Pythonkonsole nach Simulationsende

```
20      start_period: 2016
21      periods: 10
```

```
Console
==========================================
simulation done
==========================================
 * 0.20 second elapsed
 * 10000 individuals on average
 * 1282050 individuals/s/period on average
==========================================

top 10 processes:
 - ageing: 0 ms (100%)
total for top 10 processes: 0 ms

Welcome to LIAM2 interactive console.
    help:            print this help
    help [function]: print the available arguments for a function
    q[uit] or exit:  quit the program

    entities:        list the available entities
    entity [name]:   set the current entity to another entity
    periods:         list the available periods for the current entity
    period [period]: set the current period
    fields [entity]: list the fields of that entity (or the current entity)
    globals:         list the available globals
    functions:       list the available builtin functions

    show is implicit on all commands

current entity set to person
current period set to 2025
>>>
```

Quelle: Screenshot aus LIAM2

```
6
7  # Beispiel 2
8  show("Count:", count(),
9       "Average age:", avg(age),
10      "Age std dev:", std(age))
```

In unserem Fall gibt der Aufruf von Beispiel 1 die Anzahl der Personen aus, die älter als 18 Jahre sind. Daher lautet das Ergebnis nach diesem Durchlauf „9131“. Das zweite Beispiel ermittelt die Anzahl der lebenden Personen und deren durchschnittliches Alter. Das entsprechende Ergebnis dieser Anfrage sieht folgendermaßen aus:

„Count: 10000 Average age: 49.9187 Age std dev: 22.9902390225 done.“

Alle Funktionen in dieser Art stellen nur sehr rudimentäre Möglichkeiten dar, um faktische Abfragen, die nicht zu komplex sind, da die Übersichtlichkeit beim Arbeiten mit einer Konsole (und Ergebnissen, die über viele Zeilen hinweg dargestellt werden) schnell verloren geht. Daher liefert LIAM2 eine Funktion, welchen den Export der berechneten Datensätze in eine .csv-Datei ermöglicht. Ein sehr simples Beispiel hierfür lautet beispielsweise:

Listing 3.6: .csv-Export mit LIAM2 aus der Pythonkonsole heraus.

```
1  csv(avg(income))
```

Gehen wir davon aus, dass die Startperiode auf 2016 und die Anzahl der Perioden auf 2 gesetzt wurde, bewirkt dieser Funktionsaufruf folgendes: für jede simulierte Periode wird eine Datei erzeugt. Hier sind dies die Dateien „person_21016csv“ und „person_2017.csv“, welche jeweils für das entsprechende Jahr das durchschnittliche Einkommen der Population dokumentieren. In Verbindung mit den sogenannten „table expressions“ kann das Aussehen und Verhalten der .csv-Datei noch etwas weiter strukturiert werden, worauf an dieser Stelle allerdings nicht weiter eingegangen wird.

Diagramme

LIAM2 bietet wie schon erwähnt verschiedene Möglichkeiten, um erzeugte Ergeb-

nisdatensätze mittels Diagrammen grafisch darzustellen. Somit ergeben sich, neben der reinen Ausgabe von Daten in Textform, einige Möglichkeiten für die Entwickler mit den Ergebnissen weiter zu arbeiten bzw. sie zu analysieren. Alle Funktionen innerhalb von LIAM2, die mit diesen Darstellungsmöglichkeiten in Verbindung stehen, basieren auf der externen Bibliothek „matplotlib“.[6]
Als Erstes betrachten wir die „bar()“-Methode, mittels derer Balkendiagramme erzeugt werden können. Diese Möglichkeit ist seit der Version 0.8 in LIAM2 enthalten. Der Funktionsaufruf um die verschiedenen Altersgruppen einer Simulation darzustellen, sieht beispielsweise wie folgt aus:

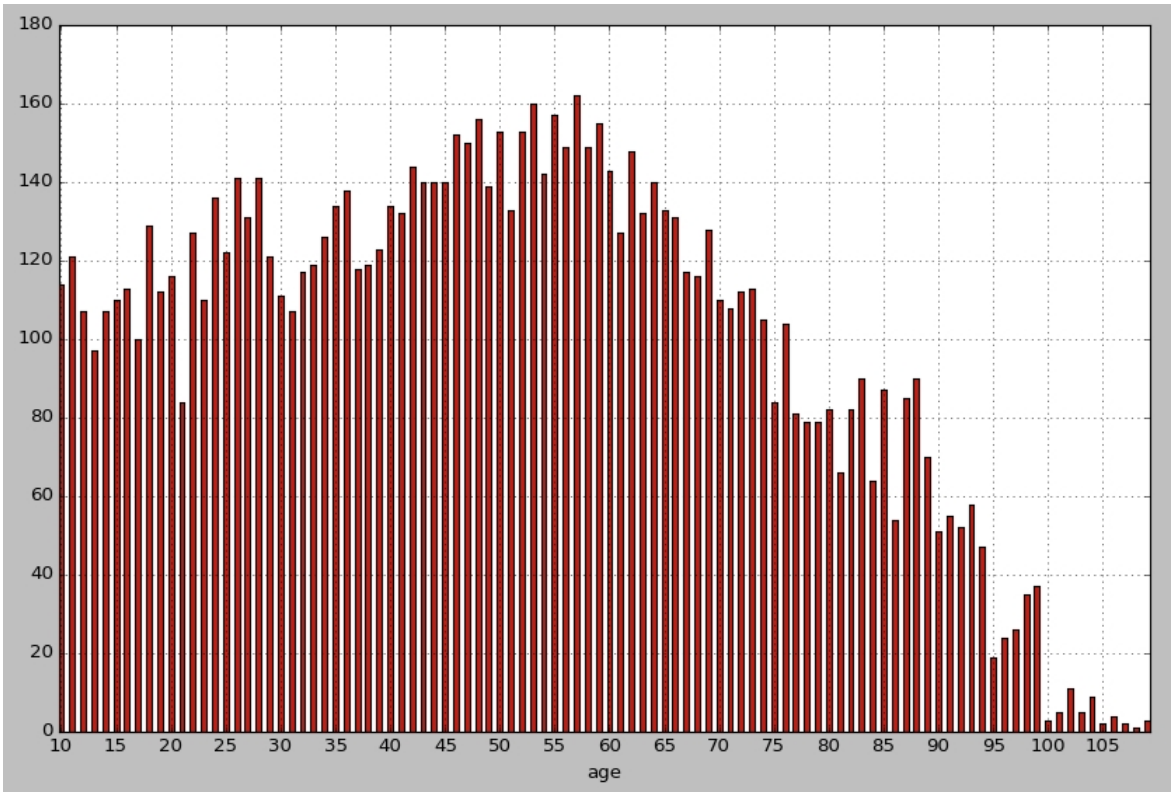
Listing 3.7: Balkendiagramm in LIAM2 erzeugen - Altersverteilung

```
1 bar(groupby(agegroup))
```

Nachdem diese Anweisung in der Pythonkonsole mit einem Klick auf „Enter“ bestätigt wurde, wird die Berechnung des Balkendiagramms angestoßen und nach Abschluss in einem neuen Fenster dargestellt. Abbildung 3.2 zeigt, wie das Ergebnis von dieser Berechnung aussieht.

[6] diese Software, entwickelt von John D. Hunter und seinem Team, ist eine 2D-Plotting-Bibliothek für Python, welche die LIAM2-Entwickler in ihr Projekt für grafische Auswertungen integriert haben. [MATPLOTLIB, 2016]

Abbildung 3.2: Balkendiagramm - Altersverteilung

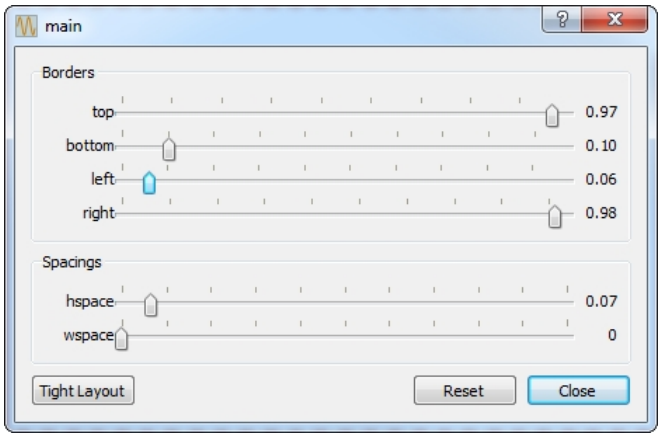


Quelle: eigener Screenshot

Auffällig hierbei ist, dass die y-Achse eine fehlende Beschriftung aufweist - allerdings ist bei diesem konkreten Beispiel schon aufgrund der selbst gestellten Abfrage klar, dass diese Achse die Anzahl der Personen in der jeweiligen Altersklasse repräsentiert. Falls die gewünschte Darstellung zu „eng“ oder anders ungewünscht erscheint, gibt es eine Funktion, die das Erstellen eines sogenannten „subplot“[7] ermöglicht. Dies bedeutet im Grunde, dass das erstellte Balkendiagramm hinsichtlich seiner beispielsweise horizontalen oder vertikalen Ausdehnung nachträglich angepasst werden kann. Dies ist zum Beispiel notwendig, wenn die X-Achse eine zu hohe Anzahl an Einträgen beinhaltet und somit die Balken zu dicht aneinander gestellt werden. In Abbildung 3.3 sieht man die Einstellungsmöglichkeiten, die dem Benutzer der Software zur Verfügung gestellt werden.

[7] „subplot“ meint hier, dass der vorhandene Plot (sprich die visuelle Ergebnisausgabe) verfeinert oder unterteilt werden kann, um eine bessere Übersichtlichkeit und Darstellung zu garantieren.

Abbildung 3.3: Balkendiagramm - Fenster zur Subploterstellung



Quelle: eigener Screenshot

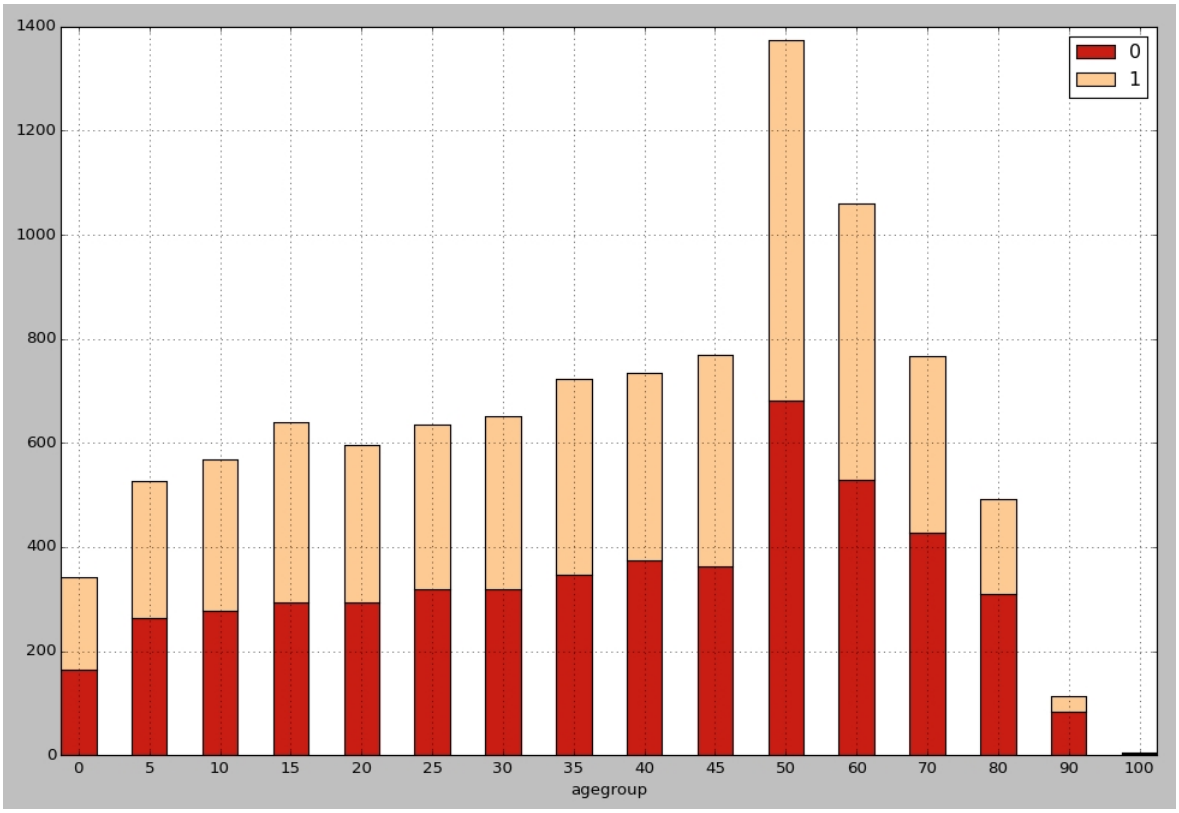
Mit der „bar()“-Methode ist es außerdem möglich Abfragen der Ergebnisdaten mit mehreren Argumenten durchzuführen. Gab es bei dem vorherigen Beispiel nur ein Argument in der Abfragemethode, sind es im folgenden Beispiel nun zwei. In den Fällen, in denen mehr als ein Argument benutzt wird, werden die entsprechenden Ergebnisse innerhalb eines Balkendiagramms übereinander gestapelt dargestellt. Eine entsprechende Abfrage kann beispielsweise folgendermaßen aussehen:

Listing 3.8: Die bar()-Methode mit zwei Argumenten

```
1 bar(groupby(agegroup, filter=not gender), groupby(agegroup,
     filter=gender))
```

In diesem Fall wird der Ergebnisausgabe oben rechts eine Legende hinzugefügt - in diesem Beispiel haben wir zwei Farbwerte im Balkendiagramm, die jeweils für „0“ und „1“ stehen - dies sind die im Programm festgelegten boole‘schen Werte, die das Geschlecht einer Person beschreiben, wie an anderer Stelle in dieser Arbeit bereits aufgezeigt wurde.
In Abbildung 3.4 kann man sehr gut erkennen, wie die beiden in der Abfrage genutzten Argumente grafisch dargestellt werden - übereinander gestapelt und mit der entsprechenden Legende versehen.

Abbildung 3.4: Balkendiagramm - bar()-Methode mit zwei Argumenten



Quelle: eigener Screenshot

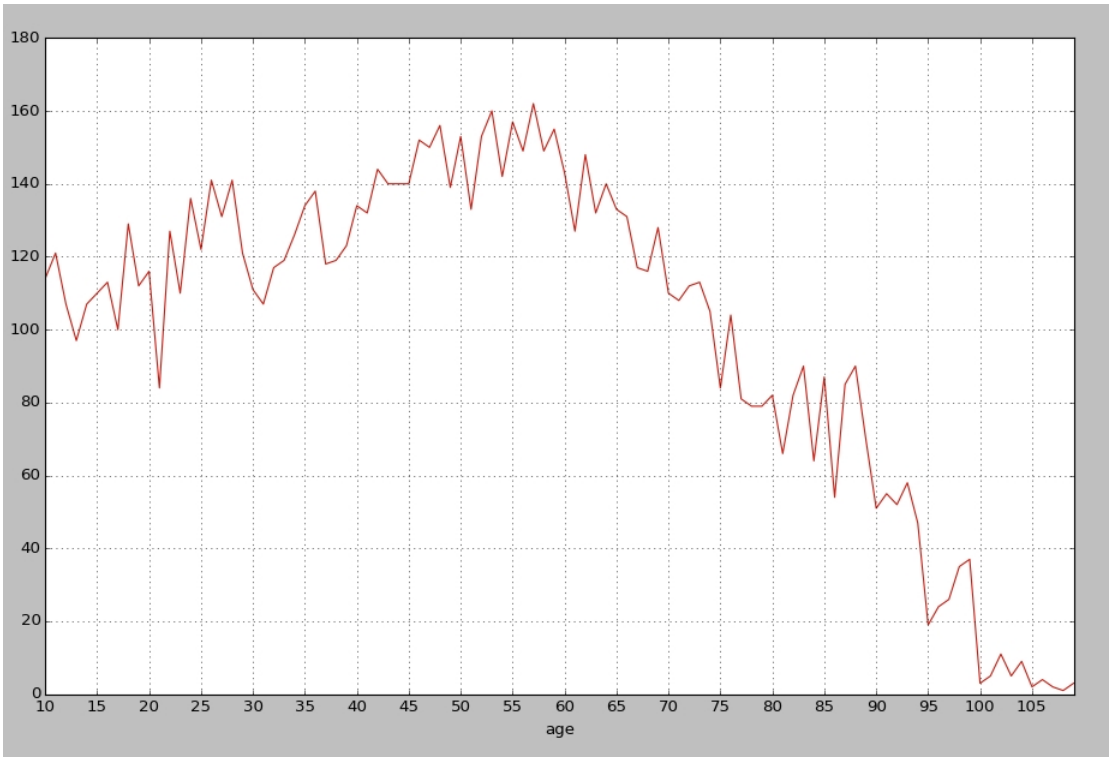
Im Weiteren wird die „plot()“-Methode genauer betrachtet. Sie versetzt den Benutzer in die Lage Liniendiagramme erstellen zu lassen. Die Funktionsaufrufe ähneln denen, die wir bereits bei der „bar()“-Methode genutzt haben. Analog dazu wollen wir das zuvor genutzte Beispiel nun auch mit dieser Methode verwenden und erhalten dadurch folgenden Funktionsaufruf:

Listing 3.9: Die plot()-Methode

```
1 plot(groupby(age))
```

Die folgende Abbildung 3.5 zeigt, wie LIAM2 diese Berechnung grafisch in ein Liniendiagramm umwandelt.

