TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

D I P L O M A R B E I T

Integration von geographischen Daten in ein agentenbasiertes mathematisches Modell zur Versorgungsanalyse im österreichischen Gesundheitssystem

Ausgeführt am Institut für
Analysis und Scientific Computing
der Technischen Universität Wien

unter der Anleitung von
Ao.Univ.Prof.DI.Dr. Felix Breitenecker

durch

Georg Romstorfer
Kesslerweg 17
2345 Brunn am Gebirge, Österreich

Datum

Unterschrift (Student)

Kurzfassung

Das österreichische Gesundheitssystem ist ein komplexes System in dem verschiedenste Parteien interagieren. Die Sozialversicherungen, Ärzte und Krankenanstalten, der Staat und nicht zuletzt die Patienten sind nur ein paar davon. Um dieses System besser zu verstehen und Auswirkungen von eventuellen Eingriffen abschätzen zu können bedarf es der Modellbildung. Unter Sparmaßnahmen oder anderen Veränderungen kann die Versorgungsdichte mit medizinischen Leistungen vor allem in ländlichen Regionen leiden. An diesem Beispiel ist zu sehen, dass die Integration von geographischen Daten in diese mathematischen Modelle notwendig ist um entsprechende Effekte in der Simulation zu erkennen. Diese Arbeit befasst sich daher mit der Integration von geographischen Daten wie zum Beispiel der Häufigkeitsverteilung der Bevölkerung in ein bereits bestehendes agentenbasiertes Modell des österreichischen Gesundheitssystems.

Zu Beginn steht ein kurzer Überblick über existierende Modelle mit Geodaten die sich mit dem Thema Gesundheit befassen. Danach werden die Modellierungsmethoden „Agentenbasierte Modellierung“ und „Diskrete Ereignis Simulation“ erklärt und verglichen. Weiters werden geographische Informationssysteme thematisiert, wobei näher auf die Definition geodätischer Koordinatensysteme eingegangen und die Umrechnung zu kartesischen Koordinaten hergeleitet wird. Ebenfalls wird auf die Vielfalt an verschiedenen Datenformaten und Datenquellen eingegangen und für Österreich frei verfügbare Daten aufgelistet.

Der Hauptteil dieser Arbeit befasst sich mit der Integration von Geodaten in agentenbasierte Modelle und stellt einen Leitfaden zur Verfügung, mit dem es möglich sein soll diese Aufgabe auch in anderen Modellen zu bewältigen.

Probleme, die bei der Integration auftraten, werden genauer untersucht. Einerseits ist das die räumliche Verteilung von Agenten entsprechend einer (relativen) Häufigkeitsverteilung, wozu zwei Algorithmen analysiert und verglichen werden. Andererseits wird die Interpolation von Wegdistanzen mittels Splines behandelt, wodurch während der Simulation eine schnellere Auswertung der Wegdistanzfunktion zwischen zwei beliebigen geographischen Punkten ermöglicht werden soll.

Der erwähnte Leitfaden wurde schließlich auch an dem Modell des Gesundheitssystems erprobt. Nach einer kurzen Beschreibung des Modells und der verwendeten Daten werden Ergebnisse der Simulation mit Geodaten im Vergleich zu dem Szenario ohne Geodaten präsentiert.

Abstract

The Austrian public health system is complex and includes various different interacting parties including social insurancies, medicals, hospitals, federal departments and patients. Modelling and simulation is a way to better understand the system and the impact of various interventions. Cost-cutting measures or other changes in the system can lead to a decrease of health supply density particularly in rural areas. This example shows that the integration of geographic data in these models is necessary to detect such effects. For this reason the present work deals with the integration of geographic data into an already existing model of the Austrian public health system.

At the beginning there is a short overview on existing models using geographic data which is followed by a discussion and comparison of "agent-based modelling" and "discrete event simulation". Afterwards the issue of geographic information systems is addressed. The definition of geodetic coordinate systems is explained in more detail and a proof for the conversion into cartesian coordinates is given. The great variety of different geographic data formats and sources is highlighted and a list of Austrian free of charge data sources is given.

The main part of this work concentrates on the integration of spatial data into agent-based models and is meant to be a guide for other modellers that have to accomplish that same task. Problems that occurred during integration are discussed in more detail: For the spatial distribution of agents according to a (relative) frequency distribution two algorithms are analysed and compared; and the interpolation of way-distances by splines is explained, which is a possibility to enable a fast evaluation of the way-distance function between two arbitrary geographic points.

To prove the applicability of the mentioned guide geographic data was incorporated into the model of the health system. After a short description of the model and the used data results of the simulation with and without geographic data are compared and presented.

Danksagung

An dieser Stelle möchte ich meinen Eltern danken, die mich während meines Studiums unterstützt haben. Ganz besonderer Dank gilt auch meiner Lebensgefährtin Mag.ª Nathalie Harbourn, die mich bei der Verfassung dieser Arbeit immer motiviert hat. Meinem Professor Ao.Univ.Prof.DI.Dr. Felix Breitenecker möchte ich auch herzlichen Dank aussprechen, insbesondere für die Erfahrungen, die ich auf der Toskana-Reise sammeln konnte. Bei DI Günther Zauner und DI Patrick Einzinger möchte ich mich ebenfalls bedanken, da sie mir einiges ihrer Zeit bei der Betreuung dieser Diplomarbeit geopfert haben.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Um ein besseres Verständnis des österreichischen Gesundheitssystems und im Speziellen dessen Inanspruchnahme zu erlangen gibt es im Allgemeinen drei Problemlösungsmöglichkeiten. Eine davon wäre, eine etablierte Theorie zu Gesundheitssystemen zu verwenden um daraus Erkenntnisse zu gewinnen. Da diese Systeme von vielen Parametern abhängen und keine oder kaum geltende Gesetze bekannt sind, gibt es diese Theorie noch nicht. Experimente mit dem System könnte man ebenfalls verwenden um mehr Einsicht zu erlangen. Probleme dabei sind, dass Eingriffe in das Gesundheitssystem unter Umständen erst Jahre später Auswirkungen zeigen und der Anfangszustand des Systems nicht mehr wiederhergestellt werden kann, um weitere Experimente durchzuführen. Modellierung und Simulation kann als dritte Möglichkeit verwendet werden um Problemstellungen zu lösen. So können mit einem Modell des österreichischen Gesundheitssystems verschiedene Experimente durchgeführt oder Szenarien simuliert werden. Dabei können Fragen zur Inanspruchnahme der medizinischen Versorgungseinrichtungen im Modell analysiert werden. Das Modell ist aber immer nur ein Abbild der Realität, welches diese nur (beliebig genau) approximiert. Die Integration von GIS-Daten ist ein entscheidender Schritt um diesen Modellen einen näheren Bezug zur Realität zu verschaffen. Somit ist es leichter die Validität der Modelle zu überprüfen und auch die Visualisierung der Simulationsergebnisse kann so mit empirischen Echtdaten verglichen werden.

1.1 Aufgabenstellung

Basierend auf einem bestehenden Modell des österreichischen Gesundheitswesens sollen geographische Daten aus verschiedenen Datenquellen in dieses Modell integriert werden. Die Integration dieser Daten soll in einer Art und Weise geschen, die auch für andere Modelle wiederverwendbar ist. Es soll außerdem untersucht werden, welche freien Quellen es für Geodaten gibt und wie diese genützt werden könnnen. Eine wichtige Rolle spielt auch die Anbindung bestehender Datenbanksysteme und die Umsetzung der graphischen Darstellung von Simulationsmodelloutputs.

1.2 Modellbildung im Gesundheitswesen

Nach Brailsford [L4] kann die Modellbildung im Gesundheitswesen in drei verschiedene Bereiche unterteilt werden:

- Modelle des menschlichen Körpers, Krankheits-Modelle
- Modelle medizinischer Versorgungsstellen
- Strategische Modelle des ganzen Systems

Modelle des *menschlichen Körpers* und Krankheits-Modelle befassen sich mit physiologischen Vorgängen im Körper, Auswirkungen verschiedener Behandlungsmethoden, Impfstrategien, Ausbreitung von Krankheiten, Epidemien, Screening, etc.

Unter der Modellierung *medizinischer Versorgungsstellen* versteht man das Lösen verschiedener Optimierungsaufgaben im Bereich von Spitälern, Krankenstationen oder einzelnen Abteilungen. Gängige Fragestellungen in diesem Bereich sind: Wie viele Betten, Ärzte, Krankenschwestern werden benötigt um einen bestimmten Patientenzustrom behandeln zu können? Nach welchen Kriterien sollen Patienten priorisiert werden?

Strategische Modelle des ganzen Systems versuchen unter anderem Antworten auf folgende Fragen zu finden: Wie müssen Standplätze für Krankentransporte gewählt werden um das Einsatzgebiet möglichst effizient abzudecken? Wie wirken sich bestimmte Änderungen im Gesundheitssystem auf die

Versorgungsqualität der Patienten aus? Welchen Einfluss haben verschiedene Bezahlsysteme für Leistungserbringer auf das Gesundheitswesen?

1.3 Modelle mit GIS-Daten

In vielen der vorher genannten Bereiche macht es Sinn geographische Information miteinzubeziehen. Besonders bei agentenbasierter Modellierung ist das einfach möglich indem man den Agenten eine zusätzliche Eigenschaft, den geographischen Raumbezug, zuweist. In der Literatur finden sich für diese Art von Modellen mit geographischer Information einige Beispiele:

1. Perez et al. [L30] implementierten ein agentenbasiertes Modell zur Ausbreitung von Masern im städtischen Umfeld. Die Agenten kommen dabei an Arbeitsplätzen, Schulen, Universitäten, Einkaufszentren oder auch am Weg dorthin miteinander in Kontakt. Für die Lage dieser Kontakträume wurden geographische Daten herangezogen.

2. King [L19] hat die Ausbreitung von Malaria im peruanischen Amazonasgebiet untersucht. Menschen und Moskitos wurden in diesem Modell als Agenten mit der natürlichen Umgebung in Verbindung gebracht. Die Umgebung bestand in diesem Fall aus einem Raster, in dem für jede Zelle die Landnutzung aus Satellitendaten bestimmt wurde. So konnte modelliert werden, dass die Moskitos in der Nähe von Gewässern ihre Eier legen und daher Siedlungen in deren Umgebung einem höheren Malaria-Risiko ausgesetzt sind.

3. Kennedy et al. [L18] haben ein agentenbasiertes Modell zur Ausbreitung von Krankheitserregern unter Makakken auf der Insel Bali erstellt. Dabei wurde auch besonders die geographische Situtation einbezogen um folgende Fragen zu beantworten: Wie schnell und auf welchen Routen breiten sich Krankheitserreger über die Insel aus? Welchen Einfluss haben Parameter für die Krankheitserreger auf die Übertragung und Ausbreitung? Ändern sich die Ergebnisse, wenn das Modell auch den Lebensraum der Menschen als Faktor berücksichtigt?

4. Germann et al. [L13] haben die mögliche Ausbreitung von Grippeviren ausgehend von 14 internationalen Flughäfen in den USA modelliert. Anstoß für diese Untersuchung war das damalige Auftreten von Vogelgrippeviren (H5N1) in Asien und deren Ausbreitung nach Osteuropa. Das agentenbasierte Modell sollte vor allem Strategien zur Ausbreitungseindämmung des Virus testen. Geographische Daten zur Populationsverteilung und zur Bewegung der Bevölkerung innerhalb des Landes wurden im Modell berücksichtigt.

Kapitel 2

Modellierungsmethoden

2.1 Agentenbasierte Modellierung

Während früher starke Annahmen und Vorraussetzungen getroffen wurden um zu einfacheren Modellen zu gelangen, versucht man heute mehr Details zu erfassen um so zu einem realistischeren Modell zu gelangen. Das Problem dabei ist, dass dadurch die Modelle komplexer werden und so ist es oft nur mehr schwierig ein globales Modell nach dem „top-down“-Prinzip zu erstellen. Beispiele für dieses Prinzip sind Differenzialgleichungen oder System Dynamics [L11]. Um zusätzliche Details modellieren zu können, müssen die zur Modellbeschreibung verwendeten aggregierten Größen in Teile mit unterschiedlichen Eigenschaften zerlegt werden. Ein Beispiel dafür ist die Verwendung von Alterskohorten anstelle einer einzigen Größe für eine zu modellierende Population. Nimmt man zehn Altersgruppen an und möchte zusätzlich auch noch zwischen den Geschlechtern unterscheiden, so werden 20 Größen zur Modellierung benötigt. Da sich die Eigenschaften auf diese Weise multiplizieren, wird das Modell schnell schwer handhabbar. Dem „top-down“-Prinzip steht das „bottom-up“-Prinzip gegenüber, welches nicht das System als Ganzes modelliert, sondern es in seine Einzelteile (die agierenden Komponenten) zerlegt. Im Fall der agentenbasierten Modellierung heißen diese Komponenten Agenten. Bei diesem Ansatz müssen nun nicht mehr Gleichungen für das ganze System gefunden werden, sondern nur noch Re-

geln für das Verhalten der einzelnen Agententypen und deren Interaktionen. Oft kann mit ein paar einfachen Regeln für die Agenten bereits ein komplexes Verhalten des gesamten Systems beschrieben werden. Die Auswirkungen von Regeln in der Ebene der Agenten auf makroskopische Eigenschaften oder Verhaltensweisen des Systems wird auch als „emerging behaviour“ [L6] bezeichnet. Bekannte Beispiele dafür sind z.B. das Game of Life [L12] oder das Boids-Modell [L8].

2.1.1 Agenten

Die Definition eines Agenten ist nicht standardisiert, vielmehr gibt es mehrere verschiedene Definitionen in der Literatur, die aber in vielen Punkten übereinstimmen. Manchen Autoren reicht schon die *Unabhängigkeit* einer Komponente von anderen um sie einen Agenten zu nennen. Andere wiederum finden es notwendig, dass die Komponenten eine gewisse *Anpassungsfähigkeit*, was die wechselnde Umgebungssituation betrifft, vorweisen müssen, um von einem Agenten sprechen zu können. Für einige ist es wichtig, dass die Agenten *autonom* agieren können. Zusammengefasst ergeben sich folgende Eigenschaften der Agenten, die sich in der Praxis als genügend und ausreichend herausgestellt haben um damit vernünftig agentenbasierte Modelle zu erstellen [L22]:

Autonomie Agenten können unabhängig von anderen Agenten agieren.

Modularität Ein Agent kann genau identifiziert werden. Er besteht aus bestimmten Eigenschaften und Verhaltensweisen. Es ist klar, was zu einem Agenten gehört und was nicht.

Interaktivität Ein Agent interagiert mit anderen Agenten. Dazu bedient er sich bestimmten Mechanismen und Protokollen. So können die Agenten Informationen austauschen, kommunizieren und sich auch gegenseitig beeinflussen.

Außer diesen notwendigen Eigenschaften gibt es auch noch andere optionale Eigenschaften. Da Agenten oft reale Individuen oder Organisationen abbilden ist es sinnvoll entsprechende Eigenschaften zu übernehmen.

Position Leben die Agenten in einer Umgebung, so ist ihnen eine Position in dieser zugeordnet. Diese Position und die Positionen anderer Agenten beeinflussen dann unter Anderem ihr Verhalten und ihre Entscheidungen.

Ziele Agenten können Ziele haben. Diese Ziele müssen nicht notwendigerweise etwas mit der Maximierung bestimmter Eigenschaften oder Parameter zu tun haben sondern können auch einfache Ziele sein, die in bestimmter Weise das Verhalten des Agenten bestimmen. Beispiele dafür sind, dass ein Patient gesund werden will oder ein Einkäufer eine bestimmte Ware erwerben will.

Adaption Agenten können aus ihrem vergangenen Verhalten lernen und so ihr zukünftiges Verhalten ändern und anpassen. Dazu benötigen sie einen Speicher, meistens in Form eines dynamischen Attributes, in dem sie Informationen zu vergangenen Zuständen oder Ereignissen hinterlegen. Mit diesen Daten können Sie dann die Regeln, die ihr Verhalten beschreiben, ändern.

Ressourcen Agenten haben oft bestimmte Ressourcen, die sich dynamisch ändern. Dazu gehören z.B. Energie, Gesundheit, Kapital und Information.

2.1.2 Umgebung und Nachbarschaft

Eine wichtige Rolle spielt auch der Begriff der Nachbarschaft, die festlegt welche Agenten miteinander interagieren können und welche nicht. Diese Nachbarschaft hängt stark von der verwendeten Umgebung ab, in der die Agenten situiert sind. Dabei unterscheidet man im Allgemeinen zwischen folgenden fünf Umgebungstypen [L22, S. 93-94]:

Ungeordnet In dieser Umgebung, im Englischen oft auch als „soup“ (Suppe) bezeichnet, können Agenten in beliebiger Kombination miteinander in Kontakt treten.

Raster Die Agenten befinden sich in den Zellen des Rasters und können nur mit benachbarten Zellen interagieren. Dabei sind verschiedene Formen der Nachbarschaft denkbar (z.B. Von-Neumann-Nachbarschaft oder Moore-Nachbarschaft). Als Beispiel seien hier die zellulären Automaten erwähnt.

Euklidischer Raum Den Agenten werden bestimmte Punkte in einem kartesischen Koordinatensystem, meist zwei- oder drei-dimensional, zugeordnet. Nachbarschaften können hier über die Distanz zu anderen Agenten bestimmt werden.

Geographisches Informationssystem (GIS) Die Agenten werden auf einer geographischen Karte positioniert und die Nachbarschaften werden aus den Eigenschaften dieser Karte oder durch die Distanzen der Agenten abgeleitet.

Netzwerk Die Agenten werden den Knoten eines bestimmten Netzwerkes zugeordnet und dieses bestimmt auch wie die Agenten untereinander interagieren können. Beispiele dafür sind Computernetzwerke, soziale Netze, U-Bahn-Netze.

2.2 Diskrete Event Simulation

2.2.1 Kontinuierliche, diskrete und Ereignis-getriebene Zeit

Um den Namen *Diskrete Event Simulation* zu motivieren, sei hier kurz der Unterschied zu Simulationen mit diskreter und kontinuierlicher Zeit beschrieben. Die Abbildungen 2.1–2.3 veranschaulichen diesen anhand der Änderung einer Zustandsvariablen. Während sich bei kontinuierlicher Zeit auch die Zustandsvariablen kontinuierlich ändern können, ist dies bei diskreter Zeit nicht möglich. Unter Simulation mit diskreter Zeit versteht man, dass die Zeitachse diskretisiert wird und nur zu bestimmten Zeitpunkten Änderungen des Systemzustands eintreten können. Üblicherweise verwendet man dazu äquidistante Zeitintervalle, wie zum Beispiel bei zellulären Automaten. Im Fall von nicht äquidistanten Zeitintervallen, wird meistens eine Repräsentation in Form von diskreten Ereignissen bevorzugt. Bei Discrete Event Simulation ist die Zeitachse kontinuierlich, aber in jeder endlichen Zeitspanne können nur endlich viele Ereignisse, die den Systemzustand verändern, auftreten [L5].

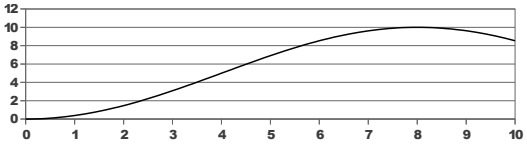

Abbildung 2.1: Änderung einer Zustandsvariablen bei Simulation mit kontinuierlicher Zeit. Auch die Zustandsvariable kann sich hier kontinuierlich ändern.

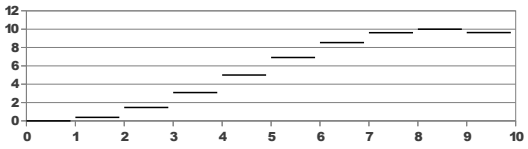

Abbildung 2.2: Änderung einer Zustandsvariablen bei Simulation mit diskreter Zeit. Die Zustandsvariable ändert sich nur zu bestimmten vorgegebenen Zeitpunkten. Diese sind meistens äquidistant angeordnet, wie hier dargestellt, müssen dies aber nicht sein.

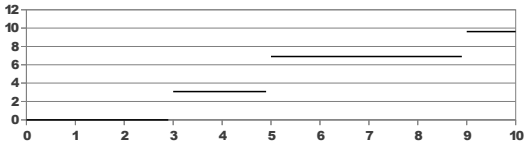

Abbildung 2.3: Änderung einer Zustandsvariablen bei Diskreter Event Simulation. Die Zeitachse ist kontinuierlich, aber in einer endlichen Zeitspanne können nur endlich viele Zustandsänderungen auftreten. In dieser Abbildung ist das zu den Zeitpunkte 3, 5 und 9.

2.2.2 Komponenten des Modells

Eine Diskrete Event Simulation (DES) besteht im Allgemeinen aus folgenden Komponenten [L1]:

Systemzustand Jene Menge an Variablen, die notwendig ist, um den Zustand des Systems zu jedem Zeitpunkt zu beschreiben

Entities Jegliche Objekte oder Komponenten im System, die explizit im Modell repräsentiert werden müssen (z.B. Krankentransporter, Ärzte, Patienten oder Güter).