Abbildung 3.5: Liniendiagramm- die plot()-Methode



Quelle: eigener Screenshot

Generell ist anzumerken, dass das zusätzliche Fenster, welches für die Darstellung eines Diagramms aus „Notepad++“ heraus geöffnet wird, in seiner Größe ändern lässt. Horizontale und vertikale Veränderungen des Fensters wirken sich auch direkt auf die Darstellung des Diagramms aus, wodurch der Nutzer die Möglichkeit bekommt, eine für seine Bedürfnisse passende Skalierung zu finden. Dies gilt nicht nur für die „plot()“-Methode, sondern auch für alle anderen.
Die „plot()“-Methode bietet verschiedene Parameter, mittels derer man das Erscheinungsbild der Diagramme anpassen kann. Zwei dieser Möglichkeiten zeigt der folgende Abfragecode:

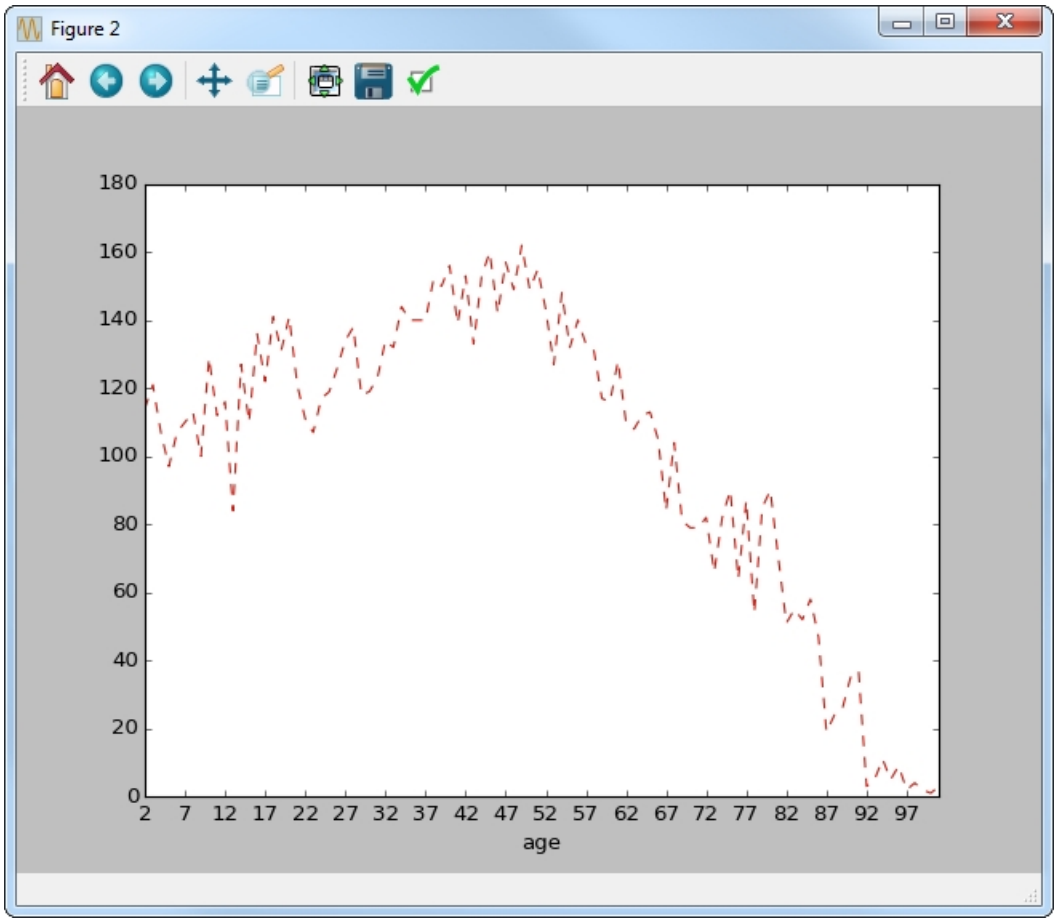
Listing 3.10: Parameter der plot()-Methode

```
1 plot(groupby(age), grid=False, linestyle='dashed')
```

Die beiden zusätzlichen Angaben „grid=False“ und „linestyle='dashed'“ bewirken in diesem Fall, dass das Diagramm zum Einen kein Hintergrundraster mehr aufweist, und

zum Anderen die standardmäßige durchgehende Linie durch eine gestrichelte ersetzt wird. Das resultierende Diagramm dieser Abfrage wird in Abbildung 3.6 dargestellt.

Abbildung 3.6: Liniendiagramm ohne Hintergrundraster und mit gestrichelter Linie



Quelle: eigener Screenshot

An dieser Stelle bewirken diese beiden Anpassungen nicht unbedingt eine Verbesserung der Lesbarkeit des Diagramms - es geht hier lediglich darum, die Variationsmöglichkeiten bei der Ausgabe zu betrachten.
Tabelle 3.2 zeigt eine Übersicht einiger Parameter inklusive deren Bedeutung und entstammt der API-Dokumentation der „matplotlib“-API-Dokumentation. [MATPLOTLIB API PLOT, 2016]

Tabelle 3.2: Parameter der plot()-Methode und deren Beschreibung

Eigenschaft	Beschreibung
agg_filter	unknown
alpha	float (0.0 transparent through 1.0 opaque)
animated	[True | False]
antialiased	[True | False]
color	any matplotlib color
dash_capstyle	['butt' | 'round' | 'projecting']
dashes	sequence of on or off ink points
drawstyle	['default' | 'steps' | 'steps-pre' | 'steps-mid' | 'steps-post']
fillstyle	['full' | 'left' | 'right' | 'bottom' | 'top' | 'none']
label	string or anything printable with '%s' conversion
linestyle	['solid' | 'dashed','dashdot','dotted' | (offset, on-off-dash-seq)
	| '-' | '–' | '-.' | ' ' | 'None' | ' ' | '']
linewidth	float value in points
markersize	float

Einige dieser Parameter werden nun in einer Abfrage zusammengefasst, um an-

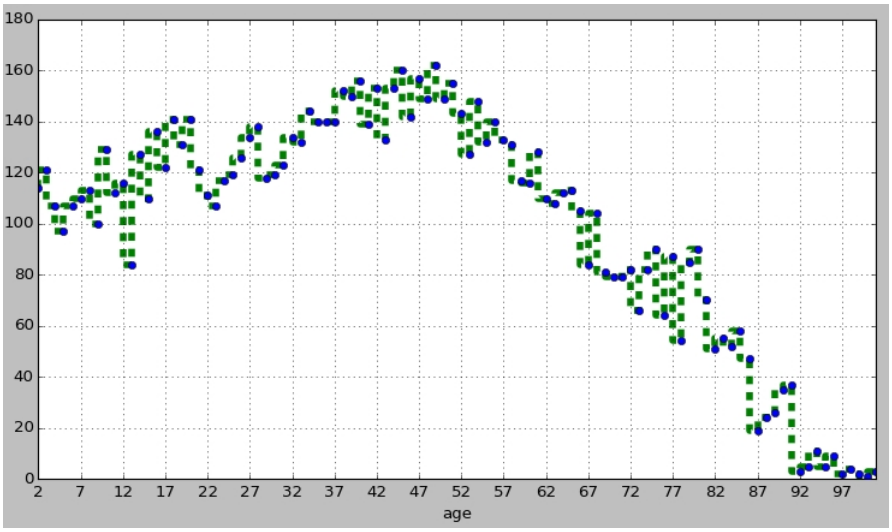
hand des daraus resultierenden Diagramms sehen zu können, welcher Einfluss auf die grafische Ergebnissausgabe genommen werden kann. Die folgende Abfrage wurde, wie auch die vorherigen Beispiele, auf den Ergebnisdatensatz einer LIAM2-Demo-Simulation angewandt:

Listing 3.11: Parameter der plot()-Methode

```
1 plot(groupby(age), drawstyle='steps', linestyle='dashed',
     marker='o', color='green', linewidth=5.0, markerfacecolor=
     'blue')
```

Die grafische Ausgabe hierzu wird in Abbildung 3.7 dargestellt. Dabei ist zu erkennen, dass eine beliebige Kombination der Parametern nicht unbedingt Sinn macht und das Ergebnis in ungünstigen Fällen auch verschlechtern, sprich schlechter „lesbar“ für den Anwender macht. Daher ist es an ihm gelegen, dass er in Bezug auf die gewünschten Informationen relevante Einstellungen und Parameter für die Diagramme aussucht. Dies ist für LIAM2-Neulinge mit Sicherheit zu Beginn mit einem erhöhten Zeitaufwand verbunden, da man zum Einen erstmal die entsprechenden Parameter finden und zum Anderen deren Verhalten und Auswirkungen bzgl. der Ausgabe kennenlernen muss.

Abbildung 3.7: Auswirkung verschiedener Parameter der plot()-Methode



Quelle: eigener Screenshot

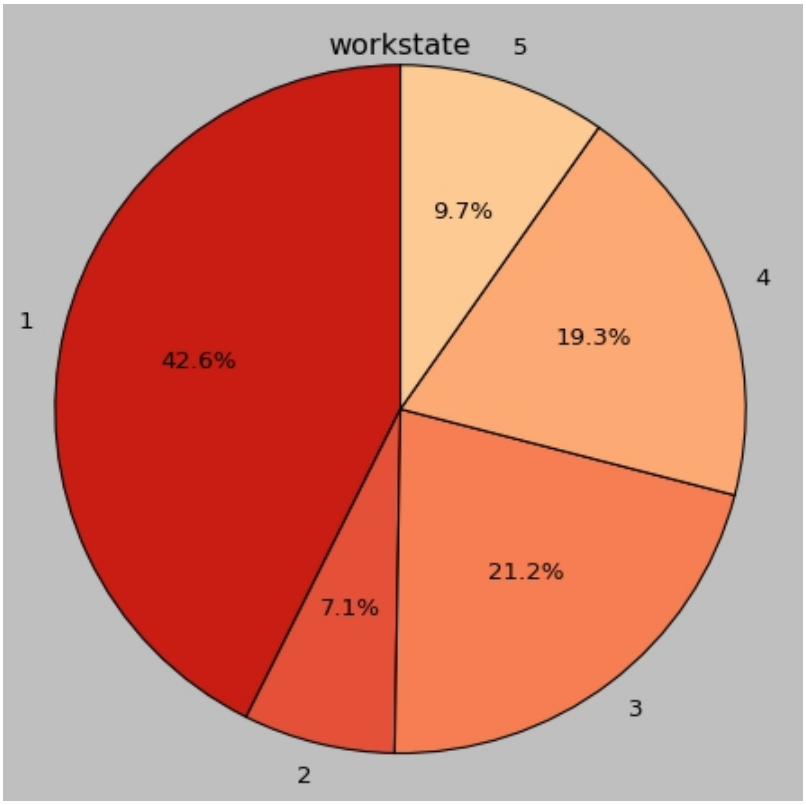
Im nächsten Schritt soll die „pie()“-Methode betrachtet, mit deren Hilfe es möglich ist Kreisdiagramme zu erstellen. Diese Methode ist in LIAM2 seit der Version 0.8. intergriert und damit relativ neu. Sie benutzt die „matplotlib.pyplot.pie“-Bibliothek [MATPLOTLIB API PIE, 2016] und erbt somit alle dort vorhandenen Argumente. Funktionsaufruf und Übergabe der Argumente sind analog zu den vorherigen Methoden, woraus sich beispielsweise diese erste einfache Beispielabfrage ergibt:

Listing 3.12: Beispiel für die pie()-Methode

```
1 pie(groupby(workstate))
```

Die hieraus resultierende grafische Ausgabe ist in Abbildung 3.8 zu sehen. LIAM2 errechnet in diesem Fall jeweils einen prozentualen Anteil, gemessen an der Anzahl von Personen, die sich am Ende einer Simulation in einem bestimmten „workstate“ befinden. Die Zahlen 1 bis 5 der Legende sind die Indizes der unterschiedlichen Möglichkeiten, die innerhalb des Ergebnisdatensatzes vertreten sind.

Abbildung 3.8: Beispiel eines Kreisdiagramms



Quelle: eigener Screenshot

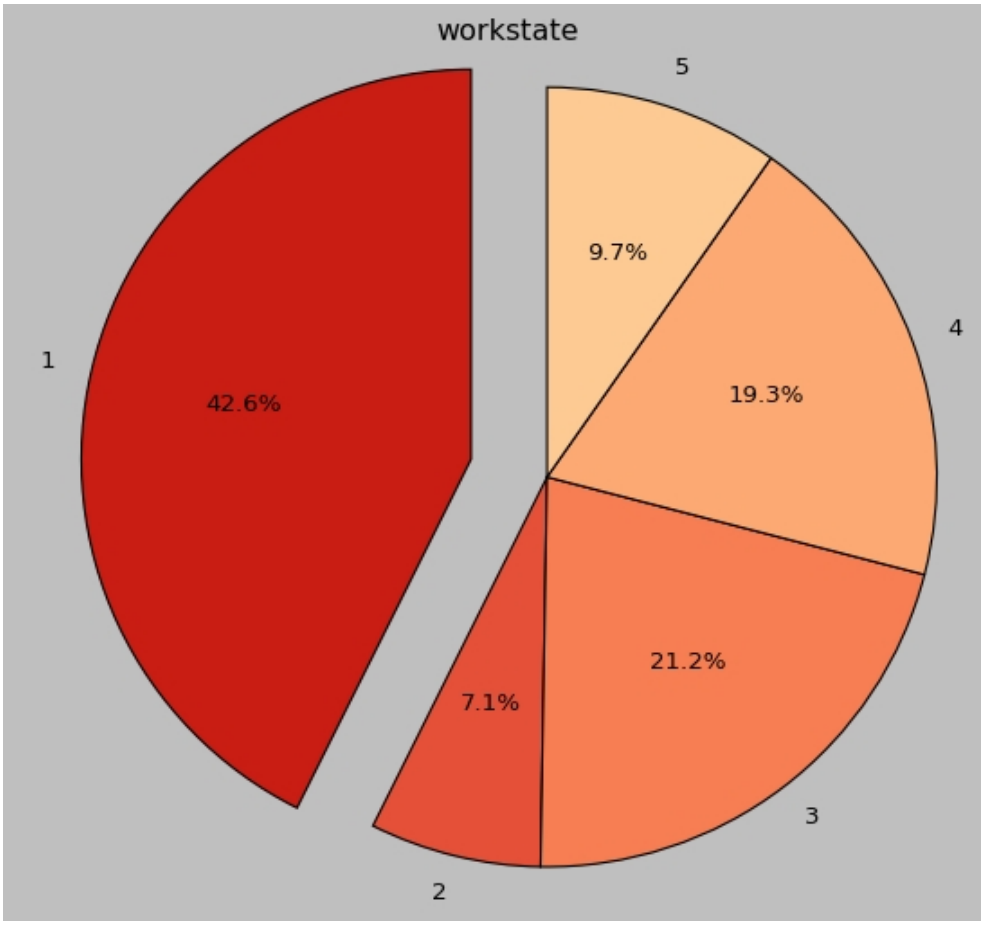
Ein weiteres Argument bei der Verwendung von Kreisdiagrammen in LIAM2 ist „explode“. Hiermit können die einzelnen Teile eines Kreises voneinander entfernt werden um eine etwas andere und in bestimmten Fällen bessere Darstellung zu erhalten. „explode“ erwartet als Parameter ein Array von Werten, welche jeweils den Abstand der einzelnen Teile voneinander beschreibt - hierzu ist es daher notwendig, dass man weiß, wie viele Kreisteile die Grafik beinhalten wird. Auf unser vorheriges Beispiel angepasst, kann die Abfrage zu Erstellung des Diagramms beispielsweise wie folgt aussehen:

Listing 3.13: Beispiel für die pie()-Methode

```
1 pie(groupby(workstate), explode=[0.2, 0, 0, 0, 0])
```

Das zu „explode“ gehörende Array mit seinen Float-Werten gibt an, dass nur der Teil des Kreises, der den ersten „workstate“ repräsentiert von den restlichen Teilen abgegrenzt und um den Faktor 0.2 verschoben werden soll - Abbildung 3.9 zeigt das Resultat dieser Anweisung.

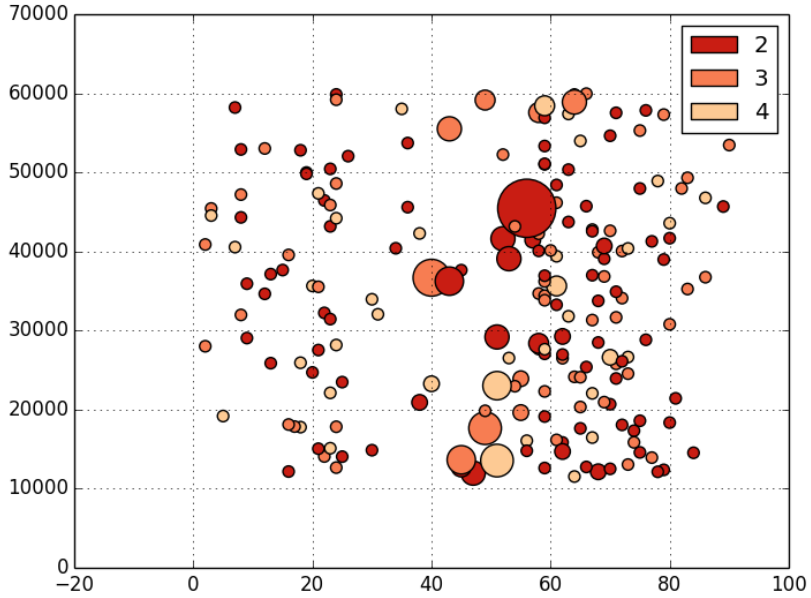
Abbildung 3.9: Verwendung von explode beim Kreisdiagramm



Quelle: eigener Screenshot

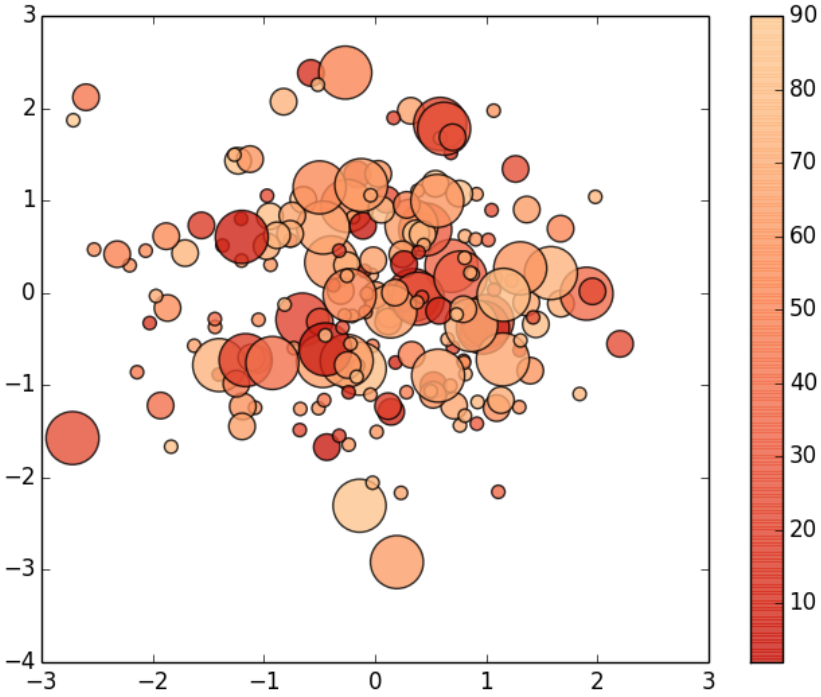
Ebenfalls neu bei LIAM2 in der Version 0.8. sind sogenannte „scatter plots“, welche mittels der „scatter()“-Methode verwendet und erzeugt werden können. Diese Methode erbt ihre Eigenschaften von der „matplotlib.pyplot.scatter“-Bibliothek. [MATPLOTLIB API SCATTER, 2016] Hiermit können Diagramme in der Art wie sie die Abbildungen 3.10und 3.11 zeigen, erstellt werden. [LIAM2 SCATTER PLOTS, 2016]