Attribute Eigenschaften der verschiedenen Entities (z.B. Auslastung eines Krankenhauses, gesundheitlicher Zustand einer Person)

Sets Mengen von (permanent oder temporär) zusammen gehörenden Entities (z.B. Patienten, die auf eine Behandlung warten)

Ereignisse Ereignisse können zu bestimmten Zeitpunkten, periodisch oder

zufällig, auftreten und verändern den Systemzustand. (z.B. Erkrankung einer Person, Aufnahme eines Patienten im Krankenhaus)

Aktivitäten Eine Zeitspanne von bestimmter Länge, die bereits bekannt ist, wenn die Aktivität beginnt (z.B. Anreisezeit, Behandlungsdauer). Die Länge der Zeitspanne kann auch zufällig verteilt sein.

Wartezeit Eine Zeitspanne von unbestimmter Länge (z.B. die Wartezeit eines Patienten bis zur Behandlung)

Sets sind meistens einfache Listen oder Warteschlangen. Eine Aktivität kann deterministisch (z.B. eine Behandlungszeit, die immer 20 Minuten dauert), zufällig in einem bestimmten Intervall (z.B. 15 ± 5 Minuten, gleichverteilt), oder im Allgemeinen jede beliebige mathematische Funktion sein, die die Länge der Aktivität berechnet. Die Dauer einer Aktivität kann aber, im Gegensatz zur Wartezeit, im Vorhinein berechnet werden. Wartezeiten enden wenn bestimmte Bedingungen wahr werden (z.B. endet die Wartezeit für einen Patienten, wenn er an die Spitze der Warteschlange vorgerückt ist).

Die betrachteten Systeme sind im Allgemeinen von dynamischem Charakter. Das heißt, dass sich der Systemzustand, die Attribute, die Anzahl der Entities, die Inhalte der Sets, die laufenden Aktivitäten und Wartezeiten im Lauf der Zeit ändern. Die Zeit selbst wird durch eine Variable, meistens CLOCK genannt, repräsentiert.

2.2.3 Ablauf der Simulation

Ausgehend von einem initialen Systemzustand wird eine Liste von Ereignissen erstellt, deren Zeitpunkt schon zu Beginn der Simulation bekannt ist. Auch für zufällig auftretende Ereignisse wird ein konkreter Zeitpunkt durch Ziehen einer Zufallsvariablen festgelegt. Die Ereignisliste ist nach den Zeiten der Ereignisse sortiert, so dass das nächste auftretende Ereignis immer das erste in der Liste ist. Dieses wird dann während der Simulation ausgeführt, was zu einer Veränderung des Systemzustands führt und eventuell neue Ereignisse zur Ereignisliste hinzufügt. Diese werden wieder entsprechend ihres Ereigniszeitpunktes in die Liste einsortiert. Das aktuelle Ereignis wird aus

der Liste entfernt und es wird mit dem nun nächsten Ereignis in der Liste fortgefahren. Dieser Vorgang wird solange wiederholt bis die Simulation zu Ende ist. Das ist der Fall, wenn entweder keine Ereignisse mehr in der Liste sind oder wenn ein Ereignis eintritt, das explizit die Simulation beendet.

Ein einfaches Beispiel für eine Diskrete Event Simulation ist das Modell eines Supermarktes. Die Entitäten in diesem Modell sind die Kunden, die Einkaufen gehen. Es gibt ein Set an Entities, das die Warteschlange an der Kasse repräsentiert, und folgende Ereignisse:

Ankunft Der Kunde kommt im Supermarkt an und beginnt einzukaufen.

Anstellen Der Kunde stellt sich an der Kasse an und wartet darauf, dass er zum Bezahlen dran kommt.

Bezahlen Der Kunde bezahlt seine Waren und verlässt den Supermarkt.

Das Ereignis *Ankunft* tritt mit einer bestimmten Rate auf. Bei Eintritt dieses Ereignisses wird festgelegt, wie lange der Kunde einkaufen wird und ein entsprechendes Ereignis *Anstellen* in die Ereignisliste eingetragen. Zum Zeitpunkt dieses Ereignisses wird der Kunde zur Kassa-Warteschlange hinzugefügt und falls diese leer war ein Ereignis *Bezahlen* mit einer kurzen Verzögerung, die dem Einscannen der Waren entspricht, erzeugt. Bezahlt der Kunde schließlich seine Waren, wird er aus der Schlange entfernt und es wird wiederum ein neues *Bezahlen*-Ereignis für den nächsten Kunden in der Schlange erzeugt, solange diese nicht leer ist.

2.2.4 Discrete Event System Specification

Die *Discrete Event System Specification* (DEVS) ist ein Formalismus zur Beschreibung eines Discrete Event Models M. Dieser Formalismus kann sowohl zur mathematischen Analyse verwendet werden als auch als Grundlage zur Entwicklung einer effektiven Modellierungs- und Simulationsmethode dienen.

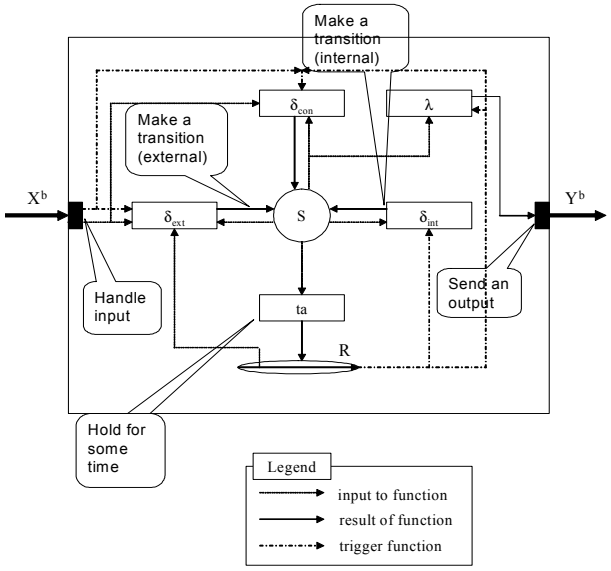

Abbildung 2.4: Arbeitsweise des DEVS Formalismus [L37]

Zeigler [L37] hat diesen Formalismus so beschrieben:

$$M = \langle X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta \rangle$$

wobei

X ist die Menge der Eingabewerte

S ist die Menge der Zustände

Y ist die Menge der Ausgabewerte

$\delta_{int} : S \to S$ ist die *interne Übergangsfunktion*

$\delta_{ext} : Q \times X^b \to S$ ist die *externe Übergangsfunktion*, wobei

$Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$ ist die *totale Zustandsmenge*

e ist die *abgelaufene Zeit* seit dem letzten Übergang

X^b ist die Sammlung von X-Gruppen (Mengen, in denen einige Elemente öfter als einmal vorkommen können).

$\delta_{con} : Q \times X^b \to S$ ist die *vereinigende Übergangsfunktion*,

$\lambda : S \to Y^b$ ist die Ausgabefunktion

$ta : S \to \mathbb{R}^+_{0,\infty}$ ist die Zeitfortschrittsfunktion

Die Interpretation dieser Elemente wird in Abbildung 2.4 dargestellt.

Zu jeder Zeit befindet sich das System in einem Zustand s. Falls kein externes Ereignis eintritt, bleibt es im Zustand s für eine Zeitspanne $ta(s)$. Man beachte, dass $ta(s)$ eine reelle Zahl sein kann, wie man es erwarten würde. Aber es kann auch 0 oder ∞ sein. Im ersten Fall ist die Verweildauer im Zustand s so kurz, dass kein externes Ereignis auftreten kann — s ist ein flüchtiger Zustand. Im zweiten Fall bleibt das System für immer im Zustand s falls kein externes Ereignis eintritt — s ist ein passiver Zustand. Wenn die Wartezeit ausläuft, das heißt, wenn die abgelaufene Zeit $e = ta(s)$ ist, gibt das System den Wert $\lambda(s)$ aus und wechselt in den Zustand $\delta_{int}(s)$. Ausgaben sind also nur vor internen Übergängen möglich.

Falls ein externes Ereignis x aus X^b eintritt bevor die Wartezeit ausläuft, d.h., wenn das System im totalen Zustand (s, e) mit $e \leq ta(s)$ ist, wechselt das System in den Zustand $\delta_{ext}(s, e, x)$. So gibt die interne Übergangsfunktion den neuen Zustand des Systems vor, wenn keine Ereignisse seit dem letzten Übergang aufgetreten sind. Während die externe Übergangsfunktion den neuen Systemzustand vorgibt, wenn ein externes Ereignis auftritt — dieser Zustand wird aus dem Input x, dem aktuellen Zustand s und der verstrichenen Zeit e bestimmt, wenn das externe Ereignis eintritt. In beiden Fällen befindet sich das System danach in einem neuen Zustand s' mit einer neuen Wartezeit $ta(s')$ und der Vorgang beginnt von neuem.

Man beachte, dass ein externes Ereignis x aus X^b eine Gruppe von Elementen aus X ist. Das bedeutet, dass ein oder mehrere Elemente zur gleichen Zeit am Eingabeport erscheinen können. Diese Eigenschaft wird benötigt, da DEVS erlaubt, dass mehrere Komponenten zur gleichen Zeit Ausgaben erzeugen und diese an Eingabeports senden.

2.3 Vergleich der Modellierungsmethoden

Ein Vergleich von DES und agentenbasierter Modellierung geschieht am Besten durch einen Vergleich von Entities und Agenten, die in beiden Modellierungsmethoden ähnliche Aufgaben erfüllen. Wenn im Folgenden von Objekten gesprochen wird dann sind damit Entities und Agenten in den entsprechenden Modellen gemeint.

Ein großer Unterschied wird bei Betrachtung des *Lebensraumes* der Objekte sichtbar. Während sich Entities durch einen Prozessgraphen bewegen, können Agenten beliebige Positionen in diskreten oder kontinuierlichen Räumen oder sogar in einem GIS annehmen. Mit der Bewegung durch einen Prozessgraphen ist gemeint, dass die Entities von einem Knoten (z.B. Aufenthalt im Supermarkt) zu einem anderen (z.B. Kassa-Warteschlange) wechseln ohne dabei einen räumlichen Bezug zum Supermarkt haben. Bei agentenbasierter Modellierung ist es möglich den Supermarkt als zweidimensionalen Raum zu modellieren, in dem sich die Agenten an jeder beliebigen Position aufhalten können. Um zur Kasse zu gelangen müssen sie sich also wirklich auf einem Weg durch den Supermarkt zur Kasse bewegen.

Sieht man die Position der Objekte als ein Attribut an, so kann man einen zusätzlichen Unterschied festestellen. Während bei agentenbasierter Modellierung eine *kontinuierliche Änderung* dieser Attribute möglich ist, können sich diese bei DES nur zu diskreten Zeitpunkten durch Ereignisse ändern. So kann sich die Position von Agenten im Raum kontinuierlich ändern, aber auch andere Eigenschaften sind denkbar (z.B. die Gesundheit eines Patienten). Bei agentenbasierter Modellierung ist dies deshalb möglich, da in keiner der gefundenen Definitionen die Simulationszeit explizit als diskret, kontinuierlich oder durch diskrete Events gesteuert festgelegt ist.

Ein weiterer Unterschied zwischen Entities und Agenten ist unter dem Aspekt der *Autonomie* feststellbar. Entities werden durch das Modell durchgereicht und bearbeitet, haben aber keine Möglichkeit eigenständig Aktionen auszuführen. Agenten hingegen können aus ihrem eigenen Antrieb heraus Entscheidungen treffen und Handlungen setzen. Es ist ihnen sogar möglich ihr Verhalten mit der Zeit zu verändern und gegebenenfalls neuen externen

Tabelle 2.1: Unterschiede zwischen Entities und Agenten

	Entities	Agenten
Position	diskret	diskret/kontinuierlich
Attribute	diskret	diskret/kontinuierlich
Autonomie	nein	ja
Interaktion zwischen Entities/Agenten	nein	ja
Adaptives Verhalten	nein	ja

Einflüssen besser anzupassen oder mit anderen Agenten zu interagieren.

Um den Vergleich abzuschließen sei hier noch gesagt, dass beide Modellierungsmethoden mit diskreten Ereignissen arbeiten, bei AB-Modellierung aber auch eine kontinuierliche Veränderung von Systemvariablen möglich ist. Diskrete Event Simulation eignet sich besser in Fällen, in denen bestimmte Entities durch einen Prozess erzeugt, bearbeitet, verändert oder behandelt werden. Agentenbasierte Modellierung ist hingegen für Modelle, in denen reale Personen, Tiere oder Organisationen modelliert werden sollen, die eigene Entscheidungen treffen und selbständig Handlungen ausführen können, die geeignetere Wahl. In Tabelle 2.1 werden die Unterschiede von Entities und Agenten noch einmal zusammengefasst.

2.4 Begründung der Methodenwahl

Als Modellierungsmethode wurde der agentenbasierter Ansatz gewählt, wobei einzelne Agenten Aggregationen von Patienten bzw. medizinische Versorgungseinrichtungen darstellen. Für die Wahl dieser Methode waren vor allem folgende Punkte ausschlaggebend:

- Patienten, Ärzte und Krankenhäuser können auf natürliche Weise als Agenten betrachtet werden.
- Das Verhalten der Patienten und deren Entscheidung, welche medizinischen Versorgungseinrichtungen sie in Anspruch nehmen, kann aus Beobachtungen abgeleitet werden.

- Es ist wichtig, dass die Patienten ihr Verhalten während der Simulation ändern, je nachdem ob sie mit ihrere Behandlung zufrieden sind, oder nicht. Die Adaptivität von Agenten bietet genau diese Möglichkeit.
- Patienten stehen in einer dynamischen Beziehung zu medizinischen Versorgungseinrichtungen.
- Patienten, Ärzte und Krankenhäuser haben eine räumliche Komponente und diese beeinflusst auch ihr Verhalten und die Entscheidungen, die sie treffen.
- Es ist wichtig, dass das Modell skalierbar ist. Es soll möglich sein, sowohl ganz Österreich, als auch nur Teilgebiete zu betrachten.

Kapitel 3

Geographische Informationssysteme

Für den Begriff *Geographisches Informationssystem*, im Folgenden kurz GIS genannt, können verschiedene Definitionen gegeben werden, zum Beispiel:

> Ein *Geoinformationssystem* dient der Erfassung, Speicherung, Analyse und Darstellung aller Daten, die einen Teil der Erdoberfläche und die darauf befindlichen technischen und administrativen Einrichtungen sowie geowissenschaftliche, ökonomische und ökologische Gegebenheiten beschreiben. [L3]

Weitere ähnliche Definitionen finden sich bei Bartelme [L2, S.15f]. Bekannte Beispiele für GI-Systeme sind OpenStreetMap [W20], Google Maps [W13], Bing Maps [W15] oder der Stadtplan Wien [W27]. Allen diesen GI-Systemen gemein ist, dass sie die geographischen Daten in einer Datenbank speichern und über das Internet eine Schnittstelle anbieten, mit der die Daten betrachtet und eventuell auch verändert werden können. Diese Schnittstelle kann eine Webseite sein oder eine Programmierschnittstelle (API) die mit Desktop-Anwendungen genutzt werden kann. Außerdem können meistens auch Abfragen zu Orten, Adressen oder anderen geographischen Objekten ausgeführt werden.

3.1 Daten

Man muss laut Bartelme [L2] zwischen verschiedenen Datentypen unterscheiden:

Rohdaten Direkt von Sensoren oder Messgeräten (GPS) erfasst, sowie fotografische Daten, wie Luftbildaufnahmen oder Scans von herkömmlichen gedruckten Karten

Interpretierte Daten Bereits durch zusätzliches Wissen klassifiziert, mit Bedeutungen belegt und ergänzt

Symbolisierte Daten Ergänzung und Zuweisung von Symbolen, Farben und Formen nach kartographischen Konventionen

Strukturierte Daten Zusammenfassung zu höheren Strukturen nach anwendungsorientierten Aspekten oder durch Hintergrundwissen

Strukturierte Daten können dann wiederum verschiedene Eigenschaften haben:

- Strukturelle Eigenschaften
- Geometrische (metrische und topologische) Eigenschaften
- Thematische Eigenschaften

Strukturelle Eigenschaften beschreiben aus welchen (atomaren) Teilen ein Objekt zusammengesetzt ist. Zum Beispiel besteht eine Straße aus mehreren Straßenabschnitten und dazwischen liegenden Kreuzungen. Die Straßenabschnitte sind wiederum aus verschiedenen Liniensegmenten zusammengesetzt.

Die *Geometrie* äußert sich im Raumbezug. So haben alle Objekte eine bestimmte Lage und Ausdehnung: Bei Häusern, Straßen, Wäldern, oder Ländern ist das offensichtlich. Aber auch Fahrverbote, Eigentümerverhätnisse und Öffnungszeiten haben einen gewissen - wenn auch schwächeren - Raumbezug, da sie sich auf bestimmte abgegrenzte geometrische Objekte beziehen. Neben weiteren metrischen Eigenschaften, wie Fläche,

Umfang, Höhenangaben und Winkel, gibt es auch noch topologische Eigenschaften. Diese charakterisieren die Beziehungen der Objekte untereinander. Das Enthaltensein von Objekten, Nachbarschaften oder Überschneidungen zählen dazu. Eine Stadt kann zum Beispiel in einem Land enthalten sein, ein Park an eine Straße grenzen oder eine Brücke über einen Fluß führen. Wenn man nach diesen Beziehungen fragt - liegt die Stadt in diesem Land oder grenzt der Park an diese Straße - so gibt es immer eine eindeutige Antwort. Es gibt aber auch Beziehungen bzw. Fragen, die nicht eindeutig beantwortet werden können. Diese Fragestellungen haben meistens mit der *Nähe* von Objekten zu tun: Liegt Ort A in der Nähe von Ort B? Diese Beziehungen nennt man *unscharf (fuzzy)*.

Jedes Objekt kann auch einem oder mehreren *Themen* zugeordnet werden. Mögliche Themenbereiche wären z.B. [L2]:

Topographie Gebirge, Talsenken, Gewässer

Grundstückseigentum Kataster, Grundbuch

Klima Temperatur, Niederschlag, Besonnung

Natürliche Ressourcen Hydrologie (Wasserhaushalt), Geologie (Lagerstätten)

Ökologische Themen Boden-, Gewässer-, und Luftgüte

Technische Einrichtungen Verkehrslagen, Projekte, Ver- und Entsorgungsnetze

Nutzung Flächennutzung, Infrastruktur

Daten der Wirtschaft Bodenertrag, Waldflächen, Viehbestand, Betriebsstruktur, Fremdenverkehr

Sozioökonomische Daten Statistische Angaben

Bei der Verwaltung der Daten kann zwischen verschiedenen Konzepten unterschieden werden:

- Entity-Relationship-Konzept
- Layerkonzept
- Objektorientiertes Konzept
- Feldbasierendes Konzept

Entity-Relationship-Konzept Dieses Konzept basiert auf der Definition von Entitäten, also kleinsten Einheiten des geographischen Datenmaterials. Diese werden nach ihren Eigenschaften in Klassen zusammengefasst um Übersichtlichkeit und Verwaltbarkeit zu gewährleisten. Beispiele solcher Klassen sind Punkte, Linien und Flächen. Den einzelnen Entitäten werden verschiedene Attribute zugewiesen um sie z.B. als Telefonzellen, Straßenabschnitte oder Wälder zu kennzeichnen. Diese Entitäten stehen in Beziehungen (Relationen) zueinander und können damit größere Objekte beschreiben: Mehrere Straßenabschnitte können zu einer ganzen Straße zusammengefasst werden, oder aber auch zu dem Verlauf einer Buslinie. Ebenso ist es denkbar die Einrichtungen eines Flughafens, wie Start- und Landebahnen, Terminals, Restaurants, Parkhäuser, etc. in einer Beziehung „Flughafen“ zusammenzufassen. Die so entstehenden Relationen können wieder mit Attributen versehen und auch in anderen Relationen wiederverwendet werden.

Layerkonzept Dieses Konzept fasst Entitäten nicht nach ihren geometrischen Eigenschaften, sondern nach dem Themenbereich, dem sie zugeordnet werden können, zusammen. Beispiele für Layer sind Verkehrsnetz, Gewässer und Landnutzung. Diese Layer kann man dann wie Overhead-Folien übereinander legen und sich so die einzelnen Themen die für die Anwendung relevant sind zusammenstellen.

Objektorientierte Konzept Dieses Konzept ähnelt stark dem Entity-Relationship-Konzept, erweitert dieses jedoch um die Eigenschaften von

Klassen, wie sie aus der objektorientierten Programmierung bekannt sind: Datenkapselung, Vererbung und Methoden. Der Vorteil hier ist, dass man z.B. eine Methode zur Flächenberechnung in der Polygon-Klasse einmal implementiert und schon lässt sich der Flächeninhalt von beliebigen Objekten berechnen, die durch ein Polygon beschrieben werden: Häuser, Parks, Seen, etc.

Feldbasiertes Konzept Bei den bisher vorgestellten Konzepten handelt es sich bei den Daten immer um geometrische Objekte mit bestimmten Eigenschaften. Es ist aber auch möglich eine gewisse Eigenschaft zu betrachten und diese an verschiedenen Punkten oder Teilgebieten einer Region auszuwerten. Dann spricht man vom *feldbasierten Konzept*. Gebiete oder Felder können beliebige Unterteilungen der Karte sein, zum Beispiel: Bundesländer, Bezirke, Gemeinden, Zählsprengel, NUTS[1]-Einheiten. Findet die Auswertung in einem (rechteckigen) Raster statt gelangt man zu einem *Rastermodell*. Beispiele für Daten, die auf diese Weise erfasst werden können sind: Niederschlag, Population und Geburtenrate, um nur einige zu nennen.

Kombination von Konzepten Konkrete GI-Systeme verwenden im Regelfall eine Kombination dieser Konzepte um für die jeweilige Anwendungssituation die entsprechenden Vorteile nutzen zu können. So eignet sich das Layerkonzept sehr gut zur Präsentation der Daten. Der Benutzer kann sich damit seine individuelle Karte aus verschiedenen Schichten (Layers) zusammenstellen. Das Entity-Relationship-Konzept hingegen eignet sich gut zur effizienten Speicherung der Daten.

3.2 Koordinaten-Bezugssysteme

Für Geodaten muss ein gewisser *Raumbezug* hergestellt werden um die Lage, Ausdehnung und Entfernung von Objekten eindeutig angeben zu können. Dazu benötigt man zuerst ein Modell für die Erdoberfläche.

[1] Nomenclature des unités territoriales statistiques

Folgende Referenzflächen sind die wichtigsten [L2]:

- Ebene
- Kugel oder Rotationsellipsoid (mathematische Definition)
- Geoid (physikalische Definition)

Die *Ebene* ist der einfachste Fall, kann jedoch auf Grund der Erdkrümmung nur auf sehr eingeschränkte lokale Regionen angewendet werden. Außerdem muss auch bei diesen Projektionen der Erdoberfläche immer angegeben werden, aus welchem Erdmodell sie hervorgegangen sind und welche Projektion verwendet wurde.

Die *Kugel* oder das *Rotationsellipsoid* sind mathematisch definierte Modelle, die global eingesetzt werden können. Für die Kugel lautet die zugehörige mathematische Bedingung z.B. $x^2 + y^2 + z^2 = r^2$.

Das *Geoid* ist ein physikalisch definiertes Modell. Dabei denkt man sich alle Punkte mit dem selben Potential auf einer gemeinsamen *Äquipotentialfläche*. Das Geoid ist dann jene Äquipotentialfläche, die der mittleren Oberfläche der ruhenden Weltmeere entspricht. Eine exakte Bestimmung des Geoids ist nicht möglich, da dafür unzählige Messungen auf und unter der Erdoberfläche notwendig wären. Daher muss man sich wieder auf mathematische Modelle beschränken die das Geoid approximieren.

Damit Koordinaten eine Bedeutung erlangen, benötigen sie ein Bezugssystem. Für so ein koordinatives Bezugssystem sind zwei Angaben notwendig:

- Datum
- Koordinatensystem (kartesisch oder geodätisch)

Das *Datum* in dieser Verwendung hat nichts mit dem Kalenderdatum zu tun sondern beschreibt das verwendete Ellipsoid, das als Modell der Erde verwendet wird. An Hand dieses Ellipsoids kann dann ein Koordinatensystem definiert werden. Die Äquatorebene stimmt dabei immer mit der Rotationsebene des Ellipsoids überein. Grundsätzlich unterscheidet man zwischen kartesischen und geodätischen Koordinatensystemen. Für ein kartesisches Koordinatensystem müssen der Ursprung sowie die Achsen für die

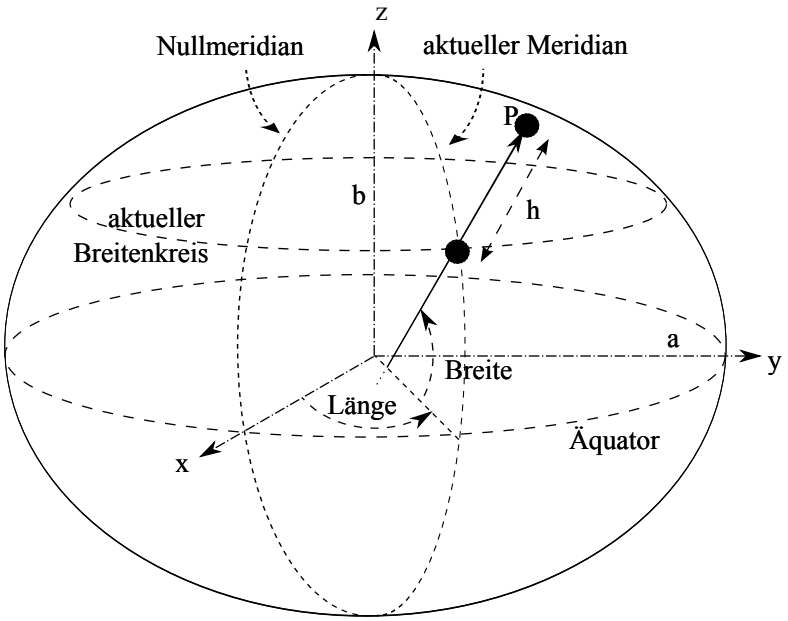

Abbildung 3.1: Geodätische Länge und Breite, ellipsoidische Höhe h [L2]

Koordinaten x, y und z angegeben werden. In einem geodätischen Koordinatensysten werden Punkte durch die Angabe der *geodätischen Breite* φ, der *geodätischen Länge* λ und der *ellipsoidischen Höhe* h angegeben (Abb. 3.1). Der Vollständigkeit halber sei hier auch noch die *orthometrische Höhe* H erwähnt, die man theoretisch erhält, indem man die Wegstrecke misst, die zurückgelegt wird, wenn man vom aktuellen Punkt P entlang der Lotlinie bis zum Geoid eindringt [L2]. Wichtig ist, dass die geodätische Breite eines Punktes P den Winkel zwischen der Normalen auf das Ellipsoid durch den Punkt P und der Äquatorebene misst und nicht den Winkel zwischen dem Fußpunkt Q, dem Mittelpunkt des Ellipsoids und der Äquatorebene (Abb. 3.2). Dieser wird geozentrische Breite ψ genannt. Da die geodätische Breite immer in der Äquatorebene den Wert Null annimmt, ist es ausreichend, den Nullmeridian, also jenen Meridian an dem die geodätische Länge den Wert Null annimmt, anzugeben um das Koordinatensystem festzulegen. Die geodätische Breite und geodätische Länge werden auch oft *geographische Breite* und *geographische Länge* genannt. Für beide Arten von Koordinatensystemen muss auch noch die verwendete Maßeinheit festgelegt werden.

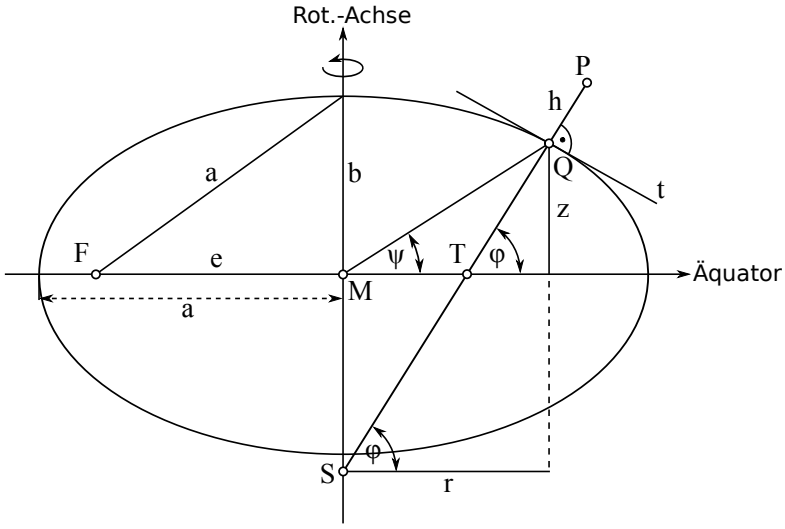

Abbildung 3.2: Umrechnung zwischen geodätischen und kartesischen Koordinaten

F: Brennpunkt der Ellipse
M: Mittelpunkt der Ellipse
P: Punkt auf der Erdoberfläche
Q: Fußpunkt der Normalen auf der Ellipse durch den Punkt P
S: Schnittpunkt der Normalen im Punkt Q mit der Rotationsachse
T: Schnittpunkt der Normalen im Punkt Q mit der Äquatorebene
φ: geodätische Breite
ψ: geozentrische Breite
a: Halbe Hauptachse der Ellipse
b: Halbe Nebenachse der Ellipse
e: Lineare Exzentrizität
h: Ellipsoidische Höhe des Punktes P
r: Abstand von Punkt Q zur Rotationsachse
t: Tangente im Punkt Q
z: z-Koordinate des Punktes Q

Um einen Einblick in den mathematischen Hintergrund von geographischen Koordinatensystemen zu geben, sei hier der Beweis für die Formel 3.1 zur Umrechnung von geodätischen Koordinaten in kartesische Koordinaten gegeben:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} (N+h)\cos\varphi\cos\lambda \\ (N+h)\cos\varphi\sin\lambda \\ (N\,(1-\varepsilon^2)+h)\sin\varphi \end{pmatrix} \tag{3.1}$$

Die Annahmen dafür sind, dass der Ursprung des kartesischen Koordinatensystems mit dem Mittelpunkt des Ellipsoids zusammenfällt und die x-Achse das Ellipsoid im Punkt $\varphi = 0, \lambda = 0$ durchstößt. Zuerst sei hier die allgemeine Gleichung für ein Rotationsellipsoid gegeben:

$$\frac{x^2}{a^2} + \frac{y^2}{a^2} + \frac{z^2}{b^2} = 1 \tag{3.2}$$

Durch die Substitution

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} r\cos\lambda \\ r\sin\lambda \end{pmatrix} \tag{3.3}$$

erhält man die Gleichung der Ellipse, aus der das Ellipsoid durch Rotation entsteht:

$$\frac{r^2}{a^2} + \frac{z^2}{b^2} = 1 \tag{3.4}$$

Abb. 3.2 zeigt diesen senkrechten Schnitt durch das Ellipsoid. Die Koordinaten des Punktes Q können nun folgendermaßen angeschrieben werden:

$$Q = \begin{pmatrix} r \\ z \end{pmatrix} = \begin{pmatrix} \overline{QS}\cos\varphi \\ \overline{QT}\sin\varphi \end{pmatrix} \tag{3.5}$$

Unter Berücksichtigung der Höhe h des Punktes P erhält man so auch dessen Koordinaten:

$$P = \begin{pmatrix} \left(\overline{QS}+h\right)\cos\varphi \\ \left(\overline{QT}+h\right)\sin\varphi \end{pmatrix} \tag{3.6}$$

Die Strecken $\overline{QS}$ und $\overline{QT}$ ergeben sich aus dem Schnitt des Normalvektors im Punkt P auf das Ellipsoid mit der Rotationsachse und der Äquatorebene.

Der Normalvektor ist der Gradient von f (3.4). Normiert sieht er so aus:

$$n_0 = \begin{pmatrix} \frac{r}{a^2} \\ \frac{z}{b^2} \end{pmatrix} \left(\frac{r^2}{a^4} + \frac{z^2}{b^4} \right)^{-\frac{1}{2}} \tag{3.7}$$

Mit diesem Normalvektor läßt sich nun auch die Strecke $\overline{QS}$ bestimmen:

$$\begin{aligned} S &= Q - \overline{QS} \cdot n_0 \\ \text{1. Koordinate: } 0 &= r - \overline{QS} \frac{r}{a^2} \left(\frac{r^2}{a^4} + \frac{z^2}{b^4} \right)^{-\frac{1}{2}} \\ \overline{QS} &= a^2 \sqrt{\frac{r^2}{a^4} + \frac{z^2}{b^4}} \end{aligned} \tag{3.8}$$

Äquivalent dazu für $\overline{QT}$:

$$\begin{aligned} T &= Q - \overline{QT} \cdot n_0 \\ \text{2. Koordinate: } 0 &= z - \overline{QT} \frac{z}{b^2} \left(\frac{r^2}{a^4} + \frac{z^2}{b^4} \right)^{-\frac{1}{2}} \\ \overline{QT} &= b^2 \sqrt{\frac{r^2}{a^4} + \frac{z^2}{b^4}} \end{aligned} \tag{3.9}$$

Mit $\overline{QS}$ und $\overline{QT}$ kann Q (3.5) nun so ausgedrückt werden:

$$Q = \begin{pmatrix} a^2 cos\varphi \\ b^2 sin\varphi \end{pmatrix} \sqrt{\frac{r^2}{a^4} + \frac{z^2}{b^4}} \tag{3.10}$$

Diese Darstellung von Q enthält immer noch die Variablen r und z. Da Q aber allein durch φ ausgedrückt werden soll, müssen diese noch eliminiert werden. Dazu setzt man Q in die Ellipsengleichung (3.4) ein, da Q diese erfüllen muss

$$a^2 \left(\frac{r^2}{a^4} + \frac{z^2}{b^4} \right) cos^2\varphi + b^2 \left(\frac{r^2}{a^4} + \frac{z^2}{b^4} \right) sin^2\varphi = 1 \tag{3.11}$$

und erhält somit folgenden Ausdruck:

$$\frac{r^2}{a^4} + \frac{z^2}{b^4} = (a^2 cos^2\varphi + b^2 sin^2\varphi)^{-1} \tag{3.12}$$

Die rechte Seite kann mit Hilfe der numerischen Exzentrizität

$$\varepsilon = \frac{e}{a} = \frac{\sqrt{a^2 - b^2}}{a} \tag{3.13}$$

noch weiter vereinfacht werden:

$$\begin{aligned} \left(a^2 cos^2\varphi + b^2 sin^2\varphi\right)^{-1} &= \\ \left(a^2 - a^2 \sin^2\varphi + b^2 \sin^2\varphi\right)^{-1} &= \\ \left(a^2 - (a^2 - b^2) \sin^2\varphi\right)^{-1} &= \\ \left(a^2 \left(1 - \varepsilon^2 \sin^2\varphi\right)\right)^{-1} & \end{aligned} \tag{3.14}$$

Damit kann Q nun unabhängig von p und z ausgedrückt werden:

$$Q = \begin{pmatrix} a^2 \cos\varphi \\ b^2 \sin\varphi \end{pmatrix} \left(a^2 \left(1 - \varepsilon^2 \sin^2\varphi\right)\right)^{-\frac{1}{2}} \tag{3.15}$$

Definiert man

$$N = a(1 - \varepsilon^2 \sin^2\varphi)^{-\frac{1}{2}} \tag{3.16}$$

so kann man Q auch so schreiben:

$$Q = \begin{pmatrix} N \cos\varphi \\ N\,(1 - \varepsilon^2) \sin\varphi \end{pmatrix} \tag{3.17}$$

Da P auf derselben geodätischen Breite φ liegt und sich nur durch einen um h längeren „Radius" von Q unterscheidet ergibt sich für P wie in Gleichung 3.6:

$$P = \begin{pmatrix} (N + h) \cos\varphi \\ (N\,(1 - \varepsilon^2) + h) \sin\varphi \end{pmatrix} \tag{3.18}$$

Rücksubstitution (3.3) ergibt schließlich die Formel zur Umrechnung von

geodätischen in kartesische Koordinaten:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} (N+h)\cos\varphi\cos\lambda \\ (N+h)\cos\varphi\sin\lambda \\ (N\,(1-\varepsilon^2)+h)\sin\varphi \end{pmatrix} \tag{3.19}$$

□

Koordinaten-Bezugssysteme werden oft im Format WKT (Well-Known-Text) angegeben. Dieses Format wurde vom Open Geospatial Consortium (OGC) spezifiziert [L24, S.18-23]. Listing 3.1 zeigt wie das Bezugssystem „World Geodetic System 1984 (WGS 84)" in diesem Format aussieht. Aus

```
GEOGCS["GCS_WGS_1984",
    DATUM["D_WGS_1984",
        SPHEROID["WGS_1984",6378137,298.257223563]],
    PRIMEM["Greenwich",0],
    UNIT["Degree",0.0174532925199433295]]
```

Listing 3.1: World Geodetic System 1984 (WGS 84) im WKT-Format

dieser Beschreibung ist abzulesen, dass es sich um ein geographisches (GEOGCS) im Gegensatz zu einem kartesischen (GEOCCS) Koordinatensystem handelt. Das Erdmodell ist ein Ellipsoid mit einer großen Halbachse von 6378137 Metern. Die inverse Abplattung $\frac{1}{f} = \frac{a}{a-b}$ beträgt 298.2572 . . ., wobei a und b die halbe Haupt- und Nebenachse sind. Weiters entspricht der Nullmeridian dem üblicherweise angenommenen durch Greenwich verlaufenden Meridian, weicht also um 0 Grad von diesem ab. Koordinatenangaben werden in einer Einheit gemessen, die sich mit dem Faktor 0.0174. . . in Radiant umrechnen lässt. Dies entspricht genau dem Faktor $\frac{2\pi}{360}$. Damit handelt es sich um Grad, wie auch durch den Bezeichner „Degree" angedeutet wird.

Oft beziehen sich die Koordinaten eines geographischen Objektes nicht direkt auf das Koordinatensystem des Erdellipsoids, sondern nur auf seine *Projektion* in die Ebene. Dann spricht man auch von einem projezierten Koordinatensystem. Ein Beispiel hierfür ist die in Österreich oft verwendete MGI-Lambert Projektion, die im WKT-Format wie in Listing 3.2 definiert

```
PROJCS["MGI_Austria_Lambert",
    GEOGCS["GCS_MGI",
        DATUM["D_MGI",
            SPHEROID["Bessel_1841",6377397.155,299.1528128]],
        PRIMEM["Greenwich",0.0],
        UNIT["Degree",0.0174532925199433]],
    PROJECTION["Lambert_Conformal_Conic"],
    PARAMETER["False_Easting",400000.0],
    PARAMETER["False_Northing",400000.0],
    PARAMETER["Central_Meridian",13.33333333333333],
    PARAMETER["Standard_Parallel_1",46.0],
    PARAMETER["Standard_Parallel_2",49.0],
    PARAMETER["Latitude_Of_Origin",47.5],
    UNIT["Meter",1.0]]
```

Listing 3.2: MGI-Lambert Projektion im WKT-Format

wird. Dass es sich hierbei um ein projeziertes Koordinatensystem handelt, erkennt man an dem einleitenden Wort „PROJCS". Für die Projektion wichtig ist natürlich das Ellipsoid das projeziert wird, die Art der Projektion und projektionsspezifische Parameter.

Das OGP Geomatics Committee [W5] verwaltet und veröffentlicht eine Datenbank an Koordinatenbezugssystemen. Jedes Bezugssystem in dieser Datenbank wird durch einen EPSG-Code identifiziert. EPSG steht für „European Petroleum Survey Group", welcher das Komitee ursprünglich unter dem Namen „Surveying & Positioning Committee" angehörte. Nach der Übernahme der EPSG durch die „International Association of Oil & Gas producers" [W16], wurde auch das Komitee umbenannt [W5]. Die Datenbank kann auf der Webseite des Geomatics Committee [W5] heruntergeladen werden. Das Komitee stellt auch eine Webseite [W4] zur Verfügung, auf der direkt nach Koordinatenbezugssystemen und deren Definition gesucht werden kann. Tabelle 3.1 listet einige Einträge dieser Datenbank auf.

Die Statistik Austria bietet ihre Daten in Bezug auf das MGI- und das ETRS89-Raster an. Diese Raster werden durch die X- und Y-Achsen der entsprechenden Koordinatensysteme aufgespannt und sind in verschie-

Tabelle 3.1: Koordinatenreferenzsysteme aus der EPSG-Datenbank

Name	Code	Typ
WGS 84[1]	EPSG::4326	GeodeticCRS (geographic 2D)
WGS 84[1] / Pseudo-Mercator	EPSG::3857	ProjectedCRS
WGS 84[1] / World Mercator	EPSG::3395	ProjectedCRS
MGI[2]	EPSG::4312	GeodeticCRS (geographic 2D)
MGI[2] / Austria Lambert[3]	EPSG::31287	ProjectedCRS
ETRS89[4]	EPSG::4258	GeodeticCRS (geographic2D)
ETRS89[4] / LAEA[5] Europe	EPSG::3035	ProjectedCRS

[1] World Geodetic System 1984
[2] Militärgeographisches Institut
[3] Lambertsche Schnittkegelprojektion (Lambert Conic Conformal Projection) mit zwei für Österreich spezifischen Standardparallelen (49 °N, 46 °E)
[4] Europäisches Terrestrisches Referenzsystem 1989
[5] Flächentreue Azimutalprojektion (Lambert Azimuthal Equal Area Projection)

denen Auflösungen verfügbar. Sie eignen sich sehr gut, um die Unterschiede zwischen Koordinatensystemen und deren Projektionen zu veranschaulichen (Abb. 3.3–3.6).