Abbildung 3.10: Streudiagramm Beispiel 1



Quelle: [LIAM2 SCATTER PLOTS, 2016]

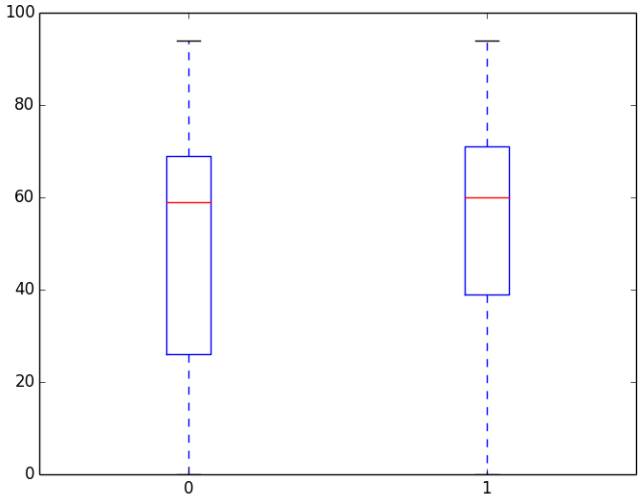
Abbildung 3.11: Streudiagramm Beispiel 2



Quelle: [LIAM2 SCATTER PLOTS, 2016]

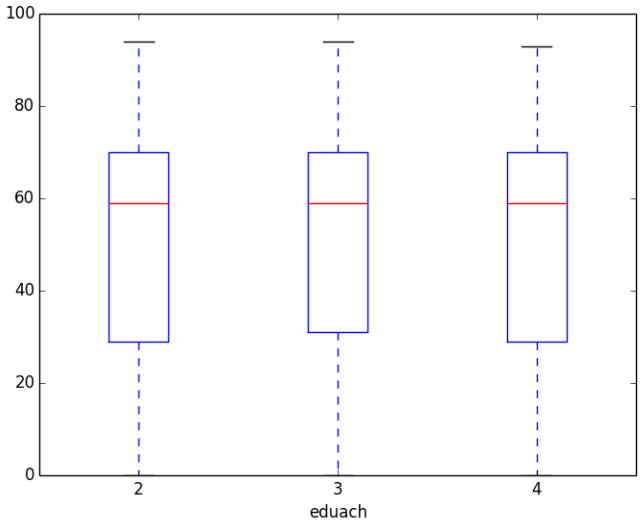
Die letzte Neuerung in der Version 0.8. von LIAM2 ist die „boxplot()“-Funktion, welche das Erstellen von Kastengrafiken ermöglicht. Die Abbildungen 3.12 und 3.13 zeigen, wie solche Auswertungen in grafischer Form aussehen können. [LIAM2 BOXPLOT, 2016]

Abbildung 3.12: Kastengrafik Beispiel 1



Quelle: [LIAM2 BOXPLOT, 2016]

Abbildung 3.13: Kastengrafik Beispiel 2

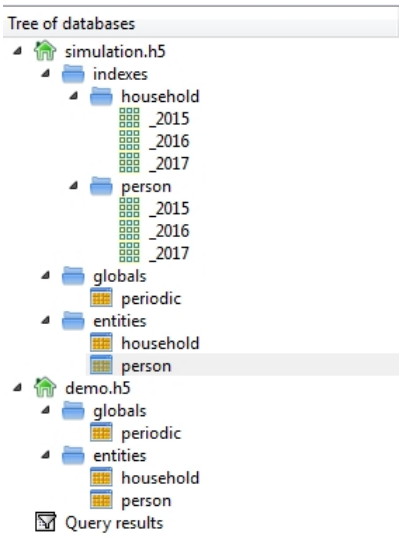


Quelle: [LIAM2 BOXPLOT, 2016]

ViTables

Wie schon einleitend in diesem Kapitel beschrieben, liefert die Standardinstallation mit „ViTables“[8] ein Programm, mittels diesem man sich die Ausgangs- sowie die Ergebnisdatensätze tabellarisch anzeigen lassen und nach eigenen Bedürfnissen filtern kann. Da diese Repräsentation der Daten keinen wirklichen grafischen Anspruch hat und im Prinzip nur eine übersichtlichere Alternative für die Ausgabe der Python-Konsole ist, wird an dieser Stelle nicht intensiver darauf eingegangen, da die nötige Relevanz im Kontext dieser Diplomarbeit nicht vorhanden ist. Der Vollständigkeit halber sei es dennoch kurz erwähnt.
Alternativ zu dem bisher aufgezeigten Prozedere wird „ViTables“ durch Betätigen der Taste „F9“ bei einem geöffneten Modell in „Notepad++“ gestartet. Daraufhin öffnet sich das Programm, welches in zwei Bereichen organisiert ist. Abbildung 3.14 zeigt den linken Bereich, welcher die Navigation durch die Datensätze ermöglicht.

Abbildung 3.14: Datenrepräsentation mittels Baumstruktur in ViTables



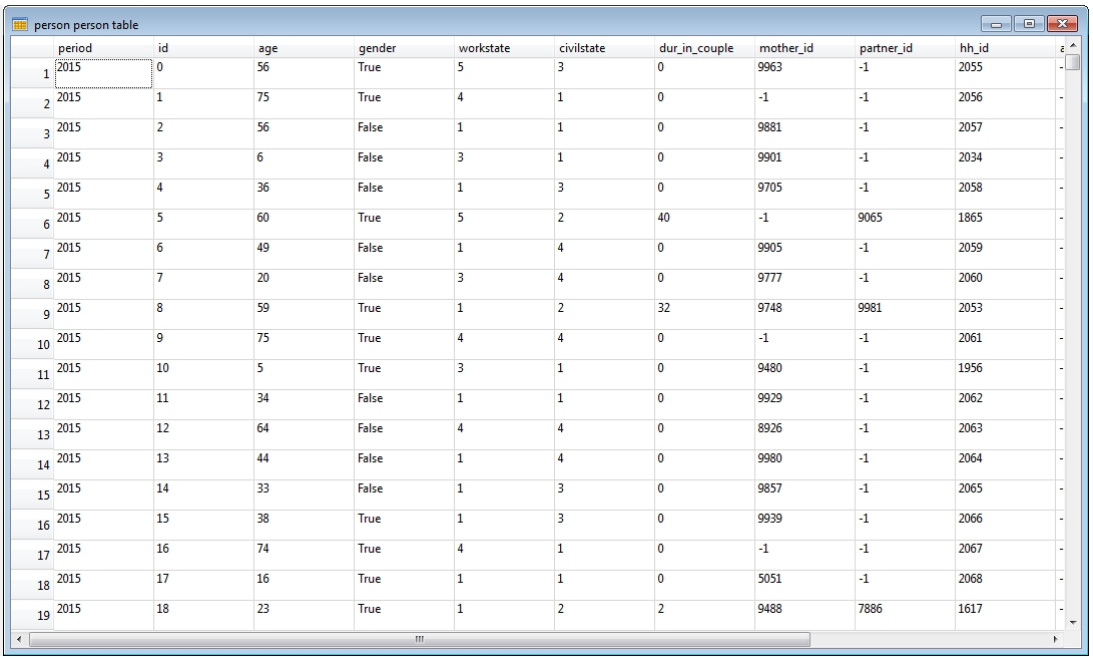
Quelle: eigener Screenshot

Im rechten Bereich des Programmfensters werden entsprechend der links getroffenen Auswahl eine oder mehrere Tabellen angezeigt. In unserem Beispiel ist es die Entität

[8] ViTables ist eine grafische Benutzeroberfläche die es ermöglicht bspw. .hdf5-Dateien in tabellarische Ansichten umzuwandeln

„person“, woraus sich die in Abbildung 3.15 gezeigte Tabellenansicht ergibt.

Abbildung 3.15: Tabellarische Datenrepräsentation in ViTables



	period	id	age	gender	workstate	civilstate	dur_in_couple	mother_id	partner_id	hh_id
1	2015	0	56	True	5	3	0	9963	-1	2055
2	2015	1	75	True	4	1	0	-1	-1	2056
3	2015	2	56	False	1	1	0	9881	-1	2057
4	2015	3	6	False	3	1	0	9901	-1	2034
5	2015	4	36	False	1	3	0	9705	-1	2058
6	2015	5	60	True	5	2	40	-1	9065	1865
7	2015	6	49	False	1	4	0	9905	-1	2059
8	2015	7	20	False	3	4	0	9777	-1	2060
9	2015	8	59	True	1	2	32	9748	9981	2053
10	2015	9	75	True	4	4	0	-1	-1	2061
11	2015	10	5	True	3	1	0	9480	-1	1956
12	2015	11	34	False	1	1	0	9929	-1	2062
13	2015	12	64	False	4	4	0	8926	-1	2063
14	2015	13	44	False	1	4	0	9980	-1	2064
15	2015	14	33	False	1	3	0	9857	-1	2065
16	2015	15	38	True	1	3	0	9939	-1	2066
17	2015	16	74	True	4	1	0	-1	-1	2067
18	2015	17	16	True	1	1	0	5051	-1	2068
19	2015	18	23	True	1	2	2	9488	7886	1617

Quelle: eigener Screenshot

3.2 Modgen

Das zweite Programm, das wir in diesem Kapitel eingehender betrachten werden ist „Modgen“[9]. Dessen Entwicklungsbeginn ist auf das Jahr 1994 zu datieren und wird von „Statistics Canada“[10] verwaltet und zur Verfügung gestellt. [MODGEN HOMEPAGE, 2016]

Zum Zeitpunkt des Verfassens dieser Diplomarbeit ist Version 11 die aktuellste, die auf der Homepage zum Download angeboten wird. „Modgen“ bietet die Möglichkeit

[9] Modgen steht für „model generator“

[10] „Statistics Canada“ ist eine kanadische Bundesbehörde, die Statistiken über die kanadische Bevölkerung, Bodenschätze, Wirtschaft und Kultur herausgibt [STATCAN HOMEPAGE, 2106]

dynamische Mikrosimulationsmodelle zu erstellen, wobei aus einer Vielzahl von unterschiedlichen Modellarten geschöpft werden kann. Der Benutzer dieser Software hat die Möglichkeit vorhandene Modelle zu benutzen, anzupassen oder eigene Modelle von Grund auf zu entwicklen.

3.2.1 Allgemeines

Zugang

Modgen ist ebenso wie LIAM2 kostenlos im Internet zum Download verfügbar. Pro Version gibt es hier jeweils zwei .exe-Dateien für unterschiedliche Ansprüche. Die „Modgen 11 Prerequisites"-Datei mit einer Größe von 6,42 MB installiert alles was benötigt wird, um ein Modell durchlaufen zu können. Möchte man hingegen als Entwickler eigene Modelle schreiben oder bestehende verändern, benötigt man im Anschluß an die Installation der Prerequisites-Datei noch die „Modgen 11"-.exe, welche eine Größe von 16.1 MB hat. Mit „Visual Studio 2010"[11] wird eine (in den meisten Fällen) kostenpflichtige Software benötigt, damit das Arbeiten mit Modgen-Modellen überhaupt möglich ist. [MODGEN DOWNLOAD, 2016]

Technische Voraussetzungen

Die auf der Webseite zum Download angebotenen Pakete wurden laut Entwickler für die Windows-Versionen Vista, 7 und 8 getestet. Zusätzlich benötigt man, wie bereits oben erwähnt, Microsoft Visual Studio - für Modgen 11 in der Version 2010, für Modgen 10 in der Version 2008. Hier wird jeweils mindestens die „Standard Edition" benötigt, da die „Express Edition" mit ihren technischen Möglichkeiten nicht ausreichend für den Betrieb von Modgen ist. Falls man nicht Teil eines studentischen oder anderen bildungstechnischen Softwareprogramms ist, finde ich es ungünstig eine Software wie diese als Voraussetzung für den Betrieb der eigenen vorauszusetzen.

[11] Visual Studio ist eine von dem Unternehmen Microsoft angebotene, integrierte Entwicklungsumgebung für verschiedene Hochsprachen wie $C++$.

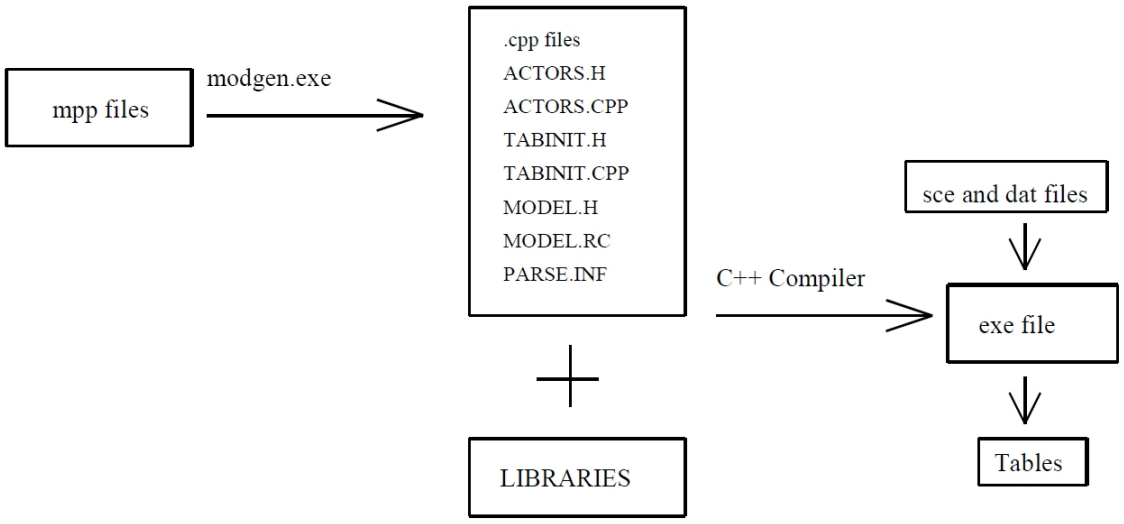
Hierdurch wird der mögliche Nutzerkreis selbstverständlich eingeschränkt, wenn auch vermutlich die Mehrheit der Modgen-User in einem Kontext arbeiten oder studieren wird, in dem der Zugang zu derartiger Software nicht privat hergestellt werden muss, sondern vergünstigt oder sogar kostenlos über eine entsprechende Bildungseinrichtung läuft.
Wie schon bei LIAM2 festgestellt wurde, fehlt bei Modgen ebenfalls eine Variante für das Macintosh- oder LINUX-Betriebssystem.

Notwendige Programmierkenntnisse

Damit ein Entwickler mit Modgen eigene Simulationen schreiben oder bestehende anpassen kann, benötigt er mindestens gute Kenntnisse in der Programmiersprache $C++$[12]. Neben den üblichen $C++$-Funktionen gibt es zusätzliche Schlüsselwörter, die der Entwickler bei der Erstellung von .mpp-Dateien[13] verwenden kann. Beispiele für diese vordefinierten Keywords sind beispielsweise „event“ oder „actor“, aber auch spezifische Funktionen, wie „RandUniform()“.
Die durch den Modgen-Precompiler erzeugten $C++$-Dateien werden anschließend gemeinsam mit den Simulationsbibliotheken und den anderen Modelldateien zu einer .exe-Datei kompiliert, mit welcher im Anschluss die eigentliche Simulation gestartet werden kann.
Abbildung 3.16 zeigt den gesamten Ablauf im Kompilierungsprozess und wurde dem „Modgen Developer‘s Guide“ entnommen. [MODGEN DEVELOPERS GUIDE, 2016]

[12] $C++$ ist eine von der ISO genormte objektorientierte Programmiersprache, welche eine Erweiterung der Programmiersprache C darstellt.
[13] .mpp-Dateien sind in Modgen notwendig um Modell-Code, -Parameter und -Szenarien zu organisieren. Diese werden später durch die Modgen.exe in $C++$-Modell-Code umgewandelt.

Abbildung 3.16: Kompilierungsprozess in Modgen



Quelle: [MODGEN DEVELOPERS GUIDE, 2016]

Flexibilität

Flexibilität wird an dieser Stelle wieder unter dem Anspruch der Verwendung von eigenen Datensätzen betrachtet. Im Gegensatz zu dem Prozedere bei LIAM2 ist es bei Modgen leider nicht möglich, automatisiert eigene Datensätze im .csv-Dateiformat einzulesen. Dieser Vorgang wird bei Modgen nicht standardmäßig abgebildet. Um dies trotzdem innerhalb einer Modgen-Modellsimulation zu erreichen, muss nach Definition der Agenten der Entwickler das korrekte Einlesen einer externen Datei zusätzlich programmieren und in das gesamte Modell mit einbinden. Dieser Punkt bestätigt die zuvor erwähnte Tatsache, dass die ausführliche Nutzung von Modgen nur auf Basis ausgereifter Programmierkenntnisse geschehen kann.

3.2.2 Modellierungsansatz

Zeitverlauf

In dieser Hinsicht, ob Simulationen diskret oder kontinuierlich abgearbeitet werden, ist Modgen LIAM2 überlegen, da das Programm die Erstellung von eventbasierten

Simulationen ermöglicht. Hierdurch sind prinzipiell auch periodenbasierte Simulationen realisierbar, indem jedes Jahr ein Event pro Agent ausgeführt wird. Zusätzlich wäre es möglich fallbasierte Simulationen zu entwickeln. Diese Form der Simulation behandelt jeden Agenten im Ausgangsdatensatz separat und isoliert von den anderen Agenten. Das bedeutet, dass der Werdegang von Agent_{n+1} erst simuliert wird, nachdem der Werdegang von Agent_{n} komplett berechnet wurde. Die jeweils gewünschte Simulationsart bezogen auf den Zeitfaktor innerhalb der Simulation kann bei Modgen bei Anlage eines neuen Modells einfach ausgewählt werden. [TIEFENAU, 2014]

Population

Wie gerade beschrieben, erlaubt, wenn auch nicht so komfortabel wie LIAM2, Modgen die Verwendung von eigenen Ausgangsdatensätzen. Daher steht es dem Benutzer der Software auch hierbei frei, ob er mit einem Populations-/Kohorten oder Querschnittsmodell arbeitet. Möchte der Entwickler synthetische Modelle verwenden, ist zusätzlicher Programmieraufwand notwendig um aus vorhandenen Datentabellen synthetische Agenten zu generieren. Die Wahl zwischen offenen und geschlossenen Modellen kann der Benutzer ebenfalls entsprechend der eigenen Bedürfnisse und Anforderungen treffen.