3.3 Datenformate

Der Austausch von Geodaten kann in verschiedensten Datenformaten stattfinden von denen hier nur ein paar erwähnt seien:

Geography Markup Language (GML) XML-basierter offener Standard, der von OpenGIS entwickelt wurde

Keyhole Markup Language (KML) XML-basierter offener Standard, der von OpenGIS entwickelt wurde

Shapefile Von ESRI entwickeltes Vektordaten-Format, das SHP-, SHX- und DBF-Dateien verwendet

TIGER Topologically Integrated Geographic Encoding and Referencing ist das Format das vom United States Census Bureau verwendet wird

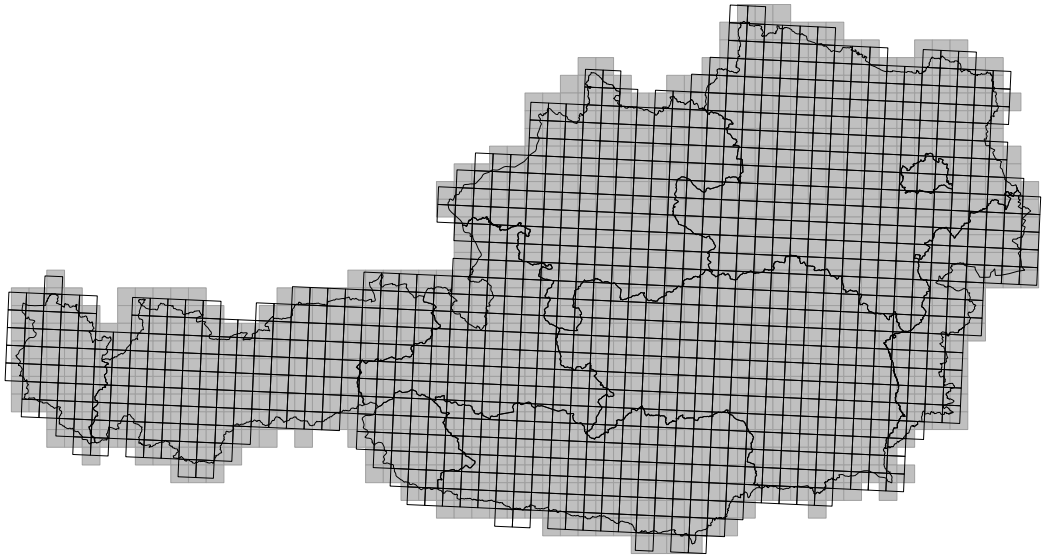

Abbildung 3.3: Das ETRS89-Raster wird hier in Bezug zum MGI-Raster (graue Fläche) dargestellt. In der verwendeten Projektion MGI / Austria Lambert (EPSG::31287) hat letzteres waagrechte und senkrechte Achsen.

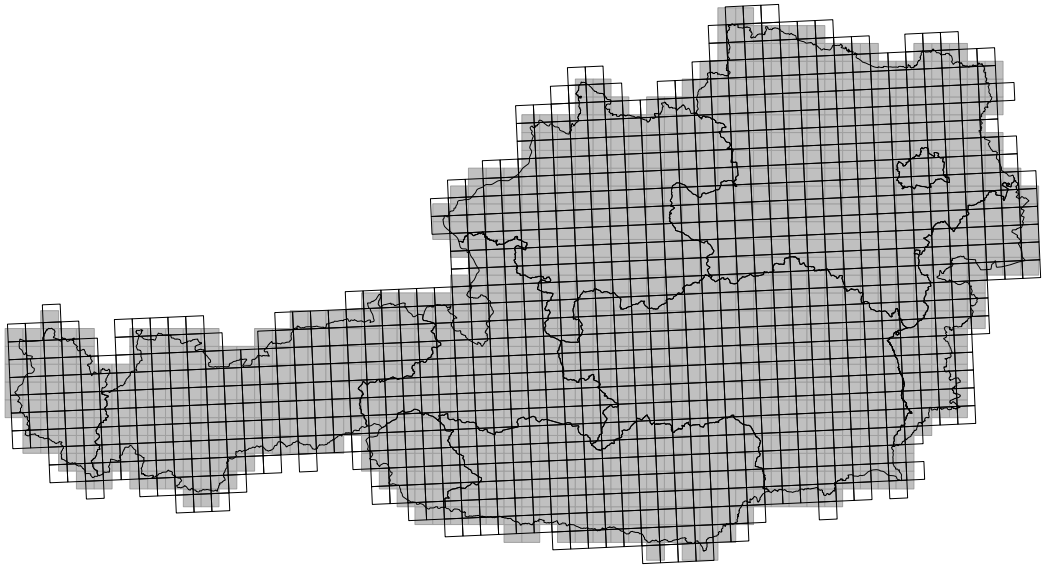

Abbildung 3.4: Das MGI-Raster wird hier in Bezug zum ETRS89-Raster (graue Fläche) dargestellt. In der verwendeten Projektion ETRS89 / LAEA (EPSG::3035) hat letzteres waagrechte und senkrechte Achsen.

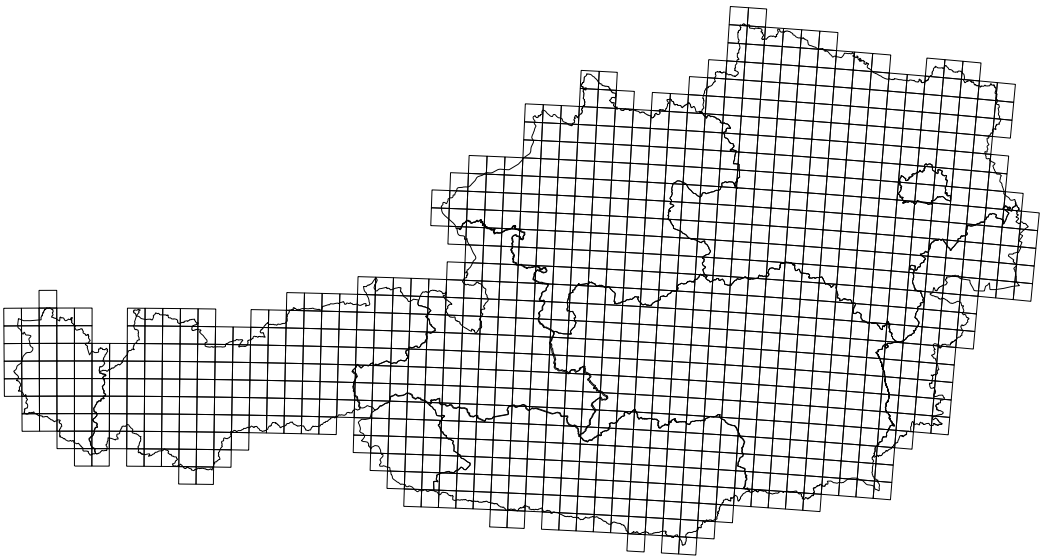

Abbildung 3.5: Das ETRS89-Raster wird hier in der World Mercator Projektion dargestellt. In dieser Projektion hat das Gradnetz waagrechte und senkrechte Achsen. Man beachte die starken Verzerrungen die dabei auftreten.

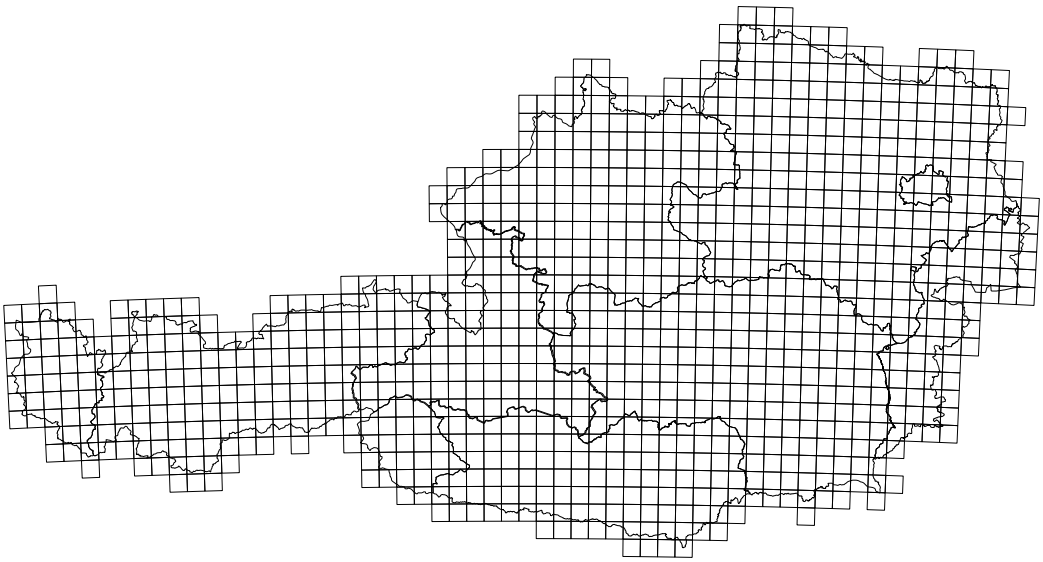

Abbildung 3.6: Das MGI-Raster wird hier in der World Mercator Projektion dargestellt. In dieser Projektion hat das Gradnetz waagrechte und senkrechte Achsen. Man beachte die starken Verzerrungen die dabei auftreten.

OSM XML OpenStreetMap eigenes Format

PostGIS Erweiterung von PostgreSQL-Datenbanken zur Speicherung von Geodaten

Im Folgenden werden nun jene Formate näher erläutert, die für diese Arbeit relevant sind.

3.3.1 Shapefile

Das Format *Shapefile* ist ein Datenformat das ursprünglich von ESRI Inc. (Environmental Systems Research Institute) [W9] für deren GIS-Produkte entwickelt wurde. Mittlerweile hat sich das Format aber weit verbreitet, da es recht einfach aufgebaut ist, und so stehen viele Daten in diesem Format zur Verfügung. Außerdem ist es mit vielen freien und kommerziellen GIS-Anwendungen möglich auf Daten in diesem Format zuzugreifen.

Aufbau

Pro Shapefile kann immer nur ein bestimmter Typ von Geodaten gespeichert werden; also entweder nur Punkte, Linien, Polygone, etc. Das Shapefile selbst besteht aus mehreren Dateien, die sich nur durch ihre Dateiendung unterscheiden. Dateien mit folgenden Endungen sind auf jeden Fall notwendig:

.shp Diese Datei enthält die Geometriedaten. Diese sind in mehrere Datensätze unterschiedlicher Länge unterteilt, wobei jeder Datensatz die Geodaten eines Objektes enthält.

.shx Der Index der Geometriedaten beinhaltet pro Geometriedatensatz einen Eintrag, der dessen Offset in der SHP-Datei angibt.

.dbf Diese Datei enthält Attributdaten im dBASE-Format. Für jeden Datensatz der SHP-Datei ist ein entsprechender Datensatz mit den zugehörigen Attributdaten vorhanden. Dabei ist zu beachten, dass die Reihenfolge und Anzahl der Datensätze in den SHP-Dateien und den DBF-Dateien gleich sein muss, da über die Datensatz-Nummer die Zuordnung der Datensätze zueinander erfolgt.

Weitere zusätzliche Dateien können im Shapefile enthalten sein, von denen hier aber nur die PRJ-Datei erwähnt sei. Diese enthält das Koordinaten-Bezugssystem, in dem die Geometriedaten im WKT-Format gespeichert sind. Ist diese Datei nicht vorhanden, muss der Anwender selbst wissen wie die Daten zu verwenden sind.

Für weitere technische Details zum Shapefile sei auf die *ESRI Shapefile Technical Description* [L10] verwiesen.

3.3.2 OSM XML

OSM XML ist ein Format, das hauptsächlich dazu verwendet wird um Daten des OpenStreetMap-Projektes bereitzustellen. Wie der Name schon sagt basiert es auf dem XML-Standard, jedoch ist die Datei-Endung .osm statt .xml. Da diese Dateien ziemlich groß werden können, werden sie meistens in komprimierter Version (z.B. mit der Dateiendung .osm.bz2) zur Verfügung gestellt.

Aufbau

Wie jede XML-Datei beginnt auch das OSM XML mit einem xml-Tag, der die XML-Version und den Zeichensatz der Datei angibt. Danach kommt ein osm-Block, der folgende Elemente enthalten kann:

- Node
- Way
- Relation

Alle diese Elemente haben eine Id als Attribut, um gegenseitiges Referenzieren zu ermöglichen. *Nodes* beschreiben Punkte und enthalten die geographische Information in Form von geodätischen Längen- und Breitenangaben. *Ways* enthalten Referenzen auf Punkte und beschreiben somit eine geordnete Menge von Punkten. Diese Menge kann auch geschlossen sein, wenn der erste und letzte Punkt übereinstimmen. Damit lassen sich also auch Polygone beschreiben. Ob ein geschlossener Weg nun eine Fläche beschreibt,

wie z.B. einen Teich, oder nur einen in sich geschlossenen Straßenzug, wird durch zusätzliche tag-Elemente definiert, die Attribute des Weges angeben. *Relationen* werden dazu verwendet um Nodes und Ways nach verschiedenen Gesichtspunkten zu gruppieren. Zum Beispiel kann eine Relation eine Buslinie darstellen und somit die einzelnen Straßenabschnitte und die Haltestellen zusammenfassen. Diese Relationen können selbst wieder Attribute enthalten, wie zum Beispiel den Namen der Buslinie. Relationen können rekursiv verschachtelt sein, wodurch der Aufbau von hierarchischen Strukturen möglich ist. Wie man hier deutlich sieht, verwendet das OpenStreetMap-Projekt das Entity-Relationship-Konzept.

3.3.3 PostGIS Datenbankformat

PostGIS ist eine Erweiterung für die PostgreSQL Datenbank, die es ermöglicht GIS-Objekte in dieser Datenbank zu speichern. PostGIS unterstützt räumliche Indizierung durch GiST[2]-basierte R-Trees und Funktionen zur Analyse und Bearbeitung von GIS-Objekten. Ein R-Tree ist eine Datenstruktur, in der Daten anhand ihrere räumlichen Position in Gruppen zusammengefasst und in einer Baumstruktur gespeichert werden. Dadurch lassen sich Daten durch die Angabe von räumlichen Gebieten besonders effizient abrufen. Eine detailliertere Beschreibung von R-Trees findet sich bei Guttman [L14]. Die Implementierung dieser Funktionen folgt dabei der OpenGIS „Simple Features Specification for SQL" [L25]. Die Software wird von Refractions Research Inc[W24], einem Unternehmen, das sich auf GIS und Datenbank-Consulting spezialisiert hat, entwickelt und unter der GNU General Public License [W10] der Allgemeinheit zur Verfügung gestellt [L32].

3.4 Datenquellen

Dieser Abschnitt befasst sich mit verschiedensten Arten von Datenquellen und geht dann im Speziellen auf frei verfügbare Datenquellen für den Raum Österreich ein. Gerade geographische Daten unterliegen oft einem strengen

[2]Generalized Search Tree

Kopierschutz, da deren Erfassung mit einem großen Aufwand verbunden ist. Daher muss unbedingt immer vor der Benutzung einer bestimmten Datenquelle abgeklärt werden, ob die Daten für den bestimmten Zweck benutzt werden dürfen und welche Rechte und Pflichten zu beachten sind.

3.4.1 Arten von Datenquellen

Druckmedien Das können Atlanten, Straßenkarten, Wanderkarten, politische Karten oder andere thematische Karten sein, wie zum Beispiel Karten die Wirtschaftswachstum, Populationsdichte, Bodennutzung, Vegetation, Temperatur oder Niederschlag darstellen, um nur einige zu nennen. Zur weiteren digitalen Benutzung dieser Daten müssen sie zuerst eingescannt und eventuell bearbeitet werden. Durch dieses Verfahren erhält man bereits eine Bilddatei, die durch ihre Pixel-Struktur einen Raster mit bestimmten Farbwerten definiert. Höchstwahrscheinlich ist der Raster viel zu fein für Modellierungszwecke und die Farbwerte sind mit Fehlerwerten belegt, doch durch Skalierung der Bilddatei und dem damit verbundenen Übergang zu einem gröberen Raster kann man quasi durch Bildung von Mittelwerten der einzelnen Farbwerte und deren Reduzierung auf bestimmte repräsentative Farben (Farbklassen) diesem Problem entgegentreten. Den verschiedenen Farbklassen können dann in der Modellbildung bestimmte Eigenschaften zugewiesen werden.

Sensoren Damit sind vor allem GPS-Geräte gemeint, die die globale Position messen und aufzeichnen können. Damit ließe sich zum Beispiel eine Karte eines Veranstaltungsortes erstellen, falls so eine noch nicht existiert, um Räumungsszenarien zu simulieren. Im Allgemeinen kann man sagen, dass diese Methode zu geographischen Daten zu gelangen, jedoch recht aufwendig ist und eher nur für kleine Gebiete in Frage kommt. Ein Vorteil allerdings ist, dass man sich keine rechtlichen Gedanken zu den Daten machen muss, wenn man sie selbst durch Messung sammelt.

Satellitenbilder, Luftbildaufnahmen Diese Daten können im Prinzip so wie *Druckmedien* als Grundlage für einen Raster zur Positionierung von Agenten verwendet werden. King [L19] und Kuckertz [L20] haben solche Daten für ihre Modelle verwendet. Beispiele für Webservices, die diese Daten verwenden und anzeigen, sind Google Maps [W13], Bing Maps [W15] und Yahoo Maps [W29].

Digitale Karten können auf die gleiche Art und Weise verwendet werden wie Printmedien, bloß dass diese nicht mehr eingescannt werden müssen. Zu den bereits vorhin genannten Beispielen für Satelliten- und Luftbildaufnahmen, die auch digitale Karten zur Verfügung stellen, kommt noch OpenStreetMap [W20] hinzu. Viel besser zur Weiterverarbeitung eignen sich allerdings sogenannte „Web Map Services (WMS)". Dort kann man Teile einer Karte für einen bestimmten Bereich mit einem Request anfordern und bekommt dann als Response die Karte für diesen Bereich übermittelt. Für die Details zu diesem Protokoll sei auf die OpenGIS®Web Map Server Implementation Specification [L26] verwiesen. Das Magistratsamt 14 (Automationsunterstützte Datenverarbeitung, Informations- und Kommunikationstechnologie) in Wien stellt zum Beispiel einen solchen WMS-Dienst mit den Daten von ganz Wien zur Verfügung [W28].

Datenbanken die geographische Daten enthalten sind wohl das Optimum, das man sich unter den Datenquellen vorstellen kann. Da es sich dabei um vektorielle Daten handelt, die mit Attributen versehen sind, lassen sich diese Daten nach bestimmten Aspekten filtern und sortieren um je nach Anwendungsgebiet die entsprechenden Daten zu verwenden. Doch nur selten hat man direkten Zugriff auf fremde Datenbanken. Eine Ausnahme dabei ist OpenStreetMap. Dieses Projekt bietet seine ganze Datenbank im OSM-Format zum Download an. Weitere Informationen dazu finden sich in deren Wiki [W21].

Vektordaten-Dateien können Dateien in verschiedensten Formaten sein (siehe Abschnitt 3.3), die sich dazu eignen Vektordaten zu speichern. Am

häufigsten werden dafür Shapefiles verwendet. Während Daten mit niedriger Auflösung meist frei zugänglich sind, muss man für hochauflösende Daten mit einigen Kosten rechnen.

Statistische Daten sind statistische Größen wie Population oder Niederschlag, gemessen in bestimmten geographischen Regionen, also Daten die dem *feldbasierten Konzept* folgen. Diese Daten werden in tabellarischer Form, entweder als Datenbanktabelle oder als CSV-Datei, gespeichert. Um die einzelnen Datensätzen in einen geographischen Kontext zu setzen, benötigt man immer eine Definition der verwendeten geographischen Regionen, zum Beispiel als Shapefile. Die einzelnen Datensätze können dann über einen Index oder eine ID den Regionen zugeordnet werden.

Adressdaten Wenn bisher von geographischen Daten und Positionen gesprochen wurde, so wurde eigentlich immer davon ausgegangen, dass diese explizit durch Punkte in einem definierten Koordinatensystem gegeben sind. Punkte können aber auch indirekt durch die Lage zu anderen Objekten bestimmt werden. So ist es zum Beispiel viel leichter den Standort eines Geschäfts durch die Straße und Hausnummer zu definieren als durch Längen- und Breitengrade. Um diese Adressdaten in einer Karte darzustellen, müssen den Adressen Koordinaten zugeordnet werden. Dieser Prozess heißt *Geokodierung* oder *Georeferenzierung*. Dabei werden Datenbanken von Straßennahmen, Postleitzahlen, usw. verwendet, die bereits entsprechende Koordinaten enthalten. Je genauer die Adressdaten sind, desto genauer entsprechen auch die so erhaltenen Koordinaten der Adresse. OpenStreetMap bietet zum Beispiel ein Werkzeug zur Geokodierung über eine Web-Schnittstelle an: Nominatim [W19]. Hat man den Adressdaten, die beispielsweise aus einer Kundendatenbank stammen, Koordinaten zugeordnet, so kann man diese wie Punktdaten, etwa aus Shapefiles, verwenden.

3.4.2 Datenquellen für das Gebiet Österreich

Dieser Abschnitt listet nun frei zugängliche und verwendbare Datenquellen verschiedenster Art für den Raum Österreich auf.

OpenStreetMap [W20] hat sich zum Ziel gesetzt eine freie editierbare globale Karte zu erstellen. Dadurch dass jeder die Möglichkeit hat die Karte zu editieren, ist die Datenmenge ständig am wachsen. Besonders in urbanen Lebensräumen ist der Detailgrad von OpenStreetMap beachtenswert. Die OpenStreetMap-Daten können entweder als Ganzes in Form einer planet.osm-Datei [W21] im Format OSM XML oder in Ausschnitten für einzelne Länder [W17] heruntergeladen werden.

Statistik Austria [W3] Die *Statistik Austria* stellt verschiedene geographische Daten zur Verfügung. Für die Parametrisierung des Modells relevant sind besonders die Daten zur Population und zum Dauersiedlungsraum [W1]. Daten zur Population sind in Form von Rasterdaten gegeben. Daher teilen sich diese Daten einerseits in die geographische Rastereinteilung [W2] und andererseits in eine Tabelle, die den einzelnen Rasterzellen Daten zur Population zuweist. Weitere Daten zu Siedlungseinheiten, Stadtregionen und NUTS[3]-Einheiten sind auch frei zugänglich.

Geoland [W11] Diese Webseite fasst sämtliche Informationen zu Geodaten der österreichischen Bundesländer zusammen. Da von den Ländern sehr viele Daten angeboten werden, besonders Karten zu verschiedensten Themenbereichen, werden hier nur die verfügbaren Vektordaten angeführt, da diese am universellsten einsetzbar sind.

Niederösterreich Gemeindegrenzen, Katastralgemeindegrenzen, Hochwasseranschlagslinien 30-, 100- und 300-jährig

Oberösterreich Gemeinde-, Bezirks- und Landesgrenzen, Bezirkshauptorte, Seen und Hauptgewässer, Autobahnen und Bundesstraßen

[3]Nomenclature des unités territoriales statistiques

Salzburg Gemeindegrenzen, Katastralgemeindegrenzen, Gemeindehauptorte, Bezirksgrenzen, Landesgrenzen

Steiermark Landesgrenze, Bezirke, Gerichtsbezirke, Gemeinden, Orte, Hauptgewässer Flüsse, Hauptverkehrsnetz Straße

Vorarlberg Gemeindegrenzen, Katastralgemeindegrenzen, Bezirksgrenzen, Landesgrenzen, Geländemodell, Naturschutzgebiete, Polizeiinspektionen, Schulen, Tankstellen, Flächenwidmungsplan, Flüsse, Seen

Wien Politische Grenzen, Zählbezirke, Auszüge aus dem Stadtentwicklungsplan, Strassenbahn - Planung, U-Bahn Linien Planung, Grünanlagen

Kapitel 4

Integration von GIS-Daten in AB-Modelle

Integration von GIS-Daten kann in sehr vielen unterschiedlichen Bereichen geschehen, unter Verwendung verschiedenster Werkzeuge und Programmiersprachen. Dieses Kapitel stellt einen Leitfaden zur Verfügung, wie GIS-Daten in ein Modell integriert werden können. Dieser Leitfaden ist sehr allgemein gehalten um die Anwendung nicht auf bestimmte Modellierungswerkzeuge oder Programmiersprachen zu beschränken. Detailliertere Beschreibungen und Beispiele müssen sich auf konkrete Werkzeuge beziehen. Um das Konzept nicht auf diese Werkzeuge zu beschränken, werden, wenn möglich, Verweise auf Alternativen gegeben.

4.1 Datenquellen zusammenführen

Hat man Daten aus verschiedensten Datenquellen gesammelt, so ist es meistens sinnvoll, wenn man diese an einem gemeinsamen Ort ablegt und verwaltet. Eine PostGIS Datenbank eignet sich dazu sehr gut, da man damit rasch auf die Daten zugreifen, sie selektieren und analysieren kann. Ein Nachteil ist allerdings, dass man auch Zugriff auf einen Datenbankserver benötigt. Man kann sich so einen Server zwar immer lokal auf dem eigenen Rechner installieren, doch wenn man in einem Team arbeitet, dann sollte dieser schon

über das Netzwerk jedem zugänglich sein. Wie die Daten in die Datenbank importiert werden können, wird im Folgenden für die beiden Formate Shapefile und OSM-XML beantwortet. Als universelles Werkzeug zum Umgang mit verschiedensten Geodaten hat sich Quantum GIS[W22] gut bewährt.

Quantum GIS (QGIS) ist ein Open Source GIS, das zahlreiche Vektor-, Raster- und Datenbankformate unterstützt. Die Daten können damit betrachtet, verändert, kombiniert und konvertiert werden. Durch ein sogenanntes Plugin-System kann QGIS auch mit weiteren Funktionalitäten ausgestattet werden.

Shapefile

Um Shapefiles in eine PostGIS Datenbank zu importieren, benötigt man das SPIT[1], das Teil von QGIS ist. Wie in Abb. 4.1 zu sehen, muss man in dem Dialog eine Verbindung zu der PostGIS Datenbank herstellen und die Shapefiles auswählen. Zu den Importoptionen ist zu sagen, dass es unbedingt notwendig ist den Spatial Reference System Identifier (SRID) anzugeben, da sonst die importierten Daten mit keinem Bezugssystem verknüpft sind. Die SRID in der EPSG-Datenbank entspricht dem EPSG-Code ohne den „EPSG::"-Präfix. Die SRID für das Bezugssystem WGS 84 wäre zum Beispiel 4326, wie aus der Tabelle 3.1 abzulesen ist. Durch den Import wird in der Datenbank eine Tabelle mit dem Namen des Shapefiles angelegt. Diese Tabelle enthält eine Primärschlüsselspalte, deren Name in den Importoptionen festgelegt werden kann und die zu jedem importierten Objekt eine eindeutige ID enthält. Außerdem wird eine weitere Spalte angelegt, die die Geometrie des Objekts in einem binären Format enthält. Auch deren Name kann in den Optionen festegelegt werden. Weitere Spalten werden für die Attribute, die im Shapefile zu den Objekten definiert sind, angelegt.

[1] Shapefile to PostGIS Import Tool

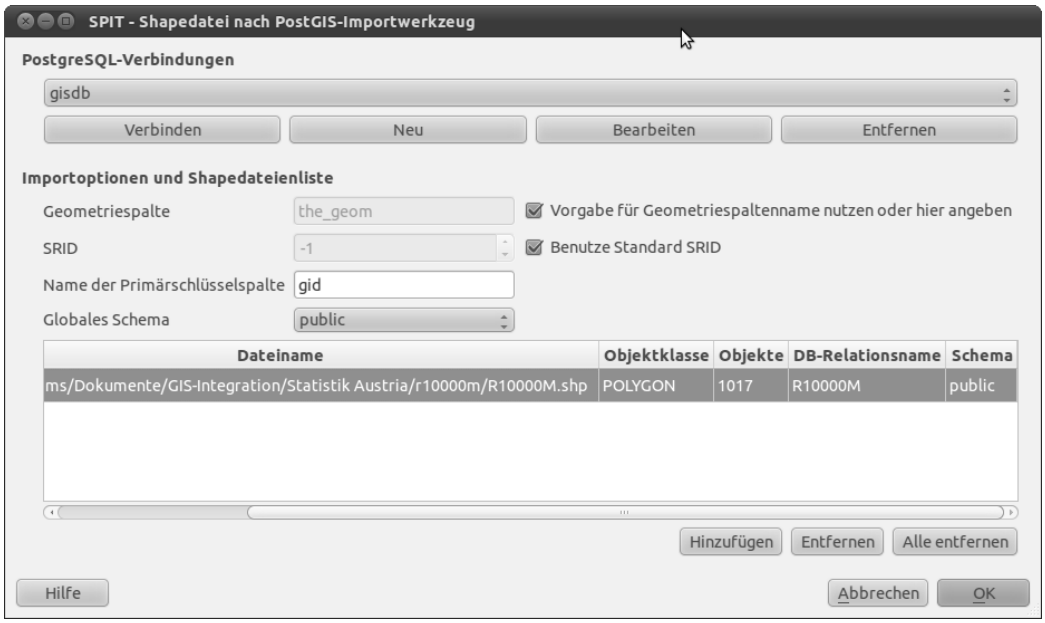

Abbildung 4.1: Shapefile to PostGIS Import Tool (SPIT)

```
bzcat downloaded.osm.bz2 | \
    osmosis --read-xml enableDateParsing=no file=- \
    --bounding-box top=49.1 left=9.5 bottom=46.3 right=17.2 \
    --write-xml file=- | \
    bzip2 > extracted.osm.bz2
```

Listing 4.1: Kommando zum Ausschnitt eines Bereiches aus der OSM-Datei

OSM-XML

Da OSM-XML-Dateien selbs in komprimierter Form sehr groß sind, ist es von Vorteil zuerst den Bereich der Daten zu extrahieren, der für die Anwendung relevant ist. Dieser Prozess kann mit dem Programm *Osmosis* durchgeführt werden. Listing 4.1 zeigt zum Beispiel, wie ein Rechteckiger Bereich, der die Daten für Österreich enthält, ausgeschnitten wird.

Danach können die OSM-Daten mit dem Programm *osm2pgsql* in eine PostGIS-Datenbank importiert werden. Für diesen Import müssen die Datenbank und die Zugangsdaten dafür bekannt sein, wie das in Listing 4.2 zu

```
osm2pgsql --database dbgis --latlong --slim --host localhost \
    --username dbuser --password gis1234 extracted.osm.bz2
```

Listing 4.2: Import von OSM-Daten in eine PostGIS-Datenbank

sehen ist. Durch den Import werden in der Datenbank diese vier Tabellen angelegt:

- **planet_osm_line** enthält importierte Linienzüge mit Attributen
- **planet_osm_point** enthält importierte Knotenpunkte mit Attributen
- **planet_osm_polygon** enthält importierte Polygone mit Attributen
- **planet_osm_roads** enthält eine Untermenge von *planet_osm_line*, die sich für grobe Auflösungen eignet

Durch den Import verschiedenster Datenquellen in eine gemeinsame Datenbank kann es vorkommen, dass sich die einzelnen Tabellen mit Geodaten auf unterschiedliche Koordinatensysteme beziehen. An und für sich ist das kein Problem, solange die Software, mit der die Daten bearbeitet und dargestellt werden sollen, damit umgehen kann. GeoTools (siehe Abschnitt 4.2.3) kann zum Beispiel „on-the-fly" vom Bezugssystem der Datenquelle in das darzustellende Koordinatensystem umrechnen, auch wenn Quellen mit verschiedenen Referenzsystemen gemeinsam dargestellt werden. Es ist allerdings anzunehmen, dass für die Umrechnung zwischen den Koordinatensystemen ein gewisser Aufwand nötig ist. Da es einige verschiedene Referenzsysteme und Projektionen gibt kann hier nicht auf die vielen unterschiedlichen Formeln und Algorithmen zur Umrechnung eingegangen werden. Es ist sogar möglich, dass es mehrere Möglichkeiten zur Transformation gibt. Eine detaillierte Beschreibung der Transformationen findet man in [L17]. Als einfaches Beispiel einer Umrechnungsformel kann auch jene aus Abschnitt 3.2 zur Umrechnung von kartesischen in geographische Koordinaten dienen. Um den zusätzlichen Aufwand der Transformation zu vermeiden, ist es eventuell von Vorteil sich für ein System zu entscheiden und die anderen Daten

in dieses zu konvertieren. Damit die Konvertierung mit PostGIS funktioniert, muss das Ziel-Bezugssystem in der Tabelle *spatial_ref_sys* vorhanden sein. Diese Tabelle wird bei der Erstellung einer PostGIS-Datenbank erzeugt und enthält über 3000 verschiedene Koordinatensysteme und Informationen darüber, wie zwischen diesen umgerechnet werden kann. Außerdem muss PostGIS auch mit der Unterstützung für *Proj*[W26] kompiliert worden sein. *Proj* ist eine Software-Bibliothek, mit der zwischen geographischen Projektionen umgerechnet werden kann. Unterstützung für *Proj* ist vorhanden, wenn im Ergebnis der folgenden Abfrage ein entsprechender Eintrag vorhanden ist.

```
SELECT PostGIS_Full_Version();

POSTGIS="1.5.3" GEOS="3.2.2-CAPI-1.6.2"
PROJ="Rel. 4.7.1, 23 September 2009"
LIBXML="2.7.8" USE_STATS
```

Für das folgende Beispiel sei *streets* eine Tabelle mit Straßendaten, die in der Spalte *way* die Geometriedaten enthält. Diese sollen in das WGS 84 (SRID 4326) System konvertiert werden. Für den Fall, dass die Umrechnung fehl schlägt und da PostGIS die Änderung des Bezugssystems in bestehenden Tabellen nicht ohne weiteres zulässt, kopiert man sich die Geodaten zuerst in eine neue Tabelle *new_streets*, in der man dann die eigentliche Transformation durchführt.

```
CREATE TABLE new_streets AS ( SELECT * FROM streets );
```

Danach kann man die Daten in der Tabelle *new_streets* transformieren. Dabei kommt die von PostGIS zur Verfügung gestellte Funktion $ST\_transform(geometry, integer)$ zur Anwendung, die genau zwei Parameter übernimmt. Erstens, die Geometrie, die transformiert werden soll und zweitens, die SRID des Ziel-Koordinatensystems. Als Rückgabewert erhält man die in das Ziel-Koordinatensystem konvertierte Geometrie.

```
UPDATE new_streets SET way = ST_transform(way,4326);
```

PostGIS enthält nach *spatial_ref_sys* noch eine zweite Meta-Tabelle *geometry_columns*, die Informationen über alle Tabellen mit Geometrieda-

name	...	longitude	latitude

Tabelle 4.1: Tabelle *hospitals*

name	...	admin_level	...	way

Tabelle 4.2: Tabelle *planet_osm_polygon*

ten enthält. In dieser Tabelle wird für jede einzelne Tabelle der Name der Tabelle und der Spalte mit Geometriedaten, die Dimension und SRID des Bezugssystems und der Type der Daten (Punkte, Linien, Polygone, ...) gespeichert. Da zu Beginn des Beispiels eine Kopie der ursprünglichen Tabelle angelegt wurde, ist die Tabelle *geometry_columns* nun nicht mehr aktuell. Die Konsistenz kann mit der Funktion *populate_geometry_columns*() wiederhergestellt werden.

```
SELECT populate_geometry_columns();
```

Dadurch wird ein neuer Eintrag für *new_streets* mit der SRID 4326 in dieser Tabelle generiert.

Hat man verschiedene Datenquellen in einer Datenbank zusammengefasst, so kann man durch deren Kombination zu neuen Informationen gelangen. Für das folgende Beispiel seien die Tabellen *hospitals* und *planet_osm_polygon* definiert. Die erste der beiden enthält eine Liste von Krankenhäusern mit zugehörigen Längen- und Breitengrade in den Spalten *longitude* und *latitude*. Der Name des Krankenhauses steht in der Spalte *name*. Die zweite Tabelle enthält die von OpenStreetMap importierten Polygon-Daten. Die Attribute (z.B. *admin_level*) stehen in einzelnen Spalten und die Geometriedaten sind in der Spalte *way* gespeichert. Die Tabellen 4.1 und 4.2 zeigen dies schematisch. Mit der Abfrage in Listing 4.3 kann dann zu jedem Krankenhaus die Gemeinde gefunden werden, in der es sich befindet. Durch die Bedingung *p.admin_level*='8' in der Abfrage werden die Gemeinde-Polygone aus den Daten herausgefiltert. Die PostGIS-Funktionen

```
SELECT h.name, p.name
    FROM hospitals AS h JOIN planet_osm_polygon AS p
        ON p.admin_level='8' AND ST_Contains(p.way,
            ST_SetSRID(ST_Point(h.longitude, h.latitude), 4326))
```

Listing 4.3: Abfrage zur Zuordnung von Krankenhäusern und Gemeinden

ST_Point, *ST_SetSRID* und *ST_Contains* werden dann in dieser Reihenfolge dazu verwendet um

1. aus den Spalten *longitude* und *latitude* ein Punkt-Objekt zu erzeugen,
2. diesem Punkt ein Bezugssystem zuzuweisen (SRID 4326) und
3. zu überprüfen ob das Gemeinde-Polygon *p.way* diesen Punkt enthält.

Der zweite Punkt ist deshalb notwendig, da PostGIS einem neuen Punkt nicht automatisch ein Bezugssystem zuweist. Bei den Koordinaten des Punktes könnte es sich auch um Entfernungen zu einem Referenzpunkt handeln. Erst die Zuweisung des Bezugssystems mit der SRID 4326 legt fest, dass es sich dabei um Längen- und Breitengrade handelt. Zu Erstellung, Vergleich und anderen Operationen auf geographischen Objekten gibt es noch viele weitere Funktionen als die oben angeführten. Eine vollständige Liste dieser PostGIS-Funktionen findet man in der PostGIS Dokumentation[W23].

4.2 Open Source GIS Software-Bibliotheken

Jede Software, die dazu dient Geodaten zu verarbeiten, muss Funktionen implementiert haben, die diese Daten einlesen, konvertieren, speichern oder in einer anderen Weise bearbeiten können. Aus diesem Grund macht es Sinn diese Funktionen in eigenständige Bibliotheken auszulagern, die dann von der Software oder einem Modell verwendet werden können um Geodaten zu bearbeiten. Dieser Abschnitt soll nun einen kurzen Überblick über die verfügbaren Bibliotheken im Open Source Bereich geben. Man kann die Software-Bibliotheken nach der verwendeten Programmiersprache in drei große Gruppen einteilen: C/C++, Java und .Net

h.name (Krankenhaus)	p.name (Gemeinde)
Landeskrankenhaus Oberpullendorf	Gemeinde Oberpullendorf
Bezirkskrankenhaus Schwaz	Gemeinde Schwaz
Krankenhaus St. Vinzenz Zams	Gemeinde Zams
Bezirkskrankenhaus Kufstein	Gemeinde Kufstein
Landeskrankenhaus Bludenz	Gemeinde Bludenz
Landeskrankenhaus Bregenz	Gemeinde Bregenz
Krankenhaus der Stadt Dornbirn	Gemeinde Dornbirn
Landesklinik St. Veit	Gemeinde Sankt Veit im Pongau
...	...

Tabelle 4.3: Zuordnung von Krankenhäusern zu Gemeinden entsprechend der Abfrage in Listing 4.3.

C/C++ Bibliotheken:

GEOS Algorithmen zur Verarbeitung von Geometriedaten

OGR/GDAL Zugriff auf Geodaten in verschiedenen Formaten

Proj4 Transformationen zwischen verschiedenen Bezugssystemen

FDO Zugriff auf Geodaten in verschiedenen Formaten

Java Bibliotheken:

OpenMap Umfassende GIS-Bibliothek

GeoTools Umfassende GIS-Bibliothek

JTS Algorithmen zur Verarbeitung von Geometriedaten

.Net Bibliotheken:

Proj.Net Portierung von Proj4

NTS Portierung von JTS

.Net-Anwendungen haben den Vorteil, dass sie auch direkt C/C++ Bibliotheken verwenden können und diese nicht extra nach .Net portiert werden müssen.

In den folgenden Abschnitten wird kurz auf die für diese Arbeit relevanten Bibliotheken eingegangen. Eine Übersicht über alle oben genannten Bibliotheken findet man in „The State of Open Source GIS" [L31].

4.2.1 OpenMap

OpenMap [W25] ist eine Java Bibliothek, die ursprünglich von BBN Technologies entwickelt wurde, 1998 aber als Open Source Projekt veröffentlicht wurde. Nach Ramsey[L31] war es sogar die erste Open Source Java-Bibliothek, die geographische Funktionen zur Verfügung stellte. Die Version 4.6.5 wurde am 5. März 2009 veröffentlicht. Diese Version kann mit Shapefiles umgehen, PostGIS-Datenbanken werden allerdings nicht unterstützt. Durch den modularen Aufbau von OpenMap ist es aber möglich weitere Funktionen selbst hinzuzufügen. Am 23. Dezember 2011 ist nun wieder eine neue Version erschienen: OpenMap 5.0. Diese ist nicht mehr abwärtskompatibel und benötigt nun Java 5. PostGIS-Datenbanken werden immer noch nicht unterstützt.