Zufallsfunktionen

Modgen arbeitet grundlegend mit den Zufallsfunktionen, die auch standardmäßig bei LIAM2 zur Verfügung gestellt werden, weshalb an dieser Stelle auf Tabelle 3.1 verwiesen darf. Modgen liefert weiterhin verschiedene eigene $C++$-Funktionen, die die Zufallsfunktionen erweitern oder für bestimmte Fälle spezialisieren. Tiefgehender wird hier aber nicht darauf eingegangen. Die entsprechenden Funktionen und deren Eigenschaften sind im „Modgen Developer‘s Guide“ ausführlich beschrieben. [MODGEN DEVELOPERS GUIDE, 2016]

3.2.3 Usability

Datenmanagement

Simulationsmodelle in Modgen werden in $C++$ definiert. Zu Beginn werden die Agentendefinitionen in einer .mpp-Datei beschrieben und anschließend durch den in Abbildung 3.16 gezeigten Kompilierungsprozess umgewandelt. Das folgende Listing zeigt beispielhaft, wie eine Simulationsbeschreibung in Modgen aussehen kann.

Listing 3.14: Beispielcode für eine Simulationsbeschreibung in Modgen

```
1  // Definition von globalen Variablen
2  parameters{
3          int StartingPopulationSize;
4          double MortalityHazard;
5  }
6  actor Person{
7          logical alive = {TRUE};
8          event timeMortalityEvent, MortalityEvent;
9          void Start();
10         void Finish();
11 }
12 // Berechnet Zeitpunkt des Todes anhand Zufallszahl und der
       globalen Variablen MortalityHazard
13 TIME Person::timeMotralityEvent(){
14         TIME tEventTime = TIME_INFINITE;
15         tEventTime = WAIT(-TIME(log(RandUniform(1)/
              MortalityHazard));
16         return tEventTime;
17 }
18 // Setzt alive auf FALSE und fuehrt Aufraeumarbeiten aus in
       Finish()
19 void Person::MortalityEvent(){
20         alive=FALSE;
21         Finish();
22 }
23 void Person::Start(){...}
```

```
24 void Person::Finish(){...}
```

Für den actor „Person“ gibt es den Beschreibungsparameter „alive“, sowie die beiden definierten Events „timeMortalityEvent“ und „MortalityEvent“, wobei das erste Event den Zeitpunkt für das zweite berechnet. Die Methoden „Start()“ und „Finish()“ legen fest, wann der Agent einer Simulation hinzugefügt bzw. von dieser wieder entfernt wird.
Möchte der Entwickler einen eigenen Datensatz verwenden, wurde bereits erwähnt, dass dies nur über einen separaten Import der entsprechenden Datei geschehen kann, welche Zeile für Zeile ebendiesen ausließt. Der entsprechende Code, bezogen auf eine extern vorhandene Datei „persons.csv“ kann wie folgt aussehen:

Listing 3.15: Beispielcode für den Import einer .csv-Datei in Modgen

```
1 std::ifstream infile("persons.csv");
2 int age, gender;
3 while (infile >> age >> gender)
4 {
5     Person *paPerson = new Person();
6     paPerson->age = age;
7     paPerson->gender = gender;
8 }
```

Analog zu LIAM2 ist es in Modgen ebenfalls möglich, die Beziehung zwischen zwei Agenten zu beschreiben. Modgen stellt hierfür das Schlüsselwort enquotelink zur Verfügung, wie im Listing 3.16 zu sehen ist. Anschließend kann diese Beziehung mittels Zeigervariablen in den Events verwendet werden. [TIEFENAU, 2014]

Listing 3.16: Beispielcode für einen Link in Modgen

```
1 link Person.pPartner; // Definiert eine Verbindung zwischen
     zwei Objekten desselben Typs
2 link Person.hHousehold Household.Inhabitants[] // Definiert
     eine Verbindung zwischen 2 Objekten verschiedener Art
3 ...
4 pPartner = pointerPerson; // Zuweisung in den Events
```

3.2.4 Ergebnisausgabe und -darstellung

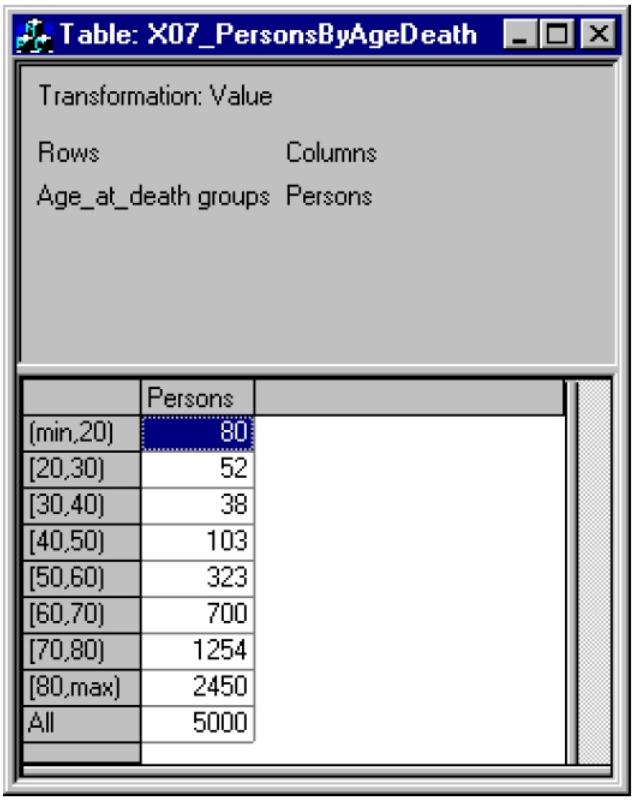
Der Output bzw. die Visualisierung der Ergebnisdaten ist in Modgen nicht ansatzweise so umfangreich wie es bei LIAM2 der Fall ist. Wie bereits gesehen, gibt es dort alleine im Bereich der Diagramme mehrere Möglichkeiten um Resultate entsprechend anzuzeigen. Modgen beschränkt sich bei diesem Aspekt fast ausschließlich auf textuelle bzw. tabellarische Ansichten, welche der Vollständigkeit halber im Folgenden aufgezeigt werden.

Ausgabetabellen

Um den Inhalt einer .mdb-Datei[14] anzeigen zu lassen, bietet Modgen die Möglichkeit sogenannter Ausgabetabellen an. Von diesen Tabellenfenstern kann nach einer Simulation eine beliebige Anzahl geöffnet werden. Ein Fenster besteht jeweils aus einem oberen und einem unteren Teil. Der obere Teil beinhaltet zum Beispiel die Angaben, welche Daten in der Spalte und welche in den Zeilen aufgelistet werden sollen. Zusätzlich wird angegeben, welche Art von Wert in der Tabelle angezeigt werden soll. In Modgen sind dies, abhängig von der Spezifizierung im Modell, die folgenden möglichen Werte: 'Value', 'Standard Error Values' oder 'Coefficient of Variation Values'. Im unteren Teil des Tabellenfensters werden dann die dementsprechenden Werte angezeigt, welche an dieser Stelle auf „read-only“ gesetzt und somit dort nicht veränderbar sind. Die folgende Abbildung 3.17 zeigt, wie ein solches Beispiel nach einem Simulationslauf innerhalb von Modgen aussehen kann:

[14] dies ist das Modgen-Dateiformat für die Modell-Ausgabedateien

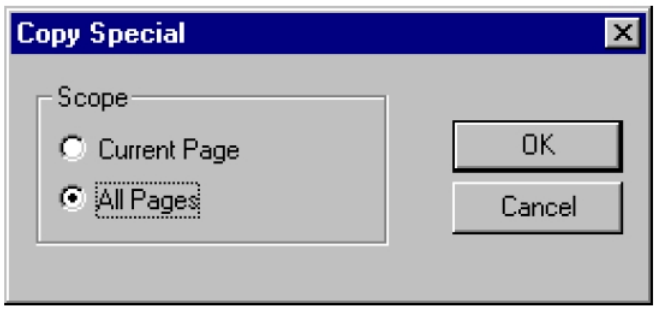
Abbildung 3.17: Tabellenfenster als Ergebnisausgabe in Modgen



Quelle: eigener Screenshot

Ein anderer Weg, um unformatierte Daten zu kopieren um sie beispielsweise für eine andere Software zugänglich zu machen ist die „Copy Special“-Funktion in Modgen. Mit ihr ist es möglich auch größere Tabellen, die mehr als zwei Dimensionen aufweisen, zu kopieren. Wie Abbildung 3.18 zeigt, kann mir auswählen ob man nur die aktuelle oder direkt alle Seiten kopiert.

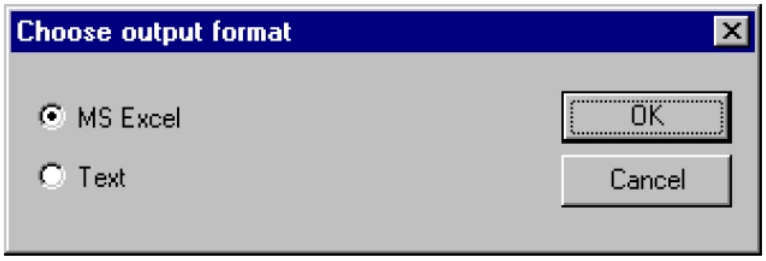
Abbildung 3.18: Copy-Special-Dialog in Modgen



Quelle: eigener Screenshot

Als „Seiten“ werden hier alle Dimensionen bezeichnet, die über die zweidimensionale Ausprägung von Zeilen und Spalten hinausgehen.
Als weitere Möglichkeit zum Export der Modelltabellen, die nach einem Simulationslauf in der Ergebnisdatenbank abgespeichert werden, gibt es die „Scenario/Export“-Funktion. Hierbei kann, wie in Abbildung 3.19 ersichtlich, zwischen einem Export für Microsoft Excel und einem Textexport gewählt werden.

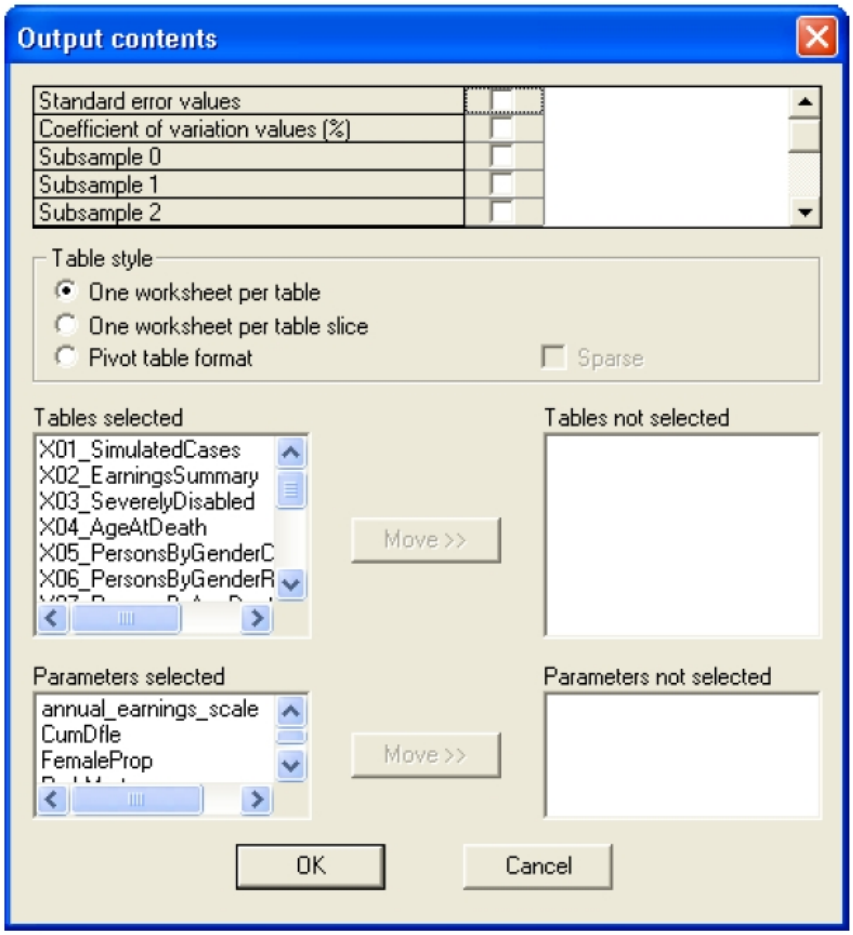
Abbildung 3.19: Scenario/Export in Modgen



Quelle: eigener Screenshot

Wurde hier „MS Excel“ als Ausgabeformat gewählt, werden entsprechende Excel Tabellen generiert und in einer .xls-Datei abgespeichert. In einem weiteren Dialog, siehe Abbildung 3.20, erhält man verschiedene Optionen um den Export zu konkretisieren. Es besteht die Möglichkeit anzugeben, welche Tabellen und/oder Parameter übernommen werden, über „Not Selected“ ist aber auch der explizite Ausschluss entsprechender Daten möglich.

Abbildung 3.20: Scenario/Export zu MS Excel in Modgen



Quelle: eigener Screenshot

Wird stattdessen „Text“ als Ausgabeformat ausgewählt, schreibt Modgen die ausgewählten Tabellen in eine .txt-Datei. Auch hier gibt es einen weiteren Dialog, der eine Auswahl der jeweiligen Tabellen ermöglicht. Dieser ist identisch zu dem in Abbildung 3.20 - mit dem Unterschied, dass die Bereiche für „Parameters“ entfallen. [MODGEN VISUAL INTERFACE GUIDE, 2016]

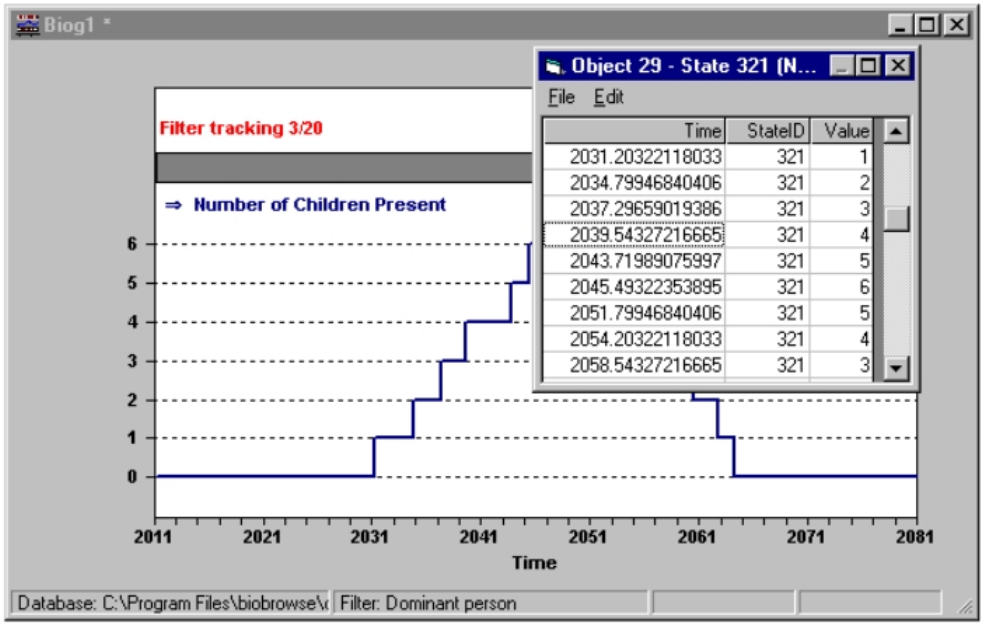
Biographical Browser

Mit dem „Biographical Browser“[15], kurz BioBrowser, liefert Modgen die einzige Möglichkeit

[15]der Modgen Biographical Browser ist eine Stand-Alone Software, die in der Modgen-Installation

im Bereich der Ergebnisausgabe, die über die reine tabellarische Darstellung der Daten hinausgeht. BioBrowser ist ein Tool, mit dessen Hilfe der Nutzer die Charakteristika und Attribute eines Agenten über dessen Lebenszeitraum hinweg visualisieren lassen kann. Um diese grafische Repräsentation realisieren zu können, greift BioBrowser auf eine spezielle Datenbankdatei zu, die während einer durchlaufenen Modellsimulation erzeugt wird. Es ist daher möglich einen Längsschnittdatensatz pro Agent zu erzeugen, der, bezogen auf bestimmte Parameter, die unterschiedlichen Zustände zu bestimmten Zeitpunkten innerhalb der Simulation grafisch darstellt.
Zusätzlich so einer solchen grafischen Darstellung, kann sich der Nutzer auch die der Grafik zugrundeliegenden Werte über den „Show Longitudinal Data“-Befehl anzeigen lassen, wie in Abbildung 3.21 zu sehen ist.

Abbildung 3.21: Biographical Browser in Modgen



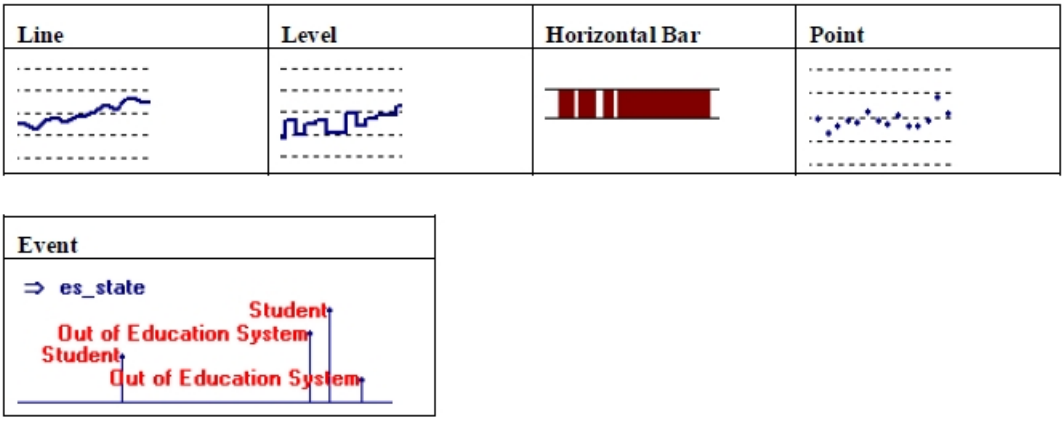
Quelle: eigener Screenshot

Es ist jederzeit möglich, die Darstellung im BioBrowser zu verändern, indem man zum Beispiel andere Werte oder Parameter auswählt. Die Ausgabe wird in diesem Fall neu erstellt und wieder auf den Beginn gesetzt. Dies ist möglich, da die jeweiligen Daten, wie bereits erwähnt, aus speziellen Datenbanktabellen gelesen werden.

mitgeliefert wird.