4.2.2 JTS

JTS Topology Suite [W8] ist eine Java Bibliothek, die grundlegende Funktionen zur Behandlung von geometrischen Objekten im 2D-Raum zur Verfügung stellt. Dazu gehören

- Flächen- und Distanzfunktionen
- Geometrische Prädikate basierend auf dem Dimensionally Extended Nine-Intersection Model (DE-9IM) [L7]
- Überlagerungsfunktionen (Schnittmenge, Differenz, Vereinigung, Symmetrische Differenz)
- Konvexe Hülle

- Geometrische Vereinfachung
- Reduktion der Genauigkeit
- Delaunay Triangulierung (mit Nebenbedingungen)
- Voronoi Diagramm
- Kleinstes umschließendes Rechteck
- Diskrete Hausdorff-Distanz
- Punkt in Polygon Bestimmung
- Quadtrees und STR[2]-Trees
  Diese Datenstrukturen partitionieren einen zwei- oder dreidimensionalen Raum um die Suche von Daten anhand von Koordinaten zu beschleunigen. Während Quadtrees den Raum regelmäßig unterteilen, passen sich STR-Trees den Daten an.

Das in JTS implementierte Geometriemodell entspricht der „OGC Simple Features Specification for SQL“ [L25]. Das Besondere an JTS ist, dass bei der Implementierung besonders auf die Robustheit der verwendeten Algorithmen geachtet wurde. Alternativen zu JTS in anderen Programmiersprachen sind GEOS und NTS.

4.2.3 GeoTools

GeoTools [W12] ist eine Java-Bibliothek, die Standard-Methoden zur Manipulation von geographischen Daten zur Verfügung stellt. Die Bibliothek entspricht ebenfalls den Spezifikationen des Open Geospatial Consortium (OGC). GeoTools verwendet JTS für die Geometriedaten.

[2]Sort-Tile-Recursive [L33]

Einige Features von GeoTools sind

- Unterstützung vieler Datei-Formate und Datenbanken wie z.B. PostGIS
- Koordinaten-Bezugssysteme und Transformationen
- Karten-Projektionen
- Filterung von Daten anhand von (räumlichen) Attributen
- Erstellung und Anzeige von Karten

GeoTools wird seit 1996 entwickelt und hat mittlerweile schon einen sehr ausgereiften Zustand erreicht. Ein Zeichen dafür ist auch, dass es einige Projekte gibt, die auf GeoTools aufbauen. Das sind zum Beispiel GeoServer, uDig, GeoVISTA Studio oder MyMaps. Alternativen zu GeoTools in anderen Programmiersprachen sind OGR/GDAL oder FDO.

4.3 Integration in Java

Einige Simulationsumgebungen verwenden Java zur Erstellung der Modelle oder bauen zumindes darauf auf und können so zusätzliche Java-Bibliotheken verwenden. Aus diesem Grund ist es von besonderem Interesse wie die Integration von Geodaten in dieser Programmiersprache funktioniert. Dafür verwendet man am Besten eine Bibliothek, die mit Geodaten umgehen kann. Hier und in allen nachfolgenden Beispielen wird GeoTools dafür verwendet. Datenquellen werden durch das Interface *DataStore* repräsentiert. Dieses wird zum Beispiel von den Klassen $ShapefileDataStore$ und $JDBCDataStore$ implementiert, welche den Zugriff auf Daten in Shapefiles oder Datenbanken ermöglichen. Von einem *DataStore* können verschiedene „Datenquellen" mit der Funktion $getFeatureSource(String\ typeName)$ abgerufen werden. Der Gedanke dahinter ist, dass ein *DataStore* mehrere $FeatureSources$ enthalten kann, in dem Sinn wie eine Datenbank mehrere Tabellen enthält. Der Parameter $typeName$ steht in diesem Fall für den Namen der Datenbanktabelle. Als Features werden die einzelnen geographischen

Objekte — Punkte, Linien und Polygone mit ihren zugehörigen Attributen — bezeichnet.

Eine *SimpleFeatureSource*, wie sie in Listing 4.4 verwendet wird, ist eine *FeatureSource* mit Features vom Typ *SimpleFeatureType*. Dies ist im Moment der einzige unterstützte *FeatureType*. Ein „Simple Feature" kann nur einfache Attribute enthalten; Attribute können nur einmal pro Feature auftreten und sie sind geordnet in dem Sinn, dass sie über einen Index abgerufen werden können, was aber nic bedeutet, dass sie nach dem Namen oder einer anderen Eigenschaft sortiert sind. Im Zuge dieser Arbeit haben sich diese Einschränkungen jedoch weder im Zusammenhang mit OpenStreetMap-Daten noch mit Shapefiles in irgendeiner Form ausgewirkt.

Genauso wie bei der *SimpleFeatureSource* bezeichnet eine *SimpleFeatureCollection* eine *FeatureCollection* mit Features des vorher genannten Typs. Eine *FeatureCollection* ist eine Sammlung bestimmter Features, die aus einer *FeatureSource* stammen. Das Abrufen der Features erfolgt durch die Funktion *getFeatures*, wobei auch die Angabe von Filtern, also bestimmten Bedingungen, die die Features erfüllen müssen, möglich ist. Diese Filter können durch die Contextual Query Language (CQL) ausgedrückt werden. Diese Abfragesprache wird in [L23, Kapitel 6.2.2] unter dem Namen Common Query Language definiert.

Um die Daten anzeigen zu können müssen diese zuerst zu einer Karte hinzugefügt werden. Der Inhalt der Karte wird durch die Klasse *MapContent* repräsentiert. Dieser Inhalt kann aus mehreren Schichten, sogenannten Layer (Klasse *FeatureLayer*) aufgebaut sein, welche wiederum je eine *FeatureSource* oder *FeatureCollection* enthalten. Diese Layer kann man sich wie Overhead-Folien vorstellen, die übereinander gelegt werden. Zu jedem Layer kann über einen *Style* definiert werden, wie dieser dargestellt werden soll. Die Klasse *SLD* (Styled-Layer Descriptor) bietet Methoden an um einfache *Styles* zu erstellen. Wie diese Styles spezifiziert sind und welche Möglichkeiten sich damit erschließen kann in [L27] und [L28] nachgelesen werden.

Schließlich kann die Karte in einem von GeoTools zur Verfügung gestellten Fenster *JMapFrame* oder mittels *JMapPane* in einem selbst erstellten

Fenster angezeigt werden. Der Vorteil von *JMapFrame* ist, dass es dort schon Möglichkeiten zum Zoomen und Verschieben der Karte gibt, die man sonst noch selbst hinzufügen muss. Listing 4.4 gibt ein Sourcecode-Beispiel für die besprochenen Schritte. Dabei werden Daten aus einem Shapefile, das vom Benutzer mit der Hilfsklasse *JFileDataStoreChooser* abgefragt wird, sowie aus einer PostGIS-Datenbank eingelesen und in einem Fenster angezeigt.

4.4 Agentenbasierte Modellierungswerkzeuge und GIS

NetLogo [L36] unterstützt Vectordaten (Shapefiles) und Rasterdaten in Form von ESRI ASCII Grid Dateien. Dieses von ESRI [W9] entwickelte Rasterdatenformat ist eine einfache Textdatei, die zu Beginn die Position und Dimension des Rasters definiert und danach Zeile für Zeile die Werte des Rasters enthält, wobei die Spalten durch Leerzeichen getrennt werden. Für die Verarbeitung von Geometriedaten verwendet NetLogo die Bibliothek JTS und für Koordinaten bzw. die Umrechnung von Koordinatensystemen die Bibliothek JScience [W7]. Nach der Definition einer Transformation vom GIS-Raum in den NetLogo-Raum können die Geodaten in der 2D- und 3D-View von NetLogo angezeigt werden.

Repast [W14] ermöglicht es sowohl die Bibliothek OpenMap als auch GeoTools mit den entsprechenden Möglichkeiten zu verwenden. Zur Anzeige der Geodaten mit den Agenten wird im Fall von GeoTools ESRI ArcMap verwendet. ArcMap ist ein Teil des von ESRI [W9] entwickelten Software-Pakets ArcGIS, welches zur Anzeige, Bearbeitung und Analyse von Geodaten eingesetzt werden kann. Bei Verwendung von OpenMap wird die von OpenMap selbst zur Verfügung gestellte Anzeige genützt.

MASON [L21] selbst unterstützt keine Geodaten, doch mit der Erweiterung GeoMason [L35] kann diese Funktionalität hinzugefügt werden. GeoMa-

```
// Shapefile vom Benutzer abfragen
File file = JFileDataStoreChooser.showOpenFile("shp", null);
if (file == null) { return; }

// Shapefile einlesen
FileDataStore shpStore = FileDataStoreFinder.getDataStore(file);
SimpleFeatureSource shpFeatureSource = shpStore.getFeatureSource();

// Daten aus einer PostGIS-Datenbank einlesen
Class.forName("org.postgresql.Driver");

Map params = new HashMap();
params.put("dbtype", "postgis");
params.put("host", "localhost");
params.put("port", new Integer(5432));
params.put("database", "gisdb");
params.put("user", "user1");
params.put("passwd", "abcd");

DataStore pgStore = new PostgisNGDataStoreFactory().
    createDataStore(params);

SimpleFeatureSource pgFeatureSource = pgStore.
    getFeatureSource("boundaries");
SimpleFeatureCollection pgFeatureCollection = pgFeatureSource.
    getFeatures(CQL.toFilter("admin_level = 8"));

// Erzeugen eines MapContent und Hinzufügen der Layer
MapContent map = new MapContent();
map.addLayer(new FeatureLayer(shpFeatureSource,
    SLD.createPolygonStyle(gray, blue, 0.5f)));
map.addLayer(new FeatureLayer(pgFeatureCollection,
    SLD.createPolygonStyle(gray, green, 0.5f)));

JMapFrame.showMap(map);
```

Listing 4.4: Integration von Geodaten in Java mittels GeoTools

son kann von sich aus Shapefiles handhaben. Für andere Dateiformate bietet GeoMason eine Schnittstelle zu GeoTools und OGR/GDAL an. Die Geodaten werden im Anzeigebereich von Mason dargestellt. Diese starke Integration in Mason wird durch die Verwendung von objektorientierten Konzepten realisiert.

AnyLogic [W6] ermöglicht die Verwendung von Geodaten als Basis für das Environment, in dem sich Agenten bewegen können. Dazu verwendet es die Bibliothek OpenMap in der Version 4.6.3. Über die graphische Oberfläche von AnyLogic können nur Shapefiles verwendet werden. Über die Programmierschnittstelle hat man aber auf alle Funktionalitäten von OpenMap Zugriff, muss dort aber auch mit den Einschränkungen, die OpenMap hat, auskommen (z.B. kein nativer Zugriff auf PostGIS-Datenbanken). Die Geodaten werden in der für AnyLogic gewohnten Weise im Environment dargestellt.

4.5 Integration in AnyLogic

Macal and North [L22] beschreiben fünf verschiedene Topologien, die verwendet werden können um Interaktionen zwischen Agenten zu repräsentieren: „Soup“ Model (ohne räumlichen Bezug), Zelluläre Automaten, Euklidischer Raum, GIS und Netzwerk. Diese Möglichkeiten existieren auch in AnyLogic durch die Verwendung von Umgebungen oder das Erzeugen von Verbindungen zwischen Agenten. Wird keine Umgebung verwendet entspricht das dem „Soup“-Model. Im Folgenden werden nur Topologien mit räumlichen Bezug betrachtet. Diese entsprechen genau den Umgebungen in AnyLogic: ein kontinuierlicher Raum in zwei oder drei Dimensionen (*EnvironmentContinuous2D*, *EnvironmentContinuous3D*), ein zwei-dimensionaler diskreter Raum (Rasterfeld) mit einer bestimmten Anzahl an Zeilen und Spalten (*EnvironmentDiscrete2D*) und eine geographische Umgebung (*EnvironmentContinuousGIS*). Abhängig von der verwendeten Umgebung unterscheiden sich die Agenten auch in ihren Eigenschaften und Methoden (Tabelle 4.5).

	AgentDiscrete2D	**AgentContinuous2D/3D**	**AgentContinuousGIS**
Koordinaten	Zeilen, Spalten	X, Y, Z[1]	Längen-, Breitengrade
Abstände	Nachbarschaft	euklidisch	GIS-induziert
Ähnliche Methoden	setCell getR, getC jumpToCell moveToNextCell getNeighbours	setXY, setXYZ[1] getX, getY, getZ[1] jumpTo moveTo getNearestAgent	
Verschiedene Methoden	getAgentAtCell getAgentNextToMe findRandomEmptyCell jumpToRandomEmptyCell	moveToNearestAgent getTargetX, getTargetY, getTargetZ[1] setVelocity getVelocity setRotation getRotation, getVerticalRotation[1] isMoving timeToArrival onArrival stop distanceTo	

[1] nur bei AgentContinuous3D vorhanden

Tabelle 4.4: Unterschiede zwischen den Agentenklassen *AgentDiscrete2D*, *AgentContinuous2D/3D* und *AgentContinuousGIS*. Die hier genannten Methoden werden in Tabelle 4.5 näher beschrieben.

Funktion	**Beschreibung**
setCell, setXY, setXYZ	Legt die Position eines Agenten anhand von Zeilen/Spalten oder Koordinaten fest
getR, getC, getX, getY, getZ	Liefert die aktuelle Agentenposition
jumpToCell, jumpTo	Bewegt einen Agenten ohne Verzögerung zu einer beliebigen anderen Zelle/Position
moveToNextCell	Bewegt einen Agenten zu einer Nachbarzelle in einer bestimmten Richtung
moveTo	Legt ein Ziel für einen Agenten fest und bewegt ihn auf direktem Weg dorthin
getNeighbours	Liefert die Agenten in benachbarten Zellen
getNearestAgent	Liefert den nächstgelegenen Agenten
getAgentAtCell	Liefert den Agenten in einer bestimmten Zelle
getAgentNextToMe	Liefert den nächsten Agenten in einer bestimmten Richtung
findRandomEmptyCell	Liefert eine zufällige leere Zelle
jumpToRandomEmptyCell	Bewegt einen Agenten ohne Verzögerung zu einer leeren Zelle
moveToNearestAgent	Bewegt einen Agenten auf direktem Weg zu jenem Agenten der ihm am nächsten ist
getTargetX, getTargetY, getTargetZ	Liefert eine Koordinate der Zielposition
setVelocity, getVelocity	Setzt/Liefert die Geschwindigkeit, mit der sich der Agent bewegt
setRotation	Legt die Rotation des Agenten fest
getRotation, getVerticalRotation	Liefert die Rotation des Agenten in der XY/XZ-Ebene
isMoving	Gibt an, ob sich der Agent bewegt oder nicht
timeToArrival	Liefert die Zeit, die der Agent noch benötigt um sein Ziel zu erreichen
onArrival	Diese Methode kann überschrieben werden um Aktionen auszuführen wenn der Agent sein Ziel erreicht.
stop	Hält die Bewegung eines Agenten an
distanceTo	Liefert die Distanz zu einem anderen Agenten oder einer beliebigen Position

Tabelle 4.5: Beschreibungen zu ausgewählten Agentenmethoden die in Zusammenhang mit der räumlichen Umgebung stehen

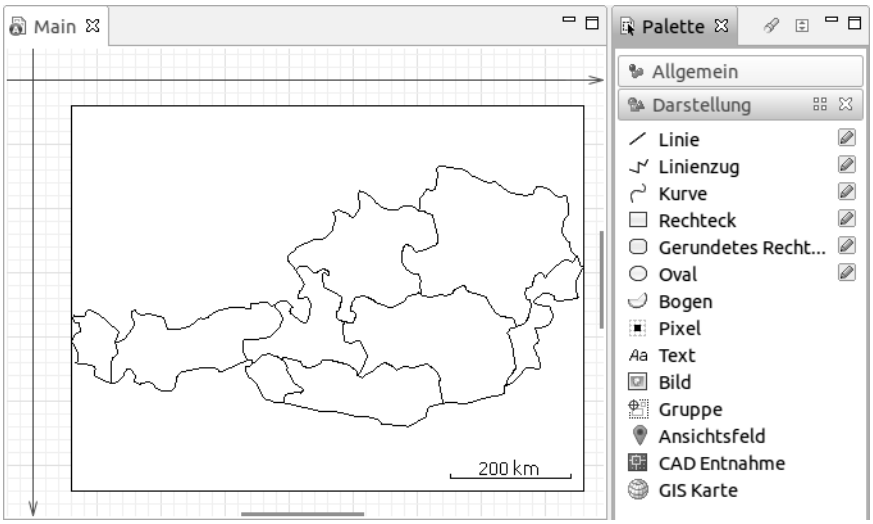

Abbildung 4.2: Hinzufügen einer GIS Karte. Auf der rechten Seite stehen verschiedene graphische Objekte zur Verfügung, die in das Modell eingefügt werden können, darunter auch die GIS Karte. Diese sieht wie links dargestellt aus.

Während normalerweise eine Umgebung als recheckiger Bereich dargestellt wird, in dem sich die Agenten befinden, muss für eine GIS-Umgebung eine Karte definiert werden, die die Repräsentation der Umgebung übernimmt. Diese Karte muss vom Typ *AbstractShapeGISMap* sein und ist nicht nur für die graphische Darstellung zuständig, sondern auch für die Berechnung von Distanzen, die Projektion der Agentenpositionen auf die Karte und die Verarbeitung von Benutzeraktionen wie Verschieben und Zoomen.

4.5.1 Vorhandene Schnittstelle zu OpenMap

Wie schon vorher erläutert gibt es in AnyLogic mit der Schnittstelle *AbstractShapeGISMap* ein allgemeines Konzept für Agenten in GIS-Umgebungen. AnyLogic selbst wird auch mit einer Implementierung dieser Schnittstelle ausgeliefert: *ShapeGISMap*. *ShapeGISMap* ist ein grafisches Objekt (*GIS Karte*), das aus der Palette „Darstellung" zum Modell hinzugefügt werden kann (Abb. 4.2). Unter den Eigenschaften der GIS Karte können dann verschiedene Shapefiles hinzugefügt werden (Abb. 4.3). Weitere Eigen-

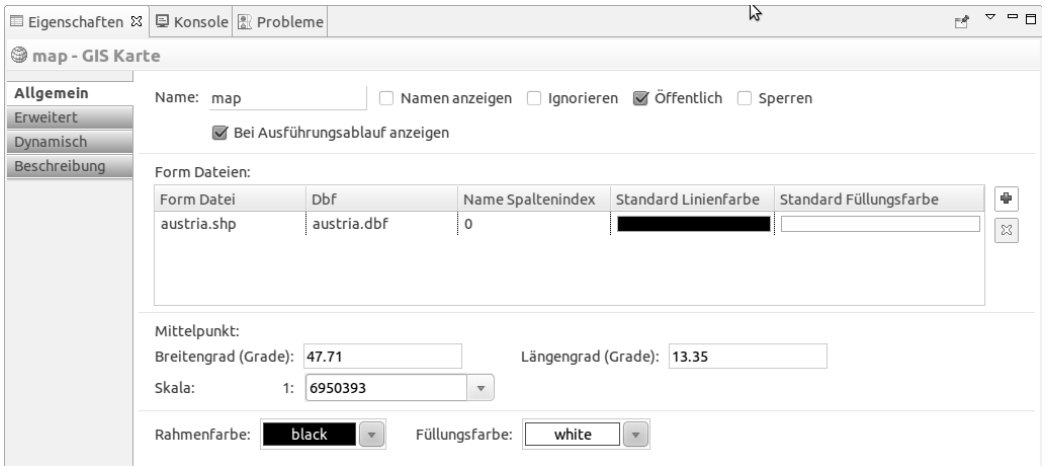

Abbildung 4.3: Eigenschaften der GIS Karte. Zur Liste der Form-Dateien können beliebig viele Shapefiles hinzugefügt werden, für die separat die Linienfarbe und Füllungsfarbe definiert werden kann. Der Mittelpunkt wird durch Angabe von Breiten- und Längengrad definiert und bezeichnet jenen Punkt, der in die Mitte der Karte projeziert wird.

schaften der Karte sind der Ausschnitt, der durch die Angabe des Mittelpunktes in Längen- und Breitengraden und der Skalierung festgelegt wird, sowie die Rahmenfarbe und die Hintergrundfarbe. Für jedes Shapefile kann separat auch die Linienfarbe und die Füllfarbe der Polygone definiert werden.

Mit *ShapeGisMap* ist es nur möglich Geodaten im Koordinatensystem WGS 84 anzuzeigen. Über die dabei verwendete Projektion gibt es in der Dokumentation von AnyLogic keine Angaben. Der Anzeige nach zu urteilen handelt es sich vermutlich um eine Mercator-Projektion. Diese entsteht durch Projektion der Erdoberfläche vom Erdmittelpunkt aus auf einen am Äquator berührenden umschließenden Zylinder. Diese Abbildung ist winkeltreu aber führt zu starken Verzerrungen abseits des Äquators. Daten in anderen Koordinatensystemen müssen zuerst in das WGS 84-System konvertiert werden um verwendet werden zu können.

4.5.2 Erweiterung durch Schnittstelle zu GeoTools

Um die Einschränkungen, die durch die vorhandene GIS-Integration in AnyLogic gegeben sind, aufzuheben, wurde eine weitere Implementierung der

GIS-Schnittstelle entwickelt, die die Bibliothek GeoTools anstatt OpenMap verwendet. Diese Implementierung heißt *ShapeGeoToolsMap*. Damit ist es nun möglich sowohl Shapefiles als auch PostGIS-Datenbanken als Datenquellen zu verwenden.

Integration von ShapeGeoToolsMap in AnyLogic

Wie sich *ShapeGeoToolsMap* in die Architektur von AnyLogic einfügt zeigt das Klassendiagramm in Abb. 4.4. In diesem Klassendiagramm sind die abstrakten Methoden in Superklassen aufgelistet, die implementiert werden mussten um eine annähernd ähnliche Handhabung von $ShapeGeoToolsMap$ im Vergleich zu $ShapeGISMap$ zu ermöglichen. Diese Methoden werden vor allem von der GIS-Umgebung verwendet. So ruft diese zum Beispiel $projectionContains$ auf um festzustellen, ob ein Agent in der aktuellen Projektion überhaupt enthalten ist, oder $convertXForward$ und $convertYForward$ um die Agenten-Koordinaten (Längen- und Breitengrade) in Bildschirmkoordinaten (X-, Y-Werte) umzurechnen. Andere Methoden wiederum ermöglichen es den Ausschnitt der Karte zu verändern (pan, $zoomIn$, $zoomOut$). Weiters ist aus dem Klassendiagramm noch ersichtlich, dass $ShapeGeoToolsMap$ eine Instanz von $ShapeRectangle$ benötigt. Der Grund dafür ist, dass AnyLogic keine Möglichkeit bietet eigene graphische Formen (Shapes) zur Palette „Darstellung" hinzuzufügen (siehe auch Abb. 4.2). Daher dient ein Rechteck während der Modellerstellung als Platzhalter für die Karte. Durch die Übergabe dieses Rechtecks im Konstruktor von $ShapeGeoToolsMap$ kann $ShapeGeoToolsMap$ dann die Eigenschaften dieses Rechtecks übernehmen: Position, Ausdehnung, Rotation, Skalierung, Hintergrund, Rahmenfarbe und -stärke.

Um die Geodaten in Form einer Karte anzeigen zu können sind eine Reihe von Koordinatensystemen und Transformationen zwischen diesen involviert. Eine schematische Darstellung dieses Transformationsvorgangs von den Geodaten bis zur Darstellung am Bildschirm findet sich in Abb. 4.5. Geographische Daten können dabei aus verschiedenen Quellen stammen, die in einzelnen Schichten, sogenannten Layern, in die Karte eingefügt werden

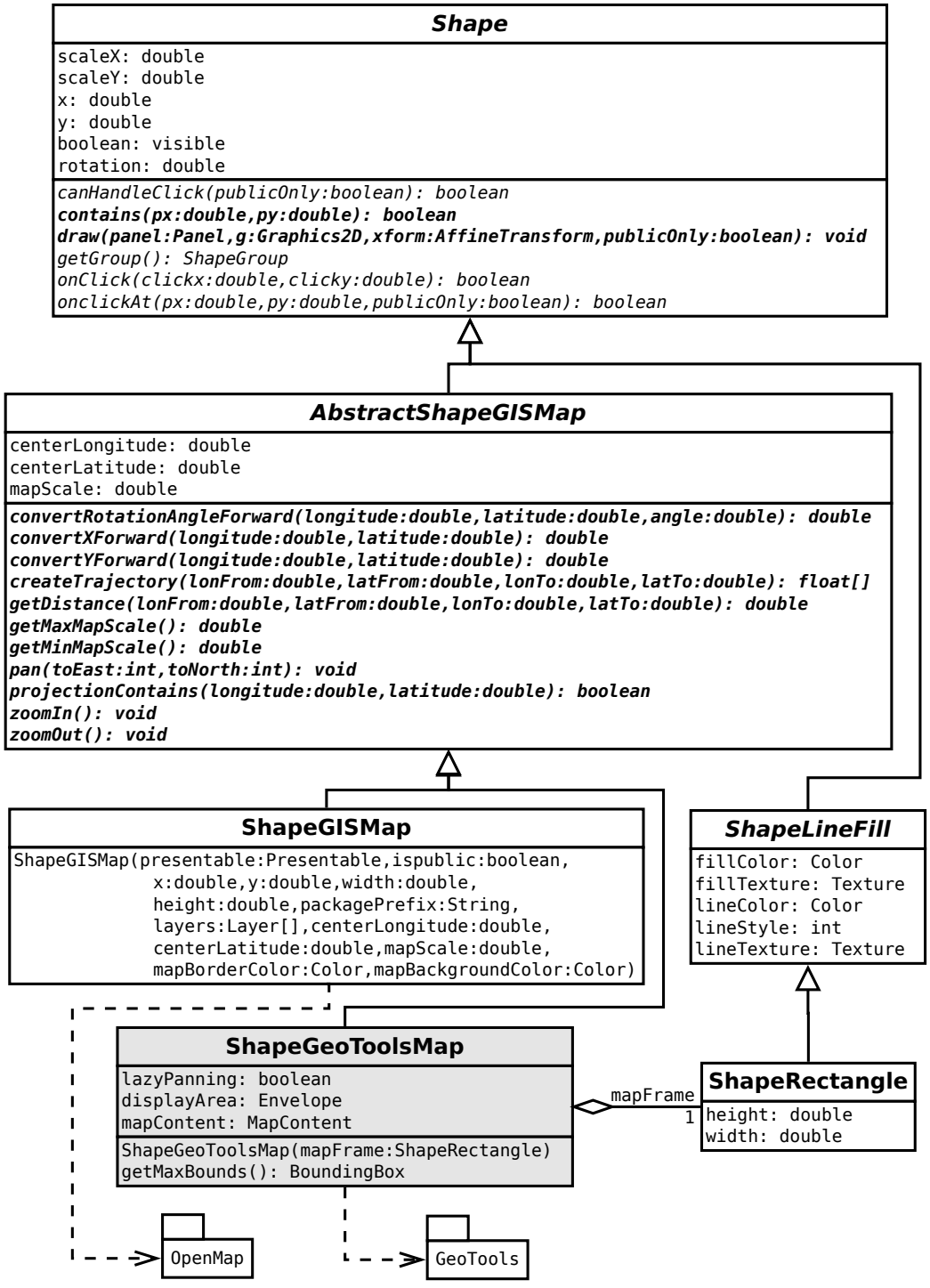

Abbildung 4.4: Klassendiagramm zu ShapeGeoToolsMap. ShapeGeoToolsMap fügt sich in das AnyLogic Framework ein und verwendet im Gegensatz zu ShapeGISMap die Bibliothek GeoTools. Die Klasse ShapeRectangle wird zur Initialisierung und Darstellung verwendet.

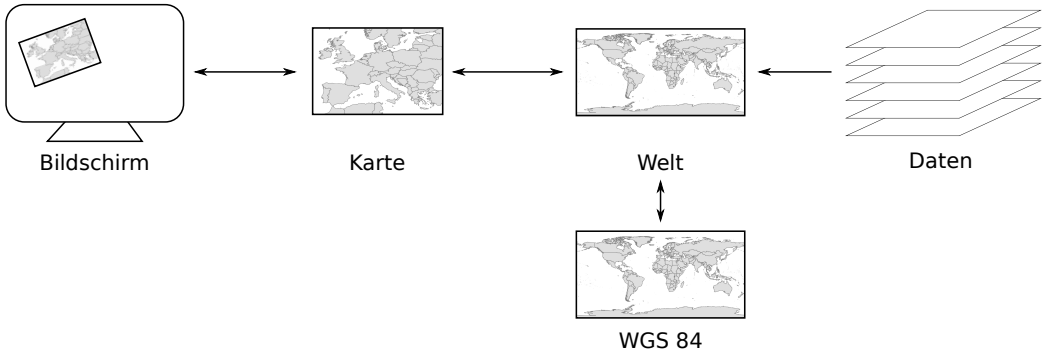

Abbildung 4.5: Transformationen von den Geodaten bis zur Darstellung. Jede Datenquelle kann ein eigenes Koordinatensystem haben, welche bei der gemeinsamen Darstellung in ein Weltkoordinatensystem umgerechnet werden müssen. Dieses Weltkoordinatensystem wird dann mit einer Projektion auf eine Karte abgebildet und schließlich am Bildschirm angezeigt. Die Agenten bewegen sich immer im WGS 84-System. Dargestellt werden sie aber dennoch auf der gleichen Karte wie die Geodaten, wodurch eine Umrechnung zwischen dem Weltkoordinatensystem und dem WGS 84-System notwendig ist.

können. Jeder Layer hat dabei sein eigenes Koordinatensystem. Zur gemeinsamen Darstellung der Daten müssen diese in ein Weltkoordinatensystem umgerechnet werden. Dieses Weltkoordinatensystem umfasst, noch immer die gesamte Erdoberfläche und auch die Maßeinheiten können hier noch beliebig gewählt sein (z.B. Grad, Meter oder Kilometer). Da der Benutzer meistens nur einen kleinen Bereich der Karte betrachtet, müssen für die Darstellung am Bildschirm der Ausschnitt berechnet und die Einheiten in Pixel umgerechnet werden. Die Karte kann am Bildschirm bewegt, rotiert und skaliert werden und aus diesem Grund findet zuletzt eine affine Transformation der Karte statt, die diesen Operationen entspricht. Zusätzlich zu den bisher genannten Koordinatensystemen, spielt auch das Koordinatensystem „WGS 84“ eine wichtige Rolle, da sich die Agenten in diesem System bewegen. Ihre Position wird immer in Längen- und Breitengraden gemessen. Dieses zusätzliche Koordinatensystem und die damit verbundenen Transformationen könnte man sich ersparen, wenn man das Weltkoordinatensystem gleich als „WGS 84” setzt. Das würde aber bedeuten, dass auch die Darstellung immer in diesem System stattfindet. Manchmal ist es aber erwünscht ein

anderes Referenzsystem zur Darstellung zu verwenden, weil dieses besondere Eigenschaften hat. Zum Beispiel könnte in einem speziellen Referenzsystem die Verzerrung in der betrachteten Region geringer sein oder die Darstellung von Rasterdaten achsenparallel erfolgen.

Verwendung von ShapeGeoToolsMap

ShapeGeoToolsMap wird als Archiv-Datei zur Verfügung gestellt. Zur Verwendung muss dieses Archiv in den Modell-Ordner entpackt werden. Das Archiv enthält die Java-Bibliothek „ShapeGeoToolsMap.jar“ sowie die benötigten GeoTools Dateien. Zur Verwendung muss die ShapeGeoTools-Bibliothek zu den Abhängigkeiten des Modells hinzugefügt werden. Weiters müssen dort auch mindestens die Dateien *gt-swing-*.jar*[3] und *gt-epsg-hsql*.jar* zum Klassenpfad hinzugefügt werden. Dabei ist darauf zu achten, dass diese Dateien nicht erneut in den Modellordner importiert werden (siehe Abb. 4.6). Für verschiedene Datenquellen können auch zusätzliche Bibliotheken notwendig sein. So benötigt man zum Beispiel für Shapefiles die Datei *gt-shapefile-*.jar* oder für den Zugriff auf Daten in einer PostGIS-Datenbank die Datei *gt-jdbc-postgis-*.jar*.

Wie schon in den vorherigen Abschnitten erwähnt, benötigt *ShapeGeoToolsMap* ein Objekt vom Typ *ShapeRectangle*. Dieses muss nun erzeugt und im Modell an der Stelle platziert werden, an der dann die Karte angezeigt werden soll. Im Folgenden wird dieses Objekt *mapFrame* genannt. Außerdem wird auch eine Variable für *ShapeGeoToolsMap* benötigt. Diese Variable, im Folgenden *gisMap* genannt, wird mit einem neuen *ShapeGeoToolsMap*-Objekt instanziert, dem der *mapFrame* übergeben wird. Diese Variable kann entweder über den graphischen Editor von AnyLogic hinzugefügt oder im „Zusatzklassen Code“ wie hier gezeigt eingefügt werden:

```
private ShapeGeoToolsMap gisMap = new ShapeGeoToolsMap(mapFrame);
```

Damit sich die Agenten im geographischen Raum bewegen können, muss sowohl der Typ der Umgebung als auch der Typ der Agenten auf GIS umgestellt werden. Zusätzlich muss bei der Umgebung auch noch die Variable

[3] Diese Dateien enthalten im Dateinamen auch immer eine Versionsinformation, die hier zwecks der Allgemeingültigkeit durch einen Stern (*) ersetzt wird.

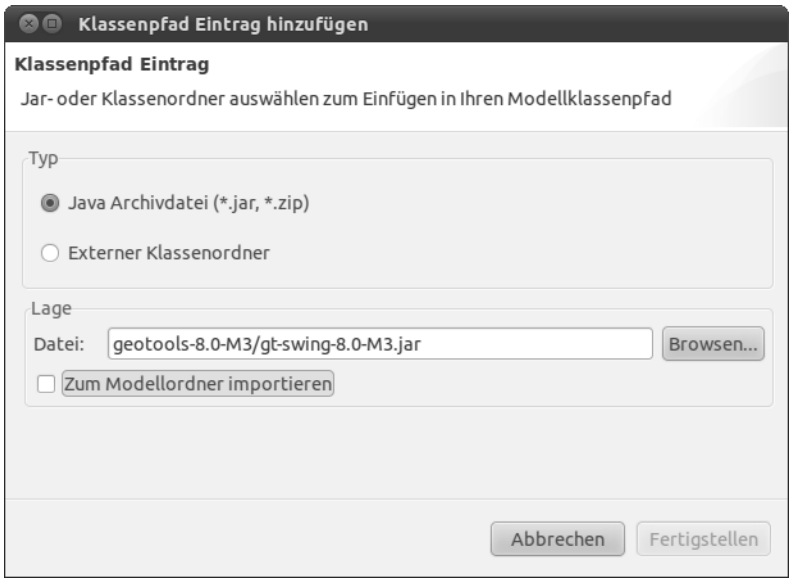

Abbildung 4.6: Abhängigkeiten zum Modell hinzufügen

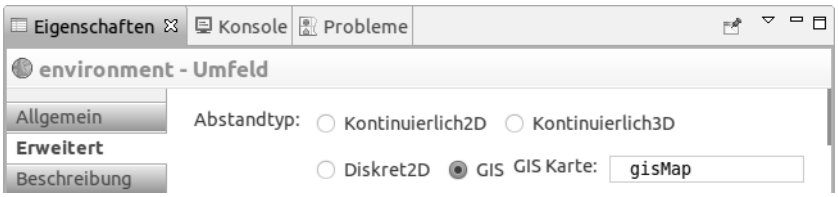

Abbildung 4.7: Einstellung für den Umgebungs-Typ

der *ShapeGeoToolsMap*-Instanz *gisMap* eingetragen werden (Abb. 4.7 und Abb. 4.8). Damit die Position der Agenten während der Simulation mit der Karte übereinstimmt, muss im Zuge der Modell-Erstellung der Ursprung des Agenten-Koordinatensystems mit dem der Karte übereinstimmen. Dazu platziert man die Agenten an der linken oberen Ecke der Karte (*mapFrame*), wie in Abb. 4.9 gezeigt.

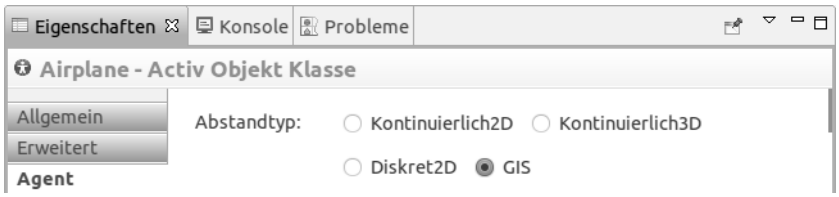

Abbildung 4.8: Einstellung für die Agenten

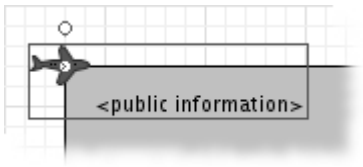

Abbildung 4.9: Das Präsentations-Objekt der Agenten (hier Darstellung eines Flugzeugs) muss während der Modell-Erstellung an der Karte (grauer Bereich) ausgerichtet werden und zwar so, dass dessen Mittelpunkt mit der linken oberen Ecke der Karte übereinstimmt. Dadurch wird sichergestellt, dass die Agenten während der Simulation an der richtigen Stelle über der Karte angezeigt werden.

Das einzige, das jetzt noch fehlt, ist der Karteninhalt. Abhängig vom Typ der Datenquelle können die Schritte, die dafür notwendig sind, anders aussehen. Allgemein kann man den Ablauf jedoch so beschreiben:

- Datenquelle öffnen
- *MapContent* erstellen
- *Layer* mit Datenquelle erstellen
- *Layer* zum *MapContent* hinzufügen
- *MapContent* zur Karte hinzufügen

Diese Schritte fügt man am besten im Anlaufscode des Modells ein. Für verschiedene Datenquellen kann es notwendig sein zusätzliche Bibliotheken den Abhängigkeiten des Modells hinzuzufügen, wie am Anfang dieses Abschnitts erwähnt. Für die Datenquellen Shapefile und PostGIS-Datenbank werden hier nun konkrete Beispiele gegeben.

Shapefile Ein Shapefile kann über einen *FileDataStore* geöffnet werden.

```
URL url = new URL(
    "file:///home/user/Models/GIS Example/world_borders.shp");
FileDataStore shpStore = FileDataStoreFinder.getDataStore(url);
SimpleFeatureSource shpFeatureSource = shpStore.getFeatureSource();

MapContent mapContent = new MapContent();
Style style = SLD.createPolygonStyle(black, antiqueWhite, 1);
FeatureLayer layer = new FeatureLayer(shpFeatureSource, style);
mapContent.addLayer(layer);

gisMap.setMapContent(mapContent);
```

Listing 4.5: Öffnen, Einlesen und Anzeige eines Shapefiles mit *ShapeGeoToolsMap*

PostGIS-Datenbank Um Daten aus einer PostGIS-Datenbank einzubinden muss man zuerst den Datenbanktreiber laden. Dann benötigt man die Zugangs- und Verbindungsdaten um eine Verbindung herzustellen. Schließlich kann man mit dem so erstellten *DataStore* einzelne Tabellen aus der Datenbank auslesen.

```
Class.forName("org.postgresql.Driver");

Map params = new HashMap();
params.put("dbtype", "postgis");
params.put("host", "localhost");
params.put("port", new Integer(5432));
params.put("database", "osm");
params.put("user", "gis");
params.put("passwd", "atlas");
DataStore pgDatastore = DataStoreFinder.getDataStore(params);
SimpleFeatureSource boundaries = pgDatastore.getFeatureSource("boundaries");
```

```
MapContent mapContent = new MapContent();
Style style = SLD.createPolygonStyle(black, antiqueWhite, 1);
FeatureLayer layer = new FeatureLayer(boundaries, style);
mapContent.addLayer(layer);

gisMap.setMapContent(mapContent);
```

Listing 4.6: Auslesen und Anzeigen von PostGIS-Daten mit *ShapeGeoToolsMap*

4.5.3 Vergleich der beiden Schnittstellen

Um die von AnyLogic zur Verfügung gestellte Implementierung der GIS-Schnittstelle *ShapeGISMap* mit jener neu entwickelten und auf GeoTools basierenden Implementierung *ShapeGeoToolsMap* zu vergleichen, wurde das Beispielmodell „GIS Example“, das AnyLogic mit der Installation ausliefert, auf *ShapeGeoToolsMap* umgestellt.

Die Agenten in diesem Modell sind Flugzeuge, die anfangs eine zufällig gewählte Start- und Zielposition zugewiesen bekommen. Dann bewegen sich die Flugzeuge entlang geodätischer Pfade, die durch die GIS-Karte berechnet werden, zu ihren Zielorten. Haben sie diesen erreicht wählen sie erneut einen zufälligen Zielort. Man kann die Simulation auch in einen manuellen Modus versetzen, so dass man durch Auswahl eines Flugzeuges und Klicken in die Karte dessen Zielflughafen festlegen kann. Zu jedem Flugzeug wird der Name und dessen aktueller Standort angezeigt. Zum ausgewählten Flugzeug, das rot dargestellt wird, werden noch weitere Informationen wie Rotation oder Geschwindigkeit angezeigt. Mit den gelben Pfeilen in der linken oberen Ecke, sowie mit den Balken rechts und unten kann die Karte verschoben werden. Die Lupen links oben dienen zur Veränderung der Kartenskalierung.

Tabelle 4.6 listet die Unterschiede und Gemeinsamkeiten der Implementierungen auf. Während der Ausführung der beiden Modelle ergeben sich kaum Unterschiede. Einzig die etwas unterschiedliche Projektion und Skalierung fallen im Vergleich der Modelle auf (Abb. 4.10 und Abb. 4.11). Au-

ßerdem ist es bei der Implementierung durch *ShapeGeoToolsMap* möglich die Karte direkt mit der Maus zu verschieben und durch Bewegen des Mausrades hinein- oder hinauszuzoomen.