Für die generelle grafische Darstellung stellt BioBrowser fünf unterschiedliche Diagrammarten zur Verfügung: 'line', 'level', horizontal bar', 'point' und 'event'. In Abbildung 3.22 wird dargestellt, wie sich diese Varianten unterscheiden:

Abbildung 3.22: Biographical Browser Diagrammarten



Quelle: [MODGEN BIOBROWSER GUIDE, 2016]

Diese unterschiedlichen Diagrammarten stellen auch gleichzeitig die einzigen Optionen dar, wenn es um das Thema grafische Ergebnisausgabe mit BioBrwoser geht. Andere Möglichkeiten oder zusätzliche Tools von Modgen selbst gibt es zum Zeitpunkt des Verfassens dieser Diplomarbeit nicht.
[MODGEN BIOBROWSER GUIDE, 2016]

4 Datenvisualisierung

„Visualisierungen veranschaulichen Inhalte und können sie trotz Komplexität oft auf den Punkt bringen, wo Texte und Tabellen scheitern. Damit ihre Verwendung sinnvoll ist, um eine erfolgreiche Aufmerksamkeit zu gewährleisten, sollten vier Hauptaspekte erfüllt sein: Visualisierungen müssen neu, nützlich, schnell verständlich und optisch ansprechend sein.“ [PHILIPSEN, 2015]

In den vergangenen Jahren ist der Bedarf an Datenvisualisierungen extrem angestiegen. Grund hierfür sind unter Anderem die stetig wachsende Menge an täglich produzierten Daten und der Bedarf nach deren Auswertung. Durch das Internet und das meist transparente und oftmals leichtsinnige Verhalten seiner Nutzer sind Daten heute wertvoller und umstrittener als jemals zuvor. Unzählige neue Firmen, besonders aus dem Marketing- und Forschungsbereich, haben das Potential der regionalen und weltweiten Datengewinnung erkannt und auf mittlerweile fast alle Bereiche in unserem Leben angewandt - „Big Data“[1] sei an dieser Stelle als eine bekannte und stellvertretende Begrifflichkeit genannt.
Die unterschiedlichen Bereiche, in denen Daten und deren Visualisierung/Auswertung ein wichtiges Thema sind, nehmen stetig zu: bei Polizei, Feuerwehr, Stadt, Medizin, Wissenschaft, Tourismus, Banken und Versicherungen, Dienstleistungen und IT, Handel und Marketing finden sie zunehmend Anwendung. Aufgaben können hierbei beispielsweise sein, dass Lagerbestände noch besser in die Zukunft geplant, Kriminalität präventiv verhindert, Kundenverhalten vorausgesagt und Versicherungen individueller gestaltet werden können. [PHILIPSEN, 2015].

[1] „Big Data“ bezeichnet Datensätze, deren Größe die Kapazität und Fähigkeit klassischer Datenbank-Softwaretools zur Erfassung, Speicherung, Verarbeitung und Analyse übersteigt. Diese Beschreibung ist subjektiv und beinhaltet, da es hier keine genauen Messgrößen gibt, keine genaue Angabe dazu, wie groß ein Datensazt sein muss, um als Big Data zu gelten. Die Frage, welches Datenvolumen als Big Data gilt, kann je nach Branche, Software sowie den in einem bestimmten Industriezweig verfügbaren oder verwendeten Tools variieren. [PWC, 2013]

4.1 Definition

Die Recherche nach einer allgemeingültigen und weit verbreiteten Definition hat sich als schwierig erwiesen. „Datenvisualisierung“ bekommt in der Literatur verschiedene Bedeutungshüllen verliehen, was dem Wunsch nach einer eindeutigen Erklärung gegenübersteht. Dies ist vermutlich dadurch zu erklären, dass a) das Forschungsfeld in diesem Bereich noch relativ jung ist und b) eine Vielzahl an Begrifflichkeiten in diesem Mikrokosmos vorhanden ist, die entweder gar nicht oder auf unterschiedliche Arten und Weisen voneinander abgegrenzt werden. Stellvertretend sei hier die Differenzierung zwischen „Datenvisualisierung“ und „Informationsvisualisierung“ genannt - etymologisch wäre hier eine Abgrenzung logisch und einfach zu treffen[2], in der Fachliteratur werden beide Begriffe jedoch teils gegensätzlich, aber auch teils synonym verwendet.
Juan Carlos Morales Posada hat im Jahre 2015 im Rahmen seiner Masterarbeit in Kommunikationsdesign an der Universidad de Buenos Aires einen Fachartikel zu dieser Problematik der Begrifflichkeiten erstellt, auf den an dieser Stelle verwiesen wird. [MORALES POSADA, 2015]

Im Rahmen dieser Diplomarbeit soll der Begriff „Datenvisualisierung“ so verstanden werden, dass er für die grafische Aufbereitung von Datensätzen steht - in diesem Fall für Ergebnisdatensätze aus Mikrosimulationsdurchläufen. Ziel einer Datenvisualisierung, in diesem Sinne, soll es daher sein, dass Daten, die ausschließlich in Textform innerhalb einer Datenbank abgespeichert sind, so dargestellt werden, dass der Nutzer durch diese andere Form der Ergebnispräsentation in die Lage versetzt wird, die Daten besser zu verstehen und analysieren zu können.

4.2 Historie

Das Forschungsgebiet „Datenvisualisierung“ ist im Vergleich zu anderen noch relativ jung. Bis 1976 führte dieser Bereich noch ein Schattendasein und wurde anfänglich beispielsweise für meteorologische Karten für Seefahrer genutzt oder aber auch im

[2] „informatio“ aus dem Lateinischen bedeutet „Form geben“, „datum“, als Vergangenheitsform von „dare“, kann der Ursprung des Wortes „Daten“ sein und somit für „das Gegebene“ stehen. Daher könnten Informationen als Produkt einer Formgebung einer bestimmten Menge an Daten angesehen und die Begriffe hierdurch entsprechend zueinander in Verbindung gesetzt werden.

mathematischen Bereich in Form von Graphen und Diagrammen um zum Beispiel Funktionsbeschreibungen grafisch darzustellen. Von Ende der Siebziger bis Ende der Achtzigerjahre entwickelten sich zunehmend erste spezialisierte Visualisierungsanwendungen. Grund für das Umdenken und die Neuentwicklungen waren unter Anderem der Bedarf daran, große unübersichtliche Datenmengen erfassen und analysieren zu können. Mit die ersten Gebiete in denen derartige Visualisierungen intensiv vorangetrieben wurden, waren Raumfahrt und Medizin. Nicht nur die zunehmende Größe der Datenmengen waren hierfür ausschlaggebend, sondern auch das vermehrt auftretende Bedürfnis nach Analyse und dem Verständnis der komplexen Zusammenhänge ebendieser. Ab ca. 1987 wurde „Datenvisualisierung“ zu einem eigenen Forschungsgebiet. [DEUSSEN-2003]

„Ende der Achtzigerjahre wurden computerbasierte Techniken intensiv bei der Bearbeitung und der Auswertung von umfangreichen Datenreihen im Rahmen der wissenschaftlichen Studie zur Erforschung der Konzentration von Ozonwerten in der Atmosphäre eingesetzt. Dieses führte zur Entstehung der Spezialdisziplin Scientific Visualization. Anfang der 1990er Jahre wurder der Begriff Information Visualization entwickelt.“ [KRYPCZYK, 2014]

Besonders in den vergangenen zehn Jahren hat im Bereich der Datengenerierung, und somit auch analog in dem der Datenvisualisierung, eine sprunghafte Entwicklung stattgefunden. Daniel A. Keim schreibt in seiner Arbeit, datiert auf 2004, dass Forscher an der Universität Berkeley berechnet haben, dass jedes Jahr 1 Exabyte[3] Daten erzeugt werden. Dies bedeutet, dass in den drei Folgejahren mehr Daten generiert wurden als in der gesamten menschlichen Entwicklung zuvor. [KEIM, 2004] Elf Jahre später, sprich 2015, legt das Magazin „Wirtschaftswoche“ aktuellere Zahlen vor: laut ihrem Artikel werden durch die drei Milliarden Web-Nutzer sowie die vielen Milliarden via Internet vernetzten Geräte inzwischen 2.5 Trillionen Byte Daten produziert - pro Tag. 90 Prozent dieser Daten sind unstrukturiert, da es sich bei ihnen um Fotos, Tweets, Log-Files etc. handelt. [WiWo, 2015] Hieran wird sehr gut ersichtlich, wie wichtig das Thema Datenvisualisierung in allen Bereichen geworden ist und auch weiter werden wird. Ohne entsprechende visuelle Strukturierungen und Darstellungen wird es zukünftig fast nicht mehr möglich sein, die Flut an Daten sinnvoll und in angemessener zeit zu bearbeiten.

[3] dies entspricht 1 Million Terabyte

4.3 Darstellung von Daten

Für die Visualisierung von Daten gibt es unzählige Möglichkeiten und Techniken sowie unzählig viele Anwendungsbereiche. Um für die eigenen Bedürfnisse die passende Darstellungsform zu finden ist es wichtig im Vorfeld zu verstehen, was das Ziel bzw. der Mehrwert der Visualisierungen sein soll.

Möchte man beispielsweise aus einem Datensatz von Bankfilialen, diejenigen herausfinden, die den meisten Umsatz generieren, ist es nicht sinnvoll, per Hand eine entsprechende Tabelle oder sogar eine Datenbank mit mehreren verknüpften Tabellen zu durchsuchen. Ein solches Vorgehen wäre zeitlich nicht tragbar. In diesem Fall ist es sinnvoll (wenn vorhanden) ein Skript über den Datensatz laufen zu lassen, welches eine entsprechende mathematisch-logische Filterung ermöglicht, oder eine visuelle Darstellung, die es ermöglicht, die entsprechend gewünschten Filialen zu ermitteln.

In diesem konkreten Beispiel würde tatsächlich die Ausgabe einer Liste/Tabelle ausreichen, da sie bereits das gewünschte Ergebnis darstellt und nicht weiter interpretiert werden muss. Würde man die Anfrage nun aber so erweitern, dass zu dem Umsatz noch eine geografische Komponente, sprich der Wohnort des Kunden, hinzukommen soll und beispielsweise auch noch eine Altersverteilung, dann würde diese zunehmende Komplexität gegen eine rein tabellarische oder textuelle Darstellung der Ergebnisse sprechen. An dieser Stelle wäre es dann spätestens sinnvoll auf eine grafisch-gestützte Ergebnisauswertung zu setzen um den zeitlichen Arbeitsaufwand herab und die Interpretierbarkeit der Daten nach oben zu setzen.

Generell sind bei der Visualisierung von Daten zwei Bestandteile ausschlaggebend für die jeweils entsprechende Umsetzung: der vorhandene Datentyp und die verwendete Visualisierungstechnik. Beide Komponenten werden in den zwei folgenden Tabellen übersichtlich aufgelistet:

Tabelle 4.1: Übersicht Datentypen

Art des Datentyp	mögliches Anwendungsgebiet
Ein-dimensional	zeitabhängige Daten
Zwei-dimensional	geographische Karten
Multi-dimensional	tabellarische Daten aus relationalen Datenbanken
Text und Hypertext	Nachrichten oder Web-Dokumente
Hierarchien und Graphen	Telefon- oder Internetverbindungen
Algorithmen und Software	Debugging-Operationen

Quelle: [SHNEIDERMAN, 1996]

Eindimensionale Datentypen besitzen meist ein kontinuierliches Attribut, welches eine vollständige Ordnung auf den Daten definiert. Bei zeitabhängigen Daten können hierbei beispielsweise jedem Zeitpunkt mehrere Daten zugeordnet werden.

Zweidimensionale Daten definieren einen eindeutigen Punkt über zwei spezielle Dimensionen. Für die Darstellung von zweidimensionalen Datentypen eignen sich zum Beispiel besonders gut X-Y-Plots, wie sie bei der Visualisierung von geographischen Daten zum Einsatz kommen.

Multidimensionale Daten werden durch mindestens drei Attribute beschrieben, weshalb sie nicht mehr mit den Standard 2D/3D-Darstellungen visualisiert werden können. Dies ist der Fall, wenn man versucht die Daten einer relationalen Datenbank grafisch darzustellen, welche hunderte oder tausende unterschiedliche Attribute besitzen kann. Für solche Fälle reichen die Standardvisualisierungsmöglichkeiten nicht

aus - im konkreten Fall einer sehr großen relationalen Datenbank kann die „parallele Koordinaten-Technik“, welche später erläutert wird, ein gutes Werkzeug zur grafischen Auswertung der Daten sein.

Text und Hypertext sind gleichermaßen Grundlage und ausschlaggebender Entwicklungsgrund für das heutige Internet. Aufgrund ihrer Komplexität und unzähligen Erscheinungsformen ist es schwierig sie in einen konkreten Datentypen zu packen, was auch eine anschließende Visualisierung selbiger erschwert. Damit sie aber dennoch visualisiert werden können, werden sie oft in sogenannte Merkmalsvektoren[4] überführt. Eine sehr simple Transformation dieser Art wäre es beispielsweise, in einem Text alle nicht-trivialen Wörter zu zählen.

Hierarchien und Graphen werden eingesetzt, wenn die Datensätze untereinander komplexe Beziehungen beinhalten, welche mit anderen Verfahren nicht oder nur sehr unübersichtlich abgebildet werden können. Ein Graph setzt sich zusammen aus Objekten, Knoten und Kanten, welche die Verbindungen zwischen den einzelnen Knoten darstellen.
Um in einem solchen Konstrukt zusätzlich eine Hierarchie zu visualisieren, kann man zusätzlich festlegen, dass die Art der Kanten nur in Top-Down-Richtung verlaufen darf. Typische Anwendungsgebiete hierfür sind die Verbindungen in Telefon- oder Computernetzwerken, sowie das Filesystem einer Festplatte.

Auch für **Algorithmen und Software** ist es sinnvoll mit geeigneten visuellen Darstellungen zu arbeiten, die Struktur, Ablauf und Zusammenhänge einfacher erkennbar machen, als es ein Blick in den Quellcode oder bestimmte Klassen kann. Ein sehr bekanntes Werkzeug hierfür ist das UML-Diagramm[5]. [SHNEIDERMAN, 1996]

[4] ein Merkmalsvektor fasst die (numerisch) parametrisierbaren Eigenschaften eines Musters in vektorieller Weise zusammen. Verschiedene, für das Muster charakteristische Merkmale bilden die verschiedenen Dimensionen dieses Vektors. Die Gesamtheit der möglichen Merkmalsvektoren nennt man den Merkmalsraum

[5] ein UML- oder auch Klassendiagramm ist ein Strukturdiagramm der „Unified Modeling Language“ (UML) zur grafischen Darstellung von Klassen, Schnittstellen sowie deren Beziehungen

Tabelle 4.2: Übersicht Visualisierungstechniken

Visualisierungstechnik	möglicher Anwendungsfall
Standard 2D/3D	Balkendiagramm, XY-Diagramm
Geometrische Transformationen	künstliche Landschaften, parallele Koordinaten
Icon-basierte Visualisierungen	Zeitschriften, Webseiten, Infografiken
Pixel-Visualisierungen	Recursive Pattern[1]

[1] sich wiederholende Muster, die im Bereich der Datenvisualisierung bei sehr großen Datensätzen verwendet werden können [ANKERST et al., 1995]

Quelle: [SHNEIDERMAN, 1996]

Heutzutage existiert eine Vielzahl von Visualisierungstechniken. Neben den weitverbreiteten **Standard-2D/3D-Techniken**, wie Balken- oder Liniendiagramme, wurden weitere Techniken entwickelt:

Geometrische Transformationen haben zum Ziel Projektionen aus multidimensionalen Datenmengen zu erstellen. Die hierbei genutzten Techniken werden unter dem Oberbegriff „Projection Pursuit“[6] zusammengefasst.

Werden Attribute eines Datensatzes mittels visueller Darstellung auf die Eigenschaf-

[6] Projection Pursuit (wörtlich Nachverfolgung der Projektion) ist ein statistisches Verfahren, eine Menge hochdimensionaler Daten so zu vereinfachen, dass möglichst „interessante“ Strukturen darin aufgedeckt werden. Dazu wird eine Hyperebene (z. B. eine Fläche) in den durch die Daten aufgespannten Raum gelegt, auf welche die Daten projiziert werden.

ten eines Icons abgebildet, hat man eine **Icon-basierte Visualisierung** erzeugt, die oft zur Vereinfachung herangezogen und heutzutage auch immer zunehmender in Infografiken verwendet zum Einsatz kommt, da die oftmals vereinfachte und komprimierte Dartellung der Daten in puncto Design viel Spielraum gewährleistet.

Bei **Pixelvisualisierungen** wird jeder Datenwert auf ein farbiges Pixel abgebildet. Entsprechend der Dimensionen werden die Pixel gruppiert (bei der Recursive Pattern Technik zum Beispiel in rechteckigen Teilbereichen) und dementsprechend über die Teilbereiche verstreut. Über ihre relative Position innerhalb dieser Teilbereiche stehen sie nun in Beziehung zueinander, was es ermöglicht lokale Beziehungen zwischen den Attributen, Korrelationen Ausnahmen zu finden. [SHNEIDERMAN, 1996]

4.4 Webbasierte Lösungen

In Vorbereitung auf Kapitel 5 und die programmiertechnische Umsetzung einer grafischen Ergebnisdatensatzausgabe wird sich dieser Teil der Arbeit mit vorhandenen, webbasierten Lösungen zur Ausgabe von Diagrammen beschäftigen. „CoMicS“, in der Variante, wie es von Christian Tiefenau programmiert wurde, ist eine Server-/Client-Softwarelösung, welche mit Hilfe eines Web-Browsers arbeitet. [TIEFENAU, 2014] Diese vorgegebene Struktur soll im Rahmen dieser Diplomarbeit erhalten und weiter ausgebaut werden. Daher ist die sinnvollste Lösung für die grafische Ergebnisverwertung, dass diese ebenfalls innerhalb eines Browsers und somit client-seitig möglich ist. In den vergangenen Jahren wurde eine Vielzahl an Javascript-Bibliotheken entwickelt, welche die Umwandlung von Daten in Diagramme ermöglichen. Eine detaillierte Analyse von einzelnen Anwendungen würde den Rahmen dieser Diplomarbeit sprengen, weshalb an dieser Stelle stellvertretend einige der populärsten Lösungen aufgelistet werden:

Tabelle 4.3: Javascript-Bibliotheken zur Diagrammerstellung

Name	Webseite
flotr2	http://www.humblesoftware.com/flotr2
highcharts	http://www.highcharts.com/
AMCharts	https://www.amcharts.com/
D3.js (data driven documents)	https://d3js.org/
Google Charts	https://developers.google.com/chart/
jqplot	http://www.jqplot.com

Quelle: eigene Recherche

Alle in der vorherigen Tabelle aufgelisteten Javascript-Bibliotheken haben gemeinsam, dass mit ihnen die Generierung von Diagrammen aus vorgegebenen Daten möglich ist. Wichtig bei der Auswahl war zudem das Kriterium, dass die entsprechenden Bibliotheken frei nutzbar sind. Dies hat zum Einen den Vorteil, dass es in verschiedenen Projekten genutzt werden kann und zum Anderen, dass in den meisten Fällen eine opensource-Community an Stabilität und Weiterentwicklung der Bibliotheken arbeitet. Weitere Kriterien bei der Analyse waren zudem, ob und in welcher Form eine Weiterverarbeitung der Diagramme möglich ist und ob das Design an eine eventuelle bereits vorhandene CI[7] des eigenen Projektes angepasst werden kann.
Um eine entsprechende Erweiterung in „CoMicS“ zu implementieren, scheint „jqplot“ die beste Variante zu sein. Es erfüllt alle zuvor genannten Kriterien und ist die einzige Bibliothek die die Implementierung von Diagrammexporten (als Grafik) ohne größere Umwege anbietet. Weiterhin bietet die Webseite von jqplot eine sehr umfangreiche

[7] „Corporate Identity“ ist die Gesamtheit der Merkmale, die ein Unternehmen kennzeichnen und es von anderen Unternehmen unterscheiden.

Dokumentation der eigenen API an. Zusätzlich kann der Benutzer selbst auswählen, ob er den Code im Rahmen der „GPL License“ [GNU General Public License Version 3, 2016] oder der „MIT License“[8] [MIT LICENSE, 2016] nutzt, wodurch es im freisteht, ob der Code in einem privaten oder kommerziellen Rahmen verwendet wird.

4.5 Relevanz im Bereich der Mikrosimulation

Die Relevanz des Themas Datenvisualisierung im Bereich der Mikrosimulation ist nicht nur ganz klar gegeben, sondern wird, wie der Eindruck nach der Recherche für Kapitel 3 erweckt wurde, oftmals vernachlässigt und scheint bei vielen Simulationsprogrammen schlichtweg nicht behandelt worden zu sein. Dies zeigt die vorhandene Vielzahl an Programmen, bei denen die einzige Möglichkeit der Ergebnisausgabe in Text- oder Tabellenform gegeben ist. Wie bereits zuvor erwähnt, kann dies bei kleineren, übersichtlichen Aufgaben auch genügen, generell ist dies meiner Meinung nach jedoch der verkehrte Ansatz. Wichtig hierbei ist nicht nur der zeitliche Faktor, der bei größeren Datensätzen eine immer wichtigere Rolle spielt, sondern auch die generelle Möglichkeit zur Analyse der Daten. Simulationen, die mit Ausgangsdatensätzen von 100.000 Haushalten, mehreren Agenten und Events starten, generieren einen Ergebnisdatensatz, der weder textuell noch tabellarisch sinnvoll durch den Nutzer erfasst werden kann. Zusätzliche Software wie „Microsoft Excel“ bietet zwar Möglichkeiten der Strukturierung und/oder Vereinfachung der Daten, aber über diesen Weg kommt der Nutzer bei größeren Datensätzen schnell an seine Grenzen. Bei Mikrosimulationsprogrammen, die mit Datenbanken arbeiten, wird es für den Nutzer der Software sogar zwingend notwendig sein, dass er mindestens über Grundkenntnisse in beispielsweise SQL[9] verfügt, um mit entsprechend programmierten Abfragen die Daten gefiltert und sortiert auslesen zu können. Solche Voraussetzungen bzw. Hürden erschweren das Arbeiten mit entsprechender Software und schränken den möglichen Nutzerkreis extrem ein. Es ist sinnvoll, dass auch Nutzer die keine Programmieraffinität aufweisen in die Lage versetzt werden, mit dem jeweiligen Programm zielführend arbeiten zu können.

[8] die MIT-Lizenz vom Massachusetts Institute of Technology erlaubt die Wiederverwendung der unter ihr stehenden Software sowohl für Software, deren Quelltext frei einsehbar ist, als auch für Software, deren Quelltext nicht frei einsehbar ist.

[9] SQL ist eine Datenbanksprache zur Definition von Datenstrukturen in relationalen Datenbanken sowie zum Bearbeiten (Einfügen, Verändern, Löschen) und Abfragen von darauf basierenden Datenbeständen.

Eines der Hauptziele der Datenvisualisierung ist die anschließende Exploration der generierten Daten. Gerade bei Mikrosimulationen können grafische Ergebnisausgaben bei diesem Vorhaben helfen und beispielsweise auch im Sinne einer Plausibilitätsprüfung genutzt werden um zu sehen, ob die errechneten Daten mit der aufgestellten Hypothese übereinstimmen bzw. ob die programmierte Simulation überhaupt korrekt durchgelaufen ist. Zukünftig könnten solche Prüfungen (was nicht Gegenstand dieser Diplomarbeit ist) sogar vollautomatisiert ablaufen und beispielsweise schon während der Simulation mittels Schwellwerten prüfen, ob Ergebnisdaten plausibel berechnet werden oder eben nicht.
Ein weiterer interessanter Punkt bei der grafischen Ergebnisauswertungen ist die Exportmöglichkeit von aus den Daten generierten Diagrammen. Wenn der Nutzer die Möglichkeit hat solche Diagramme beispielsweise als .jpg-Bilder zu exportieren, wird er damit in die Lage versetzt, unabhängig von der Software diese grafischen Ergebnisse weiterzuverwenden. Dies kann für viele Fälle interessant sein, wie zum Beispiel bei der Arbeit in dezentralisierten Teams oder aber auch, wenn erstellte Grafiken in Handbüchern oder wissenschaftlichen Berichten/Artikeln weiterverwendet werden sollen.

Insgesamt ist folglich festzustellen, dass der Datenvisualisierung im Bereich der Mikrosimulation ein hoher bzw. höherer Stellenwert zugeordnet werden sollte, da mit ihrer Hilfe eine bessere und schnellere Analyse/Auswertung von vorhandenen Ergebnissen erzielt werden kann, die über die Arbeit mit beispielsweise rein tabellarischen Ergebnisausgaben hinausgeht. Selbstverständlich ist von Fall zu Fall zu unterscheiden, inwieweit ein tatsächlicher Mehrwert für den Nutzer dadurch entsteht. Aber die allgemeine Entwicklung in puncto Datenverarbeitung und -repräsentation der vergangenen Jahre zeigt deutlich, dass hier ein wichtiges aktuelles und auch zukünftiges Forschungsgebiet liegt.

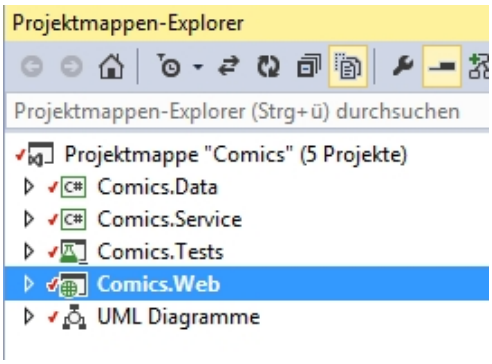
5 Grafische Ergebnisausgabe in CoMicS

Dieser Teil der Arbeit befasst sich mit der Mikrosimulationssoftware „CoMicS“, welche von Christian Tiefenau im Rahmen seiner Masther Thesis an der Hochschule Bonn-Rhein-Sieg 2014 neu konzipiert und programmiert wurde. [TIEFENAU, 2014] Insbesondere wird an dieser Stelle der Fokus auf die grafische Ergebnisausgabe gelegt, da dieser Punkt anschließend erweitert bzw. zusätzlich implementiert werden soll. Für Erweiterungen einer bestehenden Software muss deren Konstrukt erst einmal verstanden werden - diese kurze Betrachtung bildet den ersten Teil von diesem Kapitel. Anschließend werden einzelne Ziele definiert, welche insgesamt die grafische Ergebnisdarstellung in „CoMicS“ realisieren.

5.1 Aktueller Stand

Im Gegensatz zu den analysierten Programmen arbeitet „CoMicS“ Client-/Serverbasiert. Hierdurch kann das eigentliche Interface der Software in einem Web-Browser genutzt werden und ist somit plattformunabhängig. Die Simulationsdaten werden in einer MySQL-Datenbank abgelegt. Das für diese Arbeit zur Verfügung stehende „Microsoft Visual Studio“-Projekt untergliedert sich in die vier Unterprojekte, die in Abbildung 5.1 zu sehen sind.

Abbildung 5.1: CoMicS-Projektmappe



Quelle: eigener Screenshot

Besonders interessant für das weitere Vorgehen sind hierbei „Comics.Data“ und „Comics.Web“. Erstes beinhaltet die Schnittstellen und Definitionen der Simulation und Letzteres beinhaltet alles, was für den Frontendbereich, sprich die Ausgabe/Ansicht im Browser relevant ist. Das Backend dieser Software ist in C# geschrieben, die Erstellung des User-Interfaces erfolgte mittels HTML, CSS und verschiedenen Javascript-Bibliotheken, wie beispielsweise „angular.js“.[1]
Back- und Frontend sind nach dem MVC[2]-Prinzip aufgebaut, was bei den Anpassungen und Erweiterungen berücksichtigt werden muss. Desweiteren wurde das „Entity Framework 6.1.1“[3] genutzt, welche eine abstrahierte Ebene zwischen Datenbank und Programm darstellt. Hierdurch werden konkrete SQL-Befehle zum Abfragen von Ergebniswerten aus der Datenbank hinfällig, da die Einträge der Datenbank in Form von Objekten zur Verfügung gestellt und somit direkt im Code genutzt werden können.
Für die Darstellungen im Frontend besteht bereits eine grafische CI, welche mittels verschiedener .css-Dateien im Verzeichnis „Comics.Web/Content“ gesteuert und im Rahmen der Erweiterungen ergänzt und gegebenenfalls angepasst werden.
Aktuell bietet „CoMicS“ keine grafische Repräsentation der Simulationsdaten. Wie dieser Bereich erweitert werden soll, wird im folgenden Unterkapitel 5.2 erläutert.

[1] mittels Angular kann das Vokabular von HTML um selbstdefinierte, clientseitig gerenderte Elemente erweitert werden.
[2] das Model-View-Controller-Prinzip ist ein Strukturierungsmuster bei der Softwareentwicklung
[3] kostenloses, zusätzliches Package für Microsoft Visual Studio

5.2 Ziel und Vorgehensweise

Ziel der Implementierung soll es sein, eine grafische Auswertung der Simulationsergebnisse zu erstellen. Der Fokus hierbei liegt auf allgemeinen Informationen und den Events der jeweiligen Simulation und den daraus ableitbaren Erkenntnissen für den Benutzer. Simulationen mit „CoMicS“ müssen beispielsweise zwingend ein sogenanntes Init-Event haben, welches jeder Agent zu Beginn der Simulation durchläuft und somit dessen Initialisierung darstellt. Hier kann direkt geprüft werden, ob die Simulation bezüglich aller initialen Agenten korrekt gestartet ist.
Generell soll die Auswertung so konzipiert werden, dass sie dynamisch und modular angelegt ist. Das bedeutet, dass die Anzahl der Events „egal“ ist und die Auswertung für 1 - X Events funktioniert. Dementsprechend muss das Einlesen der Simulationsergebnisdaten abstrakt gehalten werden. Hierzu wird ein Controller erstellt, der über die Klassen und Definitionen in „Comics.Data“ bei einer übergebenen Simulations-ID die entsprechenden Daten abfragen und an den View („SimulationAnalysis/Details.cshtml“) weitergeben kann.
Um diese Implementierung zu realisieren wird das Vorhaben in den folgenden Unterkapiteln in separate Ziele zerlegt, einzeln definiert und im Kapitel 5.3 deren technische Umsetzung aufgezeigt.

5.2.1 Ziel: Anzeige allgemeiner Simulationsinformationen

Zu Beginn der grafischen Ergebnisdarstellung sollen allgemeine Simulationsinformationen angezeigt werden. Diese sind:

- die Simulations-ID
- der Zeitraum der Simulation
- die Anzahl der Agenten
- Anzahl der durchlaufenen Events
- Auflistung der einzelnen Events

5.2.2 Ziel: Prozentuale Verteilung der Events

Dieser Teil der Darstellung soll die prozentuale Verteilung der einzelnen Events abbilden und somit anzeigen, welchen Anteil ein Event an den Durchläufen aller Events hat. Für prozentuale Auswertungen eignet sich am besten ein Kreisdiagramm. Die einzelnen Kreissegmente sollen mit der jeweiligen Prozentzahl gekennzeichnet sein und in unterschiedlichen Farben visualisiert werden. Damit der Nutzer die Segmente dem richtigen Event zuordnen kann, soll eine entsprechende Legende ausgegeben werden, die eine Übersicht der im Kreisdiagramm vorhandenen Events darstellt.

5.2.3 Ziel: Balkendiagramm pro Event

Für jedes Event, das in der Simulation vorhanden ist, soll ein Balkendiagramm erstellt werden, welches anzeigt wie oft das jeweilige Event in jedem Jahr der Simulation durchlaufen wurde. Hier kann der Nutzer (abhängig vom Event und dessen Inhalt) beispielsweise anhand von extrem ausschlagenden Werten oder Differenzen zwischen den Jahren mögliche Fehler in der Simulation erkennen und hat somit eine Art Plausibilitätsprüfung. Wie aussagekräftig solche Werte letztendlich sind, hängt aber vom jeweiligen Event und dessen Funktionsweise innerhalb der Simulation ab. Um eine weitere Verarbeitung oder beispielsweise auch Versendung dieser Diagramme zu ermöglichen, sollen sie zum Einen als Bild abgespeichert werden können oder in eine .csv-Datei exportiert werden, welche eine zweispaltige Darstellung des Diagramms beinhaltet.

5.2.4 Ziel: Verlinkung innerhalb der Software

Die errechneten Diagramme und Simulationsinformationen werden alle innerhalb einer Ergebnisseite ausgegeben. Diese Seite muss innerhalb des Programms erreichbar sein und in die bestehende Seitenstruktur eingebaut werden. Die Seite „Simulationsübersicht“ listet alle bisher mit der Software erstellen Simulationen auf und zeigt die Werte der Parameter Simulations-ID, Name, Start-/Enddatum, Simulationsläufe, Agentenzahl und Ereignisse in einer Tabelle an. Jede Tabellenzeile beinhaltet eine Simulation und ist gleichzeitig ein weiterführender Link auf die „Details“-Seite. Dort erhält der Nutzer die Werte erneut in einem Interface, welches es ihm ermöglicht die Parameter zu manipulieren und die Simulation erneut durchlaufen zu lassen. An dieser Stelle soll ein weiterer Button eingefügt werden, der auf die Seite mit den grafischen

Ergebnisausgaben verlinkt.

5.3 Technische Umsetzung der Zieldefinitionen

Für die Implementierung der in Kapitel 5.2 definierten Ziele muss die bestehende Struktur der Software erweitert werden. Dies kann unterteilt werden in Controller- und View-Ebene. Es wird ein neuer Controller benötigt, der bei übergebener Simulations-ID die entsprechenden Werte aus der Datenbank in Objektform abstrahiert und an den View, der die Darstellung der Daten regelt, übergibt. Dieser Controller wird in einem eigenen Verzeichnis zu den bestehenden Controllern hinzugefügt. Der neue View wird ebenfalls in ein eigenes Verzeichnis ausgelagert, damit auch hier eine kontextuelle Abgrenzung gegeben ist.

5.3.1 Der SimulationData Controller

Der SimulationDataController wird in Form einer .cs Datei abgelegt unter: „Comics.Web/Controller/api/SimulationDataController.cs“. Die wichtigste Klasse „SimulationDataController“ erbt von der „ApiController“-Klasse, wodurch ihr beispielsweise die standardmäßigen get-/set-Methoden zur Verfügung stehen. Den Kopf der Datei bilden die sogenannten using-Direktiven:

Listing 5.1: using-Direktiven in SimulationDataController.cs

```
1 using Comics.Data;
2 using Comics.Data.Models;
3 using Newtonsoft.Json;
4 using System;
5 using System.Collections.Generic;
6 using System.Linq;
7 using System.Net;
8 using System.Net.Http;
9 using System.Web.Http;
```

Neben den CoMicS-eigenen Paketen werden weitere eingebunden, wie beispielsweise „System.Web.Http“, in dem sich die Definitionen für den ApiController befinden.
Die Get()-Methode erwartet einen Integerwert, der die ID der Simulation beschreibt, welche vom View mittels http.get(...); übergeben wird -dieser Vorgang wird Unter-

kapitel 5.3.2 genauer beschrieben. Mittels dieser ID wird im Comics-Context auf die entsprechenden Werte der Datenbank abstrahiert zugegriffen. Abstrahiert bedeutet an dieser Stelle, dass keine konkreten SQL-Befehle implementiert werden müssen, sondern auf die definierten Variablen und Methoden in „Comics.Data/oimicsContext.cs“ zugegriffen werden kann. Im folgenden Schritt werden die Daten der entsprechenden Simulation verarbeitet, damit sie anschließend in Form eines JSON-Objects an den View zurückgegeben werden können. Hierzu definiert die Klasse „SimulationRunTransfer“ die benötigten Variablen und die Methode, die einen Simulationslauf als Parameter erwartet. Der folgende Code-Ausschnitt zeigt die gerade erwähnten Punkte:

Listing 5.2: Ausschnitt aus der Klasse SimulationDataController

```
1  public class SimulationDataController : ApiController
2      {
3          public string Get(int id)
4          {
5              var context = new ComicsContext();
6              var run = context.SimulationRuns.AsNoTracking().
                   FirstOrDefault(sr => sr.SimulationRunId == id)
                   ;
7
8              var simulationData = new SimulationData() { Run
                   = new SimulationRunTransfer(run)};
9  //....
10 public class SimulationRunTransfer
11     {
12         private SimulationRun run;
13
14         public SimulationRunTransfer(SimulationRun run)
15         {
16             SimID = run.SimulationRunId;
17             StartYear = run.Simulation.StartDate.Year;
18             EndYear = run.Simulation.EndDate.Year;
19             Agents = run.SimulationAgents.Count;
20             Progress = run.Progress;
```

```
21              Status = run.Status;
22              Message = run.Message;
23          }
24
25          public int StartYear { get; set; }
26          public int EndYear { get; set; }
27          public int Agents { get; set; }
28          public double Progress { get; set; }
29          public EStatus Status { get; set; }
30          public string Message { get; set; }
31          public long SimID { get; set; }
32      }
33  }
```

Die Codezeilen 16 bis 22 zeigen, welche Informationen der Simulation ausgelesen und für die Übergabe an den View gespeichert werden.
In Zeile 8 dieses Listings ist zu sehen, dass ein neues Objekt der Klasse „Simulation-Data“ erzeugt wird - diese Klasse definiert get-/set-Methoden für eine Instanz von „SimulationRunTransfer“:

Listing 5.3: die Klasse SimulationData

```
1  public class SimulationData
2      {
3          public SimulationRunTransfer Run { get; set; }
4      }
```

5.3.2 SimulationAnalysis View

View-Dateien in diesem Projekt werden als .cshtml-Dateien erzeugt, was bedeutet, dass eine typische HTML-Seite generiert wird, innerhalb derer es aber auch theoretisch möglich ist C#-Code zu integrieren. Es wird sich im Laufe der Programmierarbeiten zeigen, ob dies benötigt wird oder nicht.
Der unter „Comics.Web/Views/SimulationAnalysis/Details.cshtml“ erzeugte View wird später die grafische Ergebnisdarstellung enthalten und im Web-Browser über index/-SimulationAnalysis/Details/[SimulationID] erreichbar sein. Wie genau der View in

die bestehende Linkstruktur von „CoMicS“ eingebaut wird, kann Kapitel 5.3.6 auf Seite 81 entnommen werden.
Die zuvor definierten Ziele resultieren in der Grundstruktur des Views, welche folgendermaßen aussehen wird:

Listing 5.4: HTML-Struktur des View

```
<div id="generalSimulationInfo">
        <table id="generalInfoTable">
            <!-- Tabelle für allgemeine Simulations-
                Informationen-->
        </table>
    </div>
    <h3>Events</h3>
    <div>
        <h5>Prozentuale Verteilung der Events:</h5>
        <div id="events-pie">
            <!-- Kreisdiagramm zeigt die prozentuale
                Verteilung der Events an -->
        </div>
    </div>
    <div>
        <h5>Häufigkeit der einzelnen Events pro Jahr:</h5>
    </div>
    <div id="chartContainer">
        <!-- hier werden nacheinander alle Events der
            Simulation als Balkendiagramme visualisiert -->
    </div>
</div>
<script type="text/javascript">
        /*
                - get()-Zugriff auf
                    SimulationAnalysisController
                - Einlesen und Verarbeiten der
                    Simulationsdaten
                - Erstellen der Diagramme
        */
```

```
26 </script>
```

Das Grundlayout aller in „CoMicS“ enthaltenen Seiten wird über eine allgemeine Layoutdatei gesteuert, welche unter Comics.Web/Views/Shared/_Layout.cshtml zu finden ist. Diese baut das HTML/CSS-Grundgerüst auf und rendert die in „Comics.Web/App_Start/BundleConfig.cs“ definierten Style- und Skriptpakete. Für die spätere Verwendung von „jqPlot“ wurde in dieser Datei ein neues Bundle hinzugefügt:

Listing 5.5: Erweiterung der BundleConfig.cs

```
1 bundles.Add(new ScriptBundle("~/bundles/jqplot").Include(
2                                         "~/Scripts/jquery.jqplot
                                             .js",
3                                         "~/Scripts/jqplot/*.js"
4                                         ));
```

Die erste dort verlinkte Datei beinhaltet die generellen jqPlot-Funktionen und Bibliotheken. Sie muss initiierend eingebunden werden, damit die folgenden Skripte in „Scripts/jqplot/“ funktionieren. Im ebendiesem Verzeichnis befinden sich die unterschiedlichen Renderer und Plugins von jqPlot. Ebenfalls, nach dem gleichen Prinzip, werden über die BundleConfig.cs alle weiteren CSS-/Javascript- und jQuery-Dateien eingebunden die für das Projekt relevant sind.

Um die Simulationsergebnisdaten aus der Datenbank zu bekommen, wird wie bereits vorher erwähnt der SimulationAnalysisController via http-get angesprochen. Hierzu wird die ID der aktuellen Simulation übergeben und von der get()-Methode verarbeitet. Der Controller liefert ein Objekt zurück, welches die Daten beinhaltet und vom View in ein besser zu verarbeitendes JSON-Objekt geparsed wird. Diese Abfrage ist Teil der <script>Section.

Listing 5.6: Datenabfrage an den SimulationAnalysisController

```
1 $http.get("/api/SimulationData/" + window.location.pathname.
     split(/[\s/]+/).pop()).then(function (data) {
2                       $scope.formatData(JSON.parse(data.data))
                           ;
3         });
```

Die $*http.get*()-Funktion sowie die $*scope*-Variable stammen aus dem angular.js-Kontext. Die Funktion kommuniziert mittels http-get-request mit dem Simulation-Analysis-Controller. $*scope* wird später dazu verwendet, das HTML-Vokabular so zu erweitern, dass mittels der Anweisung {{Variable}} in Javascript berechnete/-ausgelesene Werte direkt im HTML-Umfeld verwendet werden können. Hierzu wird in JavaScript ein temporärer Controller erstellt, dem unter Anderem die $*scope*-Variable als Parameter übergeben wird. Außerdem erhält dieser Controller einen eindeutigen Namen (in diesem Fall „simulationAnalysisController“), der als Angular-Controller an ein Element im HTML-Kontext übergeben werden kann. Dieses Element bildet dadurch den Gültigkeitsbereich, innerhalb dessen mit der {{Variablen-Name/Objektname}}-Schreibweise zugegriffen werden kann. Diese Erweiterung sieht folgendermaßen aus:

Listing 5.7: Temporärer Angular-Controller

```
1 comics.controller('simulationAnalysisController', function (
    $scope, $http, $location) {
2       /*
3               - Weiterverarbeiten der Daten
4               - Erstellen der Diagramme
5       */
6 };
```

Kernstück dieses Controllers bildet die Erzeugung von $*scope.formatData*, welche beim Abruf der Simulationsdaten im weiter oben erwähnten $*http.get*(....)-Befehl aufgerufen wird. Die Funktion, die $*scope.formatData* erzeugt, erwartet als Parameter ein JSON-Objekt, welches den angefragten Datensatz beinhaltet. Außerdem werden an dieser Stelle das Kreisdiagramm, die Event-Diagramme und die Tabelle mit den allgemeinen Simulationsinformationen berechnet und erzeugt.

Listing 5.8: $scope.formatData

```
1 $scope.formatData = function (json) {
2       // ...
3 };
```

Der erste Schritt innerhalb der gerade dargestellten Funktion wird es nun sein, die Daten der Simulation zu sortieren. Hierzu wird das Objekt-Array *simu* erzeugt und

mit den Daten die der Controller an den View geliefert hat gefüllt. Anschließend werden die Objekte aus *simu* auf $*scope.data* übertragen.

Listing 5.9: Übertragen der Werte in die Variable „simu“

```
$scope.formatData = function (json) {
var simu = {};
simu.simId = json.Run.SimID;
simu.start = json.Run.StartYear;
simu.end = json.Run.EndYear;
simu.agents = json.Run.Agents;

simu.events = {};
json.Statistics.forEach(function (year) {
        $scope.ticks.push(year.Year);
        year.Events.forEach(function (event) {
        if (!simu.events[event.Name]) {
                simu.events[event.Name] = [];
                simu.events[event.Name].push([]);
                }
        simu.events[event.Name][0].push(event.Value);
                });
        });
        // Anzahl der Events bestimmen
        simu.eventCount = Object.keys(simu.events).length;
};
```

Die Kapitel 5.3.3 bis einschließlich 5.3.6 konkretisieren als Nächstes die Umsetzung der einzelnen Ziele.

5.3.3 Anzeige allgemeiner Simulationsinformationen

Der in Listing 5.4 vorhandene Tabellenrumpf wird so erweitert, dass automatisiert die entsprechenden Werte dort angezeigt werden. Der resultierende HTML-Quellcode sieht wie folgt aus:

Listing 5.10: HTML-Quellcode der Tabelle

```
<table id="generalInfoTable">
<tr>
```

```
3           <th colspan="2">Allgemeine Informationen</th>
4   </tr>
5   <tr>
6           <td>Simulations-ID</td>
7           <td>{{data.simId}}</td>
8   </tr>
9   <tr>
10          <td>Zeitraum</td>
11          <td>{{data.start}} - {{data.end}}</td>
12  </tr>
13  <tr>
14          <td>Anzahl Agenten</td>
15          <td>{{data.agents}}</td>
16  </tr>
17  <tr>
18          <td>Events</td>
19          <td>{{data.eventCount}}</td>
20  </tr>
21  <tr ng-repeat="(eventname, event) in data.events">
22          <td></td>
23          <td>{{eventname}}</td>
24  </tr>
25  </table>
```

Hier zeigt sich nun auch, wie die Angular-Direktive {{...}}. Der Angular-Controller bekommt als einen seiner Parameter $\$scope$ übergeben. Wie bereits zuvor erklärt, werden die in *simu* gespeicherten Informationen $\$scope.data$ übertragen. Im HTML-Kontext ist es nun (innerhalb des Angular-Controller-Gültigkeitsbereich) möglich mit beispielsweise {{data.simId}} auf die Daten der Simulationsergebnisse zuzugreifen. Analog dazu werden in der Tabelle die weiteren Daten abgerufen. Eine weitere Angular-spezifische Methode stellt das Attribut $ng-repeat = "(eventname, event) in data.events"$ dar. Diese Direktive arbeitet so, dass sie für alle in *data.events* vorhandenen Einträge eine Tabellenreihe mit zwei Spaltenzellen erzeugt, wobei in letztere der beiden der Eventname mittels {{eventname}} geschrieben wird. Dies ermöglicht es auf einfachem Wege eine Auflistung aller in der Simulation vorhandenen Events in die Tabelle einzufügen.

Diese Tabelle ist der erste Teil, der vom Details-View im Browser angezeigt wird. Ein beispielhaftes Ergebnis zeigt Abbildung 5.2.

Abbildung 5.2: Tabelle für allgemeine Informationen

Allgemeine Informationen	
Simulations-ID	3
Zeitraum	2010 - 2050
Anzahl Agenten	170
Events	3
	Init
	Geburt
	Tod

Quelle: eigener Screenshot

5.3.4 Prozentuale Verteilung der Events

Um die prozentuale Verteilung der Events in einem Kreisdiagramm anzuzeigen zuerst pro Event berechnet werden, wie oft er insgesamt auftrat. Außerdem wird ein HTML-Element (in diesem Fall ein div) benötigt, welches über eine *id* eindeutig angesprochen werden kann. Diese zwei Komponenten sind Voraussetzung dafür, dass *jqPlot* das Diagramm entsprechend rendern kann. Das benötigte div-Element ist bereits im oben aufgeführten HTML-Grundgerüst vorhanden und mit der id $events-pie$ gekennzeichnet.
Die Implementierung hierzu erfolgte in zwei Schritten. Der erste beinhaltet die Generierung von *pieArray*. Das resultierende Objekt enthält alle Events gepaart mit der jeweiligen Anzahl an Durchläufen.

Listing 5.11: Daten für das Kreisdiagramm vorbereiten

```
1 var simulationData = Object.entries(simu.events);
2 var pieArray = [];
3 simulationData.forEach(function (index) {
```

```
4           var eventName = index[0];
5           var values = index[1][0];
6           var total = 0;
7           // Werte des Event aufsummieren
8           values.forEach(function (i) {
9                   total += i
10          });
11          pieArray.push(['' + eventName + '', total]);
12  });
```

Im folgenden Schritt wird das eigentliche Kreisdiagramm erstellt. Den Code zeigt das folgende Listing:

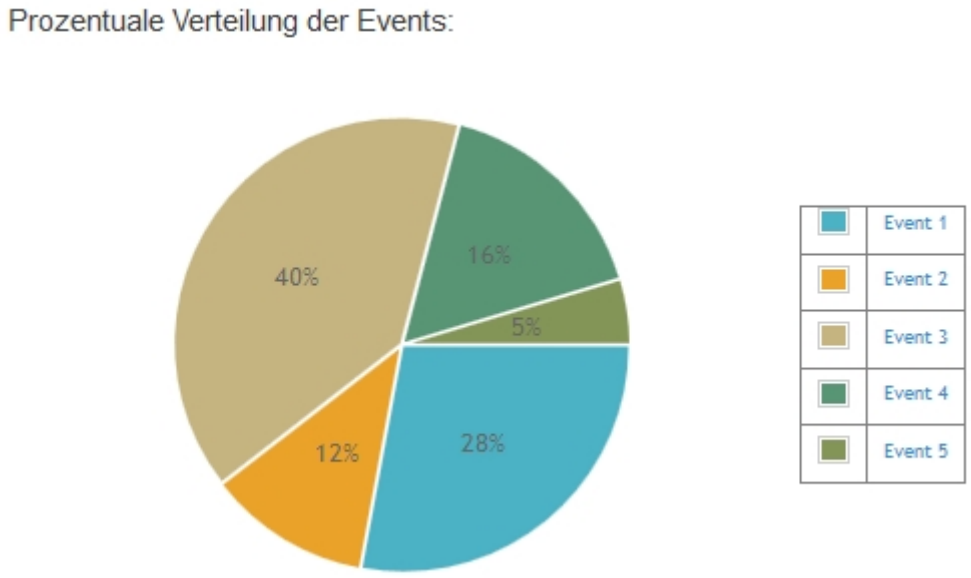
Listing 5.12: Kreisdiagramm erstellen

```
1   $(document).ready(function () {
2   var eventsPie = $.jqplot('events-pie', [pieArray], {
3           seriesDefaults: {
4                   renderer: $.jqplot.PieRenderer,
5                   rendererOptions: {
6                   showDataLabels: true,
7                   sliceMargin: 2
8                   },
9                   shadow: false
10          },
11          grid: {
12                  background: '#ffffff',
13                  shadow: false,
14                  borderWidth: 0.1,
15                  drawBorder: true
16          },
17          legend: {
18                  placement: 'inside',
19                  location: 'e',
20                  show: true
21          }
22  });
23  });
```

Der jQuery-Befehl aus Zeile 1 stellt sicher, dass der innere Code der Funktion nur dann ausgeführt wird, wenn das HTML-Dokument inklusive aller Elemente fertig vom Browser gerendert wurde. Dadurch ist gewährleistet, dass die jqplot-Funktion über den Identifier *events − pie* (Zeile 2) auch auf das entsprechende div-Element zugreifen und dort das Diagramm darstellen kann.
jqPlot bietet eine sehr große Anzahl an möglichen Parametern bei der Erstellung von Diagrammen. Im obigen Beispiel ist zu erkennen, dass es drei Konfigurationsbereiche gibt, die für dieses Diagramm angepasst wurden. „seriesDefaults“ enthält allgemeine Definitionen für das Diagramm, wie beispielsweise den zu verwendenden Renderer - in diesem Fall der PieRenderer. Diese Anweisung bewirkt, dass die Datei „jqplot.pieRenderer.js“ im Verzeichnis „Scripts/jqplot/“ für die grafischen Berechnungen dieses Diagramms genutzt wird.
Um das Erscheinungsbild des Diagramms an aktuell gängige Designrichtlinien (Bsp.: Flat-Design[4] anzupassen, wurden sämtliche Schatteneffekte entfernt. Um das Interface aufgeräumt zu erhalten, bekommen die einzelnen Kreissegmente nur die Prozentangaben mit und eine separate Legende zeigt, welche Farbe für welchen Event steht. „location: ’e’“ gibt in diesem Fall die Himmelsrichtung „east“ (bezogen auf das Diagramm) an, in der die Legende positioniert werden soll. „sliceMargin“ legt fest, wie groß der Abstand zwischen den einzelnen Kreissegmenten ist. Mittels „showDataLabels:true“ bewirkt man, dass die Prozentangaben auf den Segmenten angezeigt werden - die Berechnung der entsprechenden Werte geschieht automatisch und leitet sich aus der Gesamtzahl aller Eventdurchläufe ab. Ein Kreisdiagramm mit fünf Events kann beispielsweise wie in Abbildung 5.3 aussehen.

[4] Flat Design ist ein minimalistisches Gestaltungsprinzip, welches in den vergangenen Jahren immer präsenter wurde und vor Allem der rasanten Verbreitung von Apps geschuldet ist.

Abbildung 5.3: Prozentuale Verteilung der Events



Quelle: eigener Screenshot

5.3.5 Balkendiagramm pro Event

Alle Events einer Simulation sollen in Form von Balkendiagrammen visualisiert werden. Die x-Achse zeigt alle in der Simulation durchlaufenen Jahre an, die Werte der y-Achse zeigen auf, wie oft das jeweilige Event pro Jahr durchlaufen wurde. Die Werte der x-Achse werden in „jqplot“ als sogenannte „ticks“ bezeichnet. Lässt man diese weg, würde „jqplot“ eigene Werte aufsteigend angeben, was in diesem Kontext keinen Sinn ergeben würde. Die Werte für die x-Achse sind für alle Events identisch und wurden bereits in $\$scope.ticks$ abgelegt. Für das Einbinden in die $\$jqplot$-Funktion muss jedes Element in diesem Array noch mittels $\$scope.ticks = (\$scope.ticks).map(String)$; umgewandelt werden, da die „ticks“ in String-Form erwartet werden.
Zuerst muss für jedes vorhandene Event ein eigenes div-Element angelegt werden.

Listing 5.13: Pro Event ein div-Element erstellen

```
1 var numberOfEvents = Object.keys(simu.events).length;
2 for (eventIndex = 1; eventIndex <= parseInt(numberOfEvents)
     ; eventIndex++) {
3         $("#chartContainer").append('<div id="chart' +
             eventIndex + '" class="jqplot-target"></div><hr>')
```

```
            ;
}
```

Hierzu wird die Anzahl der Events bestimmt und pro Event mit jQuery das bereits vorhandene div mit der ID *chartContainer* um zusätzliche div-Elemente erweitert. Dies geschieht mittels der *.append*()-Funktion. Die hinzugefügten Elemente bekommen die ID *chart* mit einer fortlaufenden Indizierung um somit nachfolgend ein genaues Ansprechen und Platzieren der Diagramme zu ermöglichen. $< hr >$ fügt zwischen jedem Event eine Trennlinie ein.
Der folgende Quellcode zeigt, wie $*jqplot* für jedes Event innerhalb einer Schleife definiert wird:

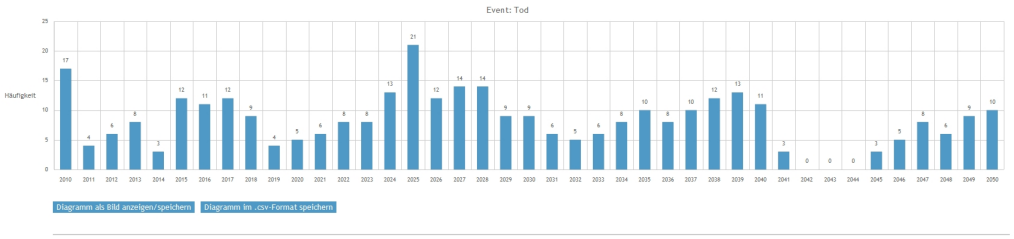
Listing 5.14: Generierung der Balkendiagramme

```
var chartID = "chart" + chartCount.toString();
$.jqplot(chartID, eventValues, {
        title: "Event: " + eventName,
        animate: !$.jqplot.use_excanvas,
        seriesColors: ['#4f99c6'],
        axesDefaults: {
                tickOptions: {
                        showMark: false
                }
        },
        seriesDefaults: {
                renderer: $.jqplot.BarRenderer,
                rendererOptions: {
                        shadowOffset: 0
                },
                pointLabels: { show: true }
        },
        axes: {
                xaxis: {
                        renderer: $.jqplot.
                           CategoryAxisRenderer,
                        ticks: $scope.ticks
                },
                yaxis: {
```

```
24                          min: 0,
25                          label: 'Häufigkeit'
26                  }
27          },
28          grid: {
29                  background: '#ffffff',
30                  shadow: false,
31                  borderWidth: 0.1,
32                  drawBorder: true
33          },
34          highlighter: { show: false }
35  });
36  chartCount += 1;
```

Der soeben gezeigte Quellcode befindet sich innerhalb einer Schleife, in welcher zuvor die Werte pro Event mittels $vareventValues = index[1]$; ausgelesen werden. Diese Werte und die generierte $chartID$ werden an die $\$jqplot$-Funktion übergeben. Ähnlich wie zuvor bei der Erstellung des Kreisdiagramm werden bei den Balkendiagrammen ebenfalls einige Optionen eingefügt um Aussehen und Verhalten zu steuern. Beispielsweise muss nun über $\$.jqplot.BarRenderer$ die entsprechende JavaScript-Datei für Balkendiagramme angesteuert werden. In puncto Design wurde erneut auf ein klares Layout geachtet und mit $seriesColors : ['\#4f99c6']$, ein Farbwert eingebunden, der bereits innerhalb von „CoMicS“ Verwendung findet. Die Überschrift der Balkendiagramme ergibt sich aus dem *title*-Attribut, welches zuvor durch $vareventName = index[0]$; definiert wurde. *pointLabels* mit dem Wert *true* zu belegen führt dazu, dass über jedem Balken der entsprechende Wert zusätzlich als Zahl angezeigt wird - dies erhöht die Übersichtlichkeit des Diagramms.
Ein Balkendiagramm mit diesen Einstellungen sieht wie folgt aus:

Abbildung 5.4: Balkendiagramm für einen Event



Quelle: eigener Screenshot

Im nächsten Schritt wird über $createImageFromPlot()$; die Funktion aufgerufen, die jedem Diagramm einen Button hinzufügt. Sie sucht mit dem jQuery-Selektor $\$("div.jqplot-target")$ alle div-Elemente der Klasse „jqplot-target“, errechnet aus den Werten eine Grafik und erzeugt einen Button. Ein Klick auf diesen Button bewirkt, dass sich das errechnet Bild des Diagramms unterhalb selbigen öffnet und über „Rechtsklick-Grafik speichern unter“ gesichert werden kann.

Abbildung 5.5: Balkendiagramm für einen Event speichern

Quelle: eigener Screenshot

Die Funktion $createImageFromPlot()$; bietet ebenfalls den Ansatzpunkt für die fehlende Erweiterung um ein Diagramm im .csv-Format abspeichern zu können. Daher wird diese Funktionalität unmittelbar nach der Erstellung der Bild-Download-Button eingefügt. Es wird ein weiterer Button erstellt und mit der Klasse $jqplot-csv-button$ gekennzeichnet. Diese CSS-Klasse und auch die für den Bilder-Download-Button wer-

den in der „examples.min.css“-Datei so gestyled, dass sie sich in die bestehende CI einfügen.
Die Erstellung der .csv-Datei wird erst bei einem Klick auf den entsprechenden Button angestoßen, damit nicht alle initial erzeugt werden, was die Ladezeit der Seite erhöhen würde. Ein Klick auf einen der Button bewirkt, dass das vorher abgebildete Diagramm analysiert wird. Eventname, chart-ID sowie die x- und y-Werte werden aus der HTML-Struktur ausgelesen und entsprechend weiterverarbeitet. Wenn der csv-Daten-String erstellt wurde, erzeugt die Funktion einen nicht sichtbaren Link, der die jeweiligen Informationen(Werte, Dateiname = Eventname) enthält und simuliert einen Klick auf diesen Link, was den eigentlichen Download der .csv-Datei erzeugt. Der Quellcode für diesen Abschnitt ist im nun folgenden Listing zu sehen:

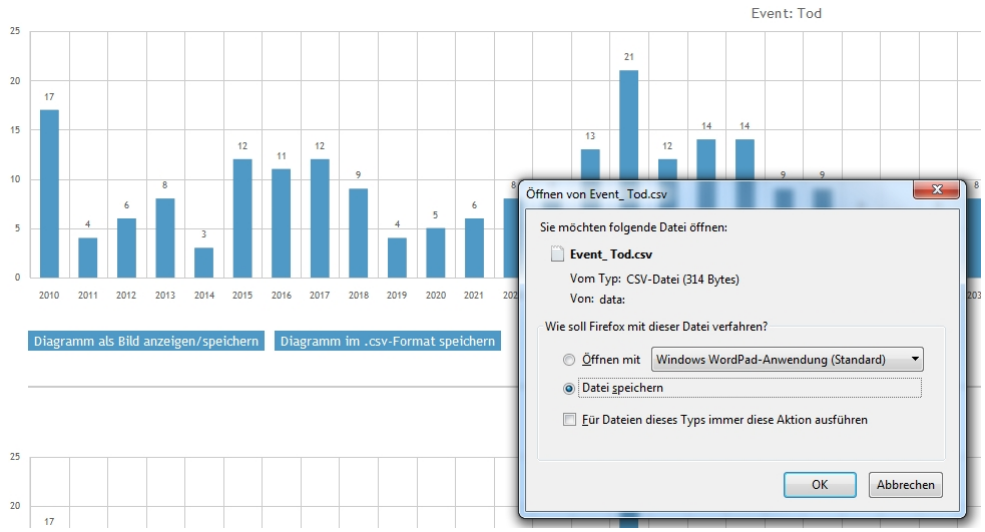
Listing 5.15: Implementierung csv.-Datei Download

```
1  var c2 = $(document.createElement("button"));
2  c2.text("Diagramm im .csv-Format speichern");
3  c2.addClass("jqplot-csv-button");
4  c2.bind("click", { chart: $(this) }, function (h2) {
5  // Diagramm-Title auslesen
6  var title = h2.data.chart.context.childNodes[1].innerText;
7
8  // chart-ID auslesen
9  var chartId = $("div:contains(" + title + "):last").parent()
       .attr('id');
10
11 // Beschriftungen (ticks) der x-Achse auslesen und im Array
       'xTicks' ablegen
12 var xTicks = (h2.data.chart.context.childNodes[2].innerText)
       .match(/.{1,4}/g);
13
14 // Werte der einzelnen Jahre bestimmten
15 var xValues = [];
16 $('#' + chartId + ' > div.jqplot-point-label').each(function
       (index) {
17         xValues[index] = $(this).text();
18 })
19 // jqPlot gibt die Werte im HTML-Kontext in umgekehrter
```

```
    Reihenfolge aus,
20  // weshalb sie hier wieder umgedreht werden müssen
21  xValues = xValues.reverse();
22
23  /*  Spaltenwerte für eine CSV-Datei werden mit '%2C'
        getrennt, eine
24      neue Zeile mit %0A eingeleitet. Entsprechend wird der
            String aufgebaut
25  */
26  var csvString = "Jahr%2CAnzahl%0A";
27  for (var i = 0; i < xValues.length; i++) {
28          csvString = csvString.concat(xTicks[i] + "%2C" +
                xValues[i] + "%0A");
29  };
30  var csvData = "data:application/csv;charset=utf-8," +
        csvString;
31
32  //   Unsichtbaren Link erstellen, der auf die CSV-Datei
        zeigt.
33  var csvLink = "<a id='" + chartId + "-a' download='" + title
        + ".csv' href='" + csvData + "' style='visibility:hidden
        ;'>csv</a>";
34  $(this).after(csvLink);
35  document.getElementById(chartId + "-a").click();
36  });
37  // csv.-Button nach Bild-Downloadbutton 'c' anfügen
38  c.after(c2);
39  // die beiden Buttonobjekte für den nächsten
        Schleifendurchgang löschen
40  c = null;
41  c2 = null;
```

Die Abbildungen 5.6 und 5.7 zeigen den Speicherdialog und den Inhalt einer heruntergeladenen .csv-Datei.

Abbildung 5.6: Diagrammwerte als .csv-Datei speichern



Quelle: eigener Screenshot

Abbildung 5.7: Inhalt einer heruntergeladenen .csv-Datei

	Jahr	Anzahl
1	2010	17
2	2011	4
3	2012	6
4	2013	8
5	2014	3
6	2015	12
7	2016	11
8	2017	12
9	2018	9
10	2019	4
11	2020	5
12	2021	6
13	2022	8
14	2023	8
15	2024	13
16	2025	21
17	2026	12
18	2027	14
19	2028	14
20	2029	9
21	2030	9
22	2031	6

Quelle: eigener Screenshot

5.3.6 Verlinkung innerhalb der Software

Die in Kapitel 5.2.4 auf Seite 63 erwähnte Seite wird über die Datei Details.cshtml gesteuert, die unter „Comics.Web/Views/Simulation“ abgelegt ist. Um hier einen But-

ton, der auf die grafische Ergebnisauswertung verlinkt, zu implementieren, wird die bestehende Struktur beibehalten und mit einer Pfadanpassung versehen, was in folgendem Code resultiert:

Listing 5.16: Verlinkung auf die grafische Ergebnisauswertung

```
1 <div class="col-md-2">
2         <a class="btn btn-sm btn-primary btn-block" href="/
             SimulationAnalysis/Details/@Model.SimulationId">
3             <i class="icon-bar-chart align-top"></i>
4             Ergebnisauswertung
5         </a>
6     </div>
```

Zusätzlich wurde an dieser Stelle die Breite aller drei Button jeweils halbiert um den oberen Teil dieser Seite aufgeräumter zu gestalten.

Abbildung 5.8: Verlinkung grafische Ergebnisauswertung

Quelle: eigener Screenshot

Durch diesen Aufruf wird die an vorheriger Stelle erzeugte Datei Details.cshtml („Comics.Web/Views/SimulationAnalysis“) angezeigt, die die grafische Auswertung der entsprechenden Simulation beinhaltet.

6 Fazit und Ausblick

Die Visualisierung von Ergebnissen einer Mikrosimulation ist für den Benutzer ein definitiver Mehrwert. Anders als bei der reinen Darstellung von Zahlenwerten in Text- oder Tabellenform hat man die Möglichkeit Zusammenhänge/Muster anders bzw. überhaupt wahrzunehmen. Dies liegt besonders daran, dass das menschliche Gehirn eine reine Ansammlung von Zahlen ab einer gewissen Menge nur sehr schwer und schon gar nicht sinnvoll filtern kann. In unserer heutigen Zeit sind Computer und Tools so schnell und leistungsstark wie nie zuvor, weshalb es im Forschungsgebiet der Datenvisualisierung immer wichtiger wird, hierauf ein verstärktes Augenmerk zu legen.
Die definierten Ziele dieser Diplomarbeit wurden erreicht und insgesamt konnte „CoMicS“ um eine grafische Ergebnisauswertung erweitert werden, welche modular und unabhängig von der eigentlichen Simulation (ihrem Umfang und ihrer Struktur) arbeiten kann. Als Manko kann rückblickend betrachtet werden, dass der Fokus zu sehr auf den Aspekt der Events gelegt wurde. Die Events bilden zwar ein wichtiges Kernstück jeder Mikrosimulation und bieten auch bereits einige Möglichkeiten einer Ergebnis- bzw. Funktionsanalyse, allerdings sind die Ergebniswerte der Agenten/der Population mindestens genauso interessant. Die implementierte Datenauswertung kann daher als Basis für zukünftige Erweiterungen betrachtet werden. Die Strukturen und Datenverbindungen wurden hierfür bereits geschaffen. Ein nächster Schritt kann und sollte daher sein, dass die Daten der Agenten mit in die Analyse einbezogen und optimalerweise in Verbindung zu den Events gesetzt werden. Der kritische Punkt wird es hierbei sein, dass trotz zunehmender Komplexität der Datenvisualisierung weiterhin das Modularitätsprinzip bewahrt bleiben kann, sodass „CoMicS“ kontinuierlich in diesem Bereich angepasst werden kann, ohne dass es in puncto Simulationsmodelle oder -abläufe Einschränkungen geben muss.
Ein weiteres denkbares Szenario wäre es, dass die Software um eine konkretere Plausibilitätsprüfung erweitert wird - nicht nur im Sinne von weiteren aussagekräftigen

Darstellungen, sondern schon auf programmtechnischer Ebene. Beispielsweise könnte man dem Nutzer die Möglichkeit zur Verfügung stellen, dass bei der Simulation permanent anhand eines Schwellwertes geprüft wird ob die Fortschreibung der Simulation noch richtig ist oder abgebrochen werden sollte. Um diese Entscheidung dem Nutzer selbst in die Hand zu legen, könnte man die Ergebnisse in Echtzeit während der Simulationsberechnung visualisieren lassen. Hierbei stellt sich allerdings, je nach Komplexität, die Frage, ob sich der zeitliche Aufwand dadurch im Endeffekt rechnet. Abschließend kann zusammengefasst werden, dass für Software im Bereich der Mikrosimulationen und hierbei vor Allem bei den Auswertungsmöglichkeiten der Ergebnisdaten, noch ein großes Entwicklungspotential und -bedarf besteht.

Literaturverzeichnis

[ANKERST et al., 1995] Ankerst, M., Kriegel H.P. & Keim D.A. (1995) *Recursive Pattern: A Technique for Visualizing Very Large Amounts of Data* veröffentlicht in „Proc. Visualization'95" Atlanta,GA,1995

[DEUSSEN-2003] Deussen, O. (2003) *Datenvisualisierung* (Skript zur Vorlesung „Computergrafik II" an der TU Dresden)

[HANNAPPEL UND TROITZSCH, 2014] Hannappel, M. & Troitzsch, K.G. (2014) *Mikrosimulationsmodelle*

[LI UND O'DONOGHUE, 2013] Li, J. & O'Donoghue, C. (2013) *A Survey Of Dynamic Microsimulation Models: Uses, Model Structure And Methodology*

[PHILIPSEN, 2015] Philipsen, Q. (2015) *(Explorative) Datenvisualisierung* Hochschule für Angewandte Wissenschaften Hamburg

[MORALES POSADA, 2015] MORALES POSADA, J.C. (2015) *(Datenvisualisierung, Informationsvisualisierung und Infographik* Universidad de Buenos Aires

[SHNEIDERMAN, 1996] Shneiderman, B. (1996) *The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations*

[SPIELAUER, 2009] Spielauer, M. (2009) *What Is Dynamic Social Science Microsimulation?*

[TIEFENAU, 2014] Tiefenau, C. (2014) *CoMicS - Entwicklung eines anwendungsorientierten Mikrosimulationssystems* (Master-Thesis, verfasst an der Hochschule Bonn-Rhein-Sieg)

[KEIM, 2004] Keim, D. A. (2004) *Datenvisualisierung und Data Mining* zuerst erschienen in „Grundlagen der praktischen Information und Dokumentation", München: Saur, 2004

Online-Quellen:

[HDF GROUP-2016] *https://support.hdfgroup.org/HDF5/* (Stand: 06. September 2016)

[GNU General Public License Version 3, 2016] *http://www.gnu.org/licenses/gpl-3.0.de.html* (Stand: 04. Okotber 2016)

[LIAM2 BOXPLOT, 2016] *http://liam2.plan.be/documentation/0.11/processes.html#boxplot* (Stand: 04. Okotber 2016)

[LIAM2 DATA-IMPORT, 2016] *http://liam2.plan.be/documentation/0.11/import.html* (Stand: 04. Okotber 2016)

[LIAM2 DOKUMENTATION, 2016] *http://liam2.plan.be/pages/documentation.html* (Stand: 04. Okotber 2016)

[LIAM2 GITHUB, 2016] *http://liam2.plan.be/pages/documentation.html* (Stand: 04. Okotber 2016)

[LIAM2 HOMEPAGE, 2016] *http://liam2.plan.be* (Stand: 04. Okotber 2016)

[LIAM2 SCATTER PLOTS, 2016] *http://liam2.plan.be/documentation/0.11/processes.html#scatter-plots* (Stand: 04. Okotber 2016)

[LIAM2 0.11, 2015] *http://liam2.plan.be/index.html* (Stand: 04. Okotber 2016)

[MATPLOTLIB, 2016] *http://www.matplotlib.org* (Stand: 10. September 2016)

[MATPLOTLIB API PIE, 2016] *http://matplotlib.org/api/pyplot_-api.html#matplotlib.pyplot.pie* (Stand: 10. September 2016)

[MATPLOTLIB API PLOT, 2016] *http://matplotlib.org/api/pyplot_-api.html#matplotlib.pyplot.plot* (Stand: 10. September 2016)

[MATPLOTLIB API SCATTER, 2016] *http://matplotlib.org/api/pyplot_-api.html#matplotlib.pyplot.scatter* (Stand: 10. September 2016)

[MIT LICENSE, 2016] *https://opensource.org/licenses/MIT* (Stand: 04. Okotber 2016)

[MODGEN BIOBROWSER GUIDE, 2016] *BioBrowser User's Guide v4.0.3* (Stand: 30. Juni 2016)

[MODGEN DEVELOPERS GUIDE, 2016] *Modgen Developer's Guide v10.1.0* (Stand: 30. Juni 2016)

[MODGEN DOWNLOAD, 2016] *http://www.statcan.gc.ca/eng/microsimulation/modgen/download* (Stand: 30. Juni 2016)

[MODGEN HOMEPAGE, 2016] *http://www.statcan.gc.ca/eng/microsimulation/modgen/modgen* (Stand: 07. August 2016)

[MODGEN VISUAL INTERFACE GUIDE, 2016] *Guide to the Modgen Visual Interface v10.1.0* (Stand: 30. Juni 2016)

[NOTEPAD++, 2016] *https://notepad-plus-plus.org/* (Stand: 04. Okotber 2016)

[PWC, 2013] PricewaterhouseCoopers Aktiengesellschaft Wirtschaftsprüfungsgesellschaft (2013) *Big Data - Bedeutung, Nutzung, Mehrwert* online verfügbar unter: https://www.pwc.de/de/prozessoptimierung/assets/pwc-big-data-bedeutung-nutzen-mehrwert.pdf (Stand: Juni 2013)

[STATCAN HOMEPAGE, 2106] *http://www.statcan.gc.ca/eng/start* (Stand: 07. August 2016)

[KRYPCZYK, 2014] Krypczyk, V. (2014) *Datenvisualisierung: Theorie und Praxis* online verfügbar unter: https://entwickler.de/online/datenbanken/datenvisualisierung-theorie-und-praxis-114322.html

[WiWo, 2015] Magazin Wirtschaftswoche (04/2015) *Big Data: 2,5 Trillionen Byte Daten jeden Tag, wächst vier Mal schneller als Weltwirtschaft* online verfügbar unter: http://blog.wiwo.de/look-at-it/2015/04/22/big-data-25-trillionen-byte-daten-jeden-tag-wachst-vier-mal-schneller-als-weltwirtschaft/ (Stand: 22. April 2015)