	ShapeGISMap	**ShapeGeoToolsMap**
GIS-Bibliothek	OpenMap	GeoTools
Erdmodelle und Koordinatensysteme		
Daten-Koordinatensystem	WGS 84	beliebig
Anzeige-Koordinatensystem	WGS 84	beliebig
Erdmodell für Projektion	Kugel	projektionsspezifisch
Erdmodell für Geodäten[1]	Kugel	WGS 84
Datenformate		
Shapefiles	ja	ja
PostGIS-DB	nein	ja
Interaktion (Verschieben, Zoomen, Klicken)		
Funktionale Steuerung (Programmatorisch, Controls)	ja	ja
Maussteuerung	nein	ja
Mausklick-Aktionen	ja	ja (mit *mapFrame*)
Zurücksetzen der Karte	nein	ja (Doppelklick)
Modifikation der Karte zur Laufzeit		
Hinzufügen/Entfernen von Daten/Layer	nein	ja
Projektion	nein	ja
Anzeige von Agenten		
Bewegung entlang von Geodäten	ja	ja
Rotation entsprechend der Bewegung	ja	ja
Darstellung der Karte		
Anzeige des Maßstabs	ja	nein
Anzeigeeigenschaften	Füll- und Linienfarbe	Stylesheets

[1] Die Geodäten werden benötigt um die Bewegungen der Agenten zu steuern. ShapeGeoToolsMap verwendet das Sphäroid des WGS 84-Referenzsystems, da dieses auch die Abplattung an den Polen berücksichtigt.

Tabelle 4.6: Vergleich von ShapeGISMap und ShapeGeoToolsMap

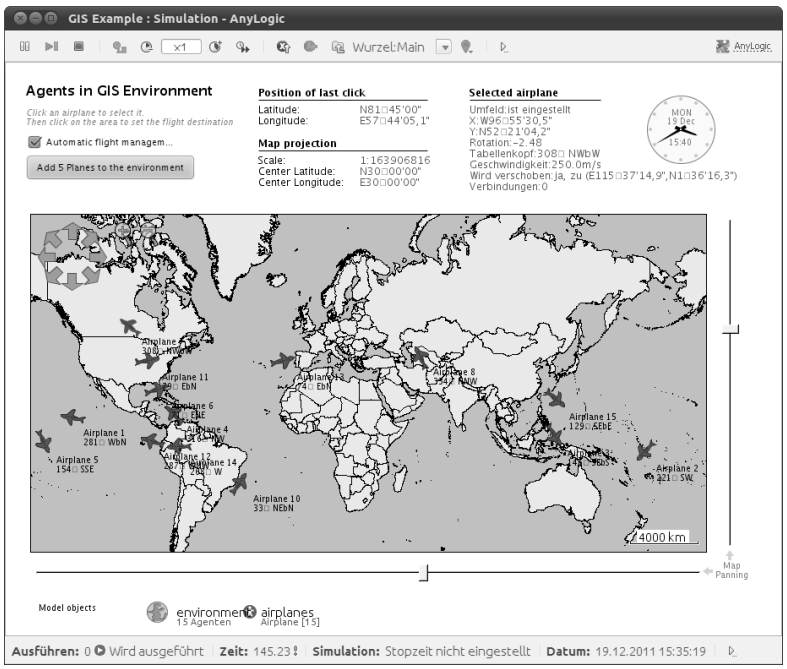

Abbildung 4.10: GIS-Beispielmodell mit ShapeGISMap

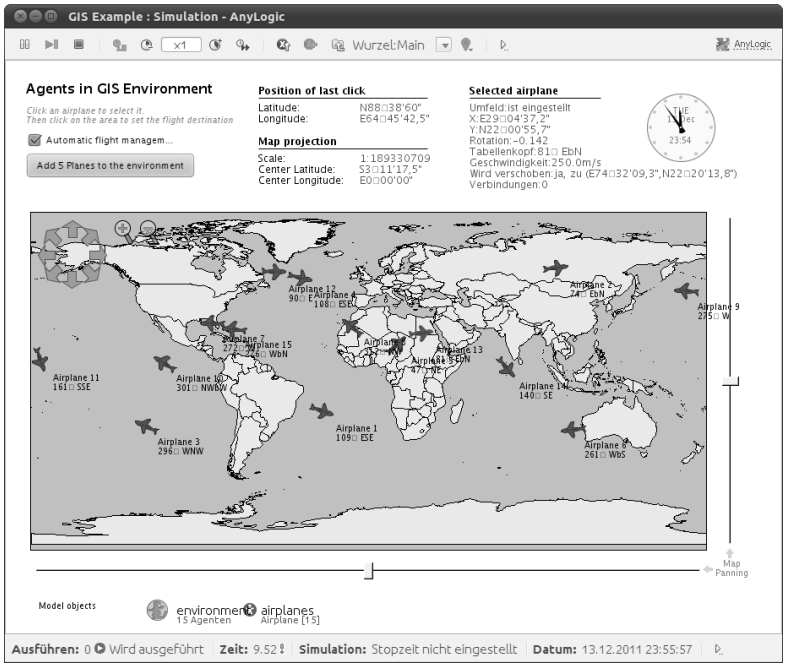

Abbildung 4.11: GIS-Beispielmodell mit ShapeGeoToolsMap

Kapitel 5

Allgemeine Problemstellungen mit GIS-Daten

5.1 Initialisierung

Sind die Positionen von Agenten bereits durch Daten fest vorgegeben, so ist die Initialisierung trivial. Dies ist beispielsweise bei Krankenhäusern, Schulen oder Tankstellen der Fall. Handelt es sich bei den Agenten um Individuen, für die nur eine bestimmte Verteilung in einer Region bekannt ist, so stellt sich das Problem, wie man die einzelnen Agenten initialisiert, so dass die Verteilung der Agenten der durch die Daten vorgegebenen Verteilung entspricht. Im Modell des österreichischen Gesundheitssystems (Kapitel 6) sind die Agenten Personen, die entsprechend der Populationsverteilung, die in Form von Rasterdaten vorliegt, initialisiert werden sollen. Dabei soll die Einschränkung, dass Agenten nur im Dauersiedlungsraum positioniert werden dürfen, berücksichtigt werden. Allgemein könnte man das Problem so formulieren: Ausgehend von einer Häufigkeitsverteilung, sollen Agenten in einer ausgezeichneten Region (der Zielregion) so positioniert werden, dass ihre Verteilung der gegebenen entspricht. Dabei ist zu beachten, dass die Verteilung außerhalb der Zielregion konstant gleich Null sein muss. Die erwähnte Häufigkeitsverteilung wird durch die Anzahl der Personen in den Zellen eines Rasters bestimmt.

Für die Lösung des Problems ist ein Algorithmus gesucht, der die Position eines Agenten berechnet. Bei mehrmaliger Anwendung dieses Algorithmus, sollen die so erzeugten Positionen der gegebenen Verteilung entsprechen. Innerhalb der Rasterzellen sollen die Agenten gleichverteilt positioniert werden. Das Problem kann in zwei Teile zerlegt werden:

1. Auswahl einer Rasterzelle

2. Berechnung einer Position innerhalb der Rasterzelle

Da für jede Rasterzelle c_i, die Anzahl der Personen $n(c_i)$, die in diesem Raster leben, bekannt ist, kann man sich für jede Rasterzelle eine Wahrscheinlichkeit (5.1) berechnen und eine Auswahl entsprechend diesen Wahrscheinlichkeiten treffen.

$$P(c_i) = \frac{n(c_i)}{\sum_j n(c_j)} \tag{5.1}$$

Die Berechnung einer Position innerhalb der Rasterzelle c_i unter Berücksichtigung eines einschränkenden Polygons e entspricht dem Problem, einen beliebigen Punkt innerhalb eines oder mehrerer Polygone φ_j zu finden, wobei die Polygone φ_j aus dem Schnitt von c_i mit e entstehen. Die Aufgabe zu entscheiden, ob ein Punkt in einem Polygon enthalten ist oder nicht, wird auch Point-in-Polygon (PIP) Problem genannt. Für die Positionierung der Agenten werden nun zwei Lösungen präsentiert.

5.1.1 Positionierung mittels Point-in-Polyon Tests

Die naive Lösung ist, aus dem kleinsten umschließenden Rechteck – das kleinste Rechteck, das alle Polygone φ_j enthält – einen zufälligen Punkt auszuwählen und zu überprüfen, ob sich dieser Punkt in einem der Polygone φ_j befindet. Schlägt dieser Test fehl, wird erneut ein zufälliger Punkt gewählt und die Prozedur wiederholt, bis ein Punkt gefunden ist, der den Kriterien entspricht. Es ist leicht zu erkennen, dass die Anzahl der durchzuführenden Tests direkt vom Verhältnis des Flächeninhalts des umschließenden Rechtecks zum gesamten Flächeninhalt der Polygone φ_j abhängt. Wie das Point-in-Polygon (PIP) Problem gelöst werden kann und welche Aufwände damit

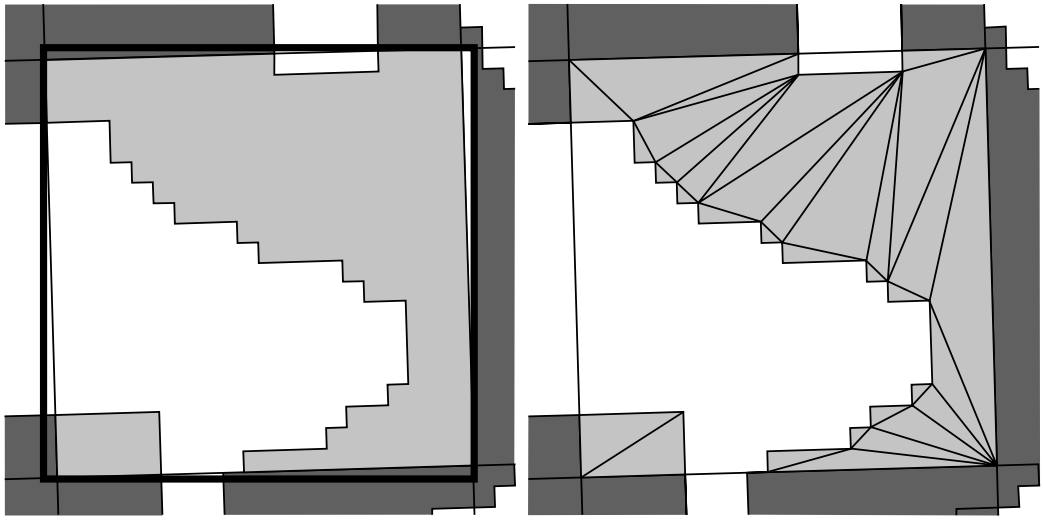

Abbildung 5.1: Relevante Regionen zur Positionierung von Agenten.
Links: PIP-Methode. Rechts: Triangulierungsmethode

verbunden sind, kann in „Point in Polygon Strategies" [L15] oder „Geometric Tools for Computer Graphics" [L34, S.695ff.] nachgelesen werden. Auch wenn manche Algorithmen zur Lösung des PIP-Problems nur einen geringen Aufwand haben, so kann es doch nötig sein, diesen Test oftmals zu wiederholen, bis ein Punkt innerhalb der Polygone gefunden ist. Abb. 5.1 (a) zeigt die bei dieser Methode involvierten Regionen.

5.1.2 Positionierung mittels Triangulierung

Diese Methode benutzt eine zuvor erstellte Triangulierung der Polygone, um zufällige Positionen innerhalb dieser zu ermitteln. Das Erstellen der Triangulierung benötigt zwar einen zusätzlichen Aufwand, im Vergleich zur Berechnung der Schnittmenge zwischen der Rasterzelle und der Zielregion ist dieser aber vernachlässigbar. Für eine erste Implementierung wurde der Cutting-Ear-Algorithmus zur Triangulierung verwendet. Dabei wird von dem Polygon solange ein Ohr abgeschnitten, bis das Polygon vollständig trianguliert ist.

Ein Polygon-Ohr ist in diesem Fall ein Dreieck, das von drei aufeinanderfolgenden Punkten des Polygons gebildet wird und ganz im Inneren des Polygons liegt [L34, S.772-775]. Abb. 5.1 (b) zeigt die so erstellte Triangulierung der Zielregion. Nach der Erstellung der Triangulierung wird jedes Dreieck mit seinem Flächeninhalt gewichtet. Um nun einen Punkt innerhalb des Polygons auszuwählen, wählt man zufällig, unter Berücksichtigung der Gewichte, ein Dreieck aus und innerhalb dessen einen zufälligen Punkt. Baryzentrische Koordinaten eignen sich dazu am Besten.

Jeder Punkt p innerhalb eines Dreiecks kann eindeutig als eine Affinkombination (Gleichungen 5.2 und 5.3) der Eckpunkte (A, B, C) dargestellt werden, so dass die *Baryzentrischen Koordinaten* (α, β, γ), alle im Intervall $[0, 1]$ liegen.

$$p = \alpha \cdot A + \beta \cdot B + \gamma \cdot C \tag{5.2}$$

$$1 = \alpha + \beta + \gamma \tag{5.3}$$

Ein erster Ansatz zur Generierung eines zufälligen Punktes mittels baryzentrischer Koordinaten könnte folgender sein: Erzeugung von drei Zufallszahlen und Normierung dieser entsprechend der Gleichung 5.3. Auf diese Weise erhält man aber keine im Dreieck gleichverteilten Punkte, sondern die Verteilung sieht wie in Abb. 5.2 (a) aus. Richtigerweise werden nur nur zwei Zufallszahlen im Intervall $[0, 1]$ erzeugt und die dritte Koordinate γ ergibt sich aus der Gleichung 5.3. Ist γ negativ, so liegt p außerhalb des Dreiecks (Abb. 5.2 (b)). Dieser Fehler kann korrigiert werden, indem p am Halbierungspunkt der Seite AB gespiegelt wird, wie in Gleichung 5.4 gezeigt (Abb. 5.2 (c)).

$$(\alpha', \beta', \gamma') = (1, 1, 0) - (\alpha, \beta, \gamma) \tag{5.4}$$

5.1.3 Vergleich der beiden Positionierungsmethoden

Zum Vergleich der beiden Methoden wurden einerseits die Rechenzeiten für die initialen Berechnungen, wie den Schnitt zwischen Rasterzellen und Ziel-

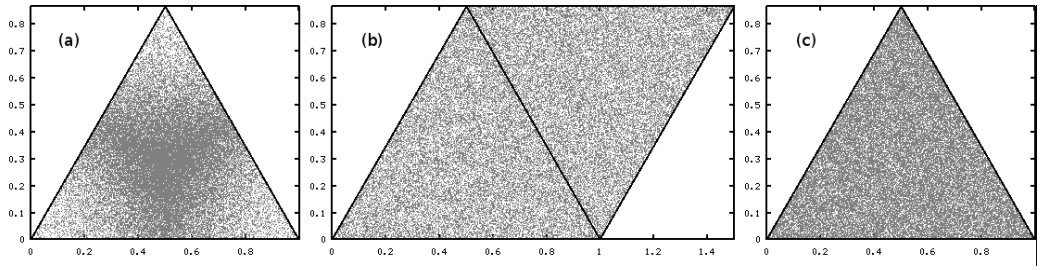

Abbildung 5.2: Punktverteilung mittels Baryzentrischer Koordinaten
Ergebnisse bei zufälliger Wahl
(a) aller drei baryzentrischen Koordinaten und entsprechender Normierung gemäß Gleichung 5.3
(b) von nur zwei baryzentrischen Koordinaten und Berechnung der dritten aus der Gleichung 5.3
(c) von nur zwei baryzentrischen Koordinaten und Berechnung der dritten aus der Gleichung 5.3 mit zusätzlicher Behandlung der außenliegenden Punkte

region, sowie die Triangulation und andererseits für die Positionierung verschiedener Agentenanzahlen gemessen. Für alle folgenden Ergebnisse wurde ein Rechner mit einem Intel® Core™ i7-2620M Prozessor und 4GB RAM verwendet.

Die Daten, die für die Berechnungen verwendet wurden, sind einerseits die Populationsdaten der Statistik Austria mit dem zugehörigen MGI-Lambert-Raster und andererseits als Zielregion für die Agenten der Dauersiedlungsraum Österreichs.

Die anfängliche Berechnung der Schnittmengen dauerte im Durchschnitt 54 Sekunden. Diese ist von der Anzahl der Agenten unabhängig. Der zusätzliche Aufwand der Triangulation wirkte sich nur minimal aus, so dass sich für die zweite Methode eine Rechenzeit von 55 Sekunden ergibt. Die Messungen zur Berechnung der Agentenpositionen ohne Initialaufwand sind zusammen mit den Regressionsgeraden in Abbildung 5.3 zu sehen. Die Triangulationsmethode benötigt nur in etwa 1/5 der Zeit der PIP-Methode.

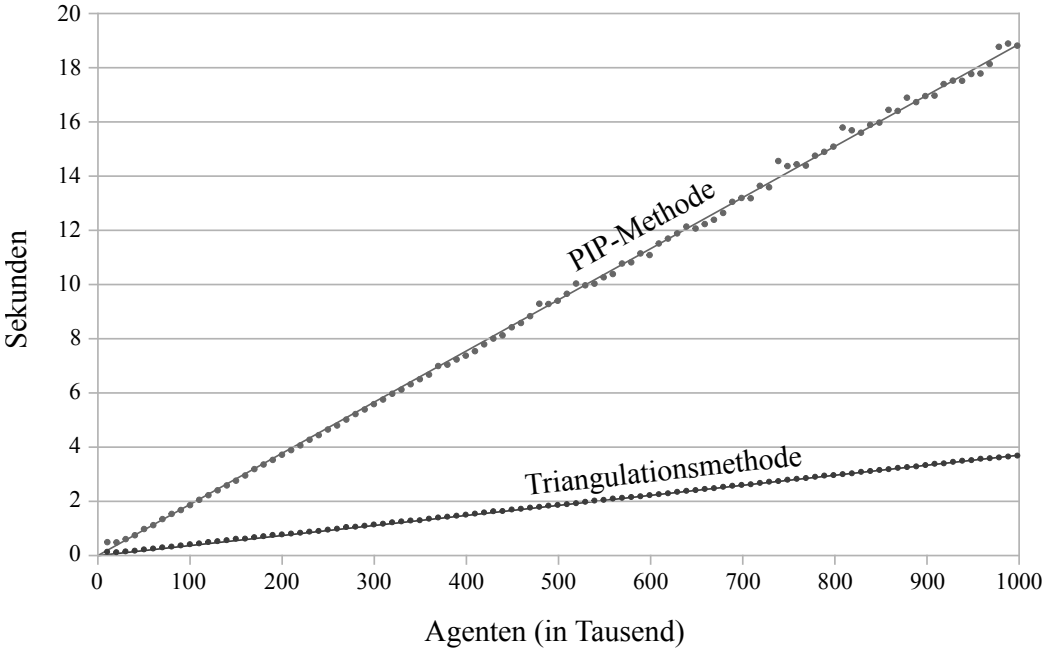

Abbildung 5.3: Laufzeiten zur Positionierung der Agenten. Auf der x-Achse ist die Anzahl der Agenten aufgetragen und auf der y-Achse die Dauer der Positionierung mit beiden Methoden.

5.2 Berechnung von Distanzen zwischen Agenten und geographischen Punkten

5.2.1 Grundlagen

Die Berechnung von Entfernungen zu oder von Agenten ist eine unvermeidbare Aufgabe in der agentenbasierten Modellierung mit Geodaten. Würden Distanzen zwischen Agenten und geographischen Punkten, seien das andere Agenten oder sonstige für das Modell relevante Punkte, keine Rolle spielen, dann könnte man die Agenten auch zufällig verteilen und würde so die Verwendung von GIS-Daten ad absurdum führen. Entfernungen können auf verschiedenste Weise berechnet und gemessen werden:

- in der Kartenprojektion
- in einem Modell der Erdoberfläche
- in einem Netzwerk auf der Erdoberfläche

Die Messung von Entfernungen direkt in der Karte, also nach der *Projektion*, ist am ungenauesten, da sich sämtliche Verzerrungen, die durch die Projektion auftreten, in den Messergebnissen wiederspiegeln.

Zur Distanzberechnung mittels eines *Modells der Erdoberfläche* benötigt man zuerst ein geeignetes Modell. Eine Kugel hat den Vorteil, dass die kürzesten Verbindungen (Geodäten) zweier Punkte immer auf einem Großkreis liegen und sich so leicht berechnen lassen. Bei Verwendung eines Rotationsellipsoids ist dies ungleich schwieriger. Abb. 5.4 stellt die Entfernungen von Wien im Raum Österreich dar, die mit dem WGS84-Ellipsoid berechnet wurden. Entfernungen können auch entlang kürzester Wege in *Netzwerken* wie dem Straßennetz oder einem Schienennetz berechnet werden. Dabei kann die Distanz zwischen zwei Knoten wieder auf geodätischen Linien gemessen werden. Es ist aber auch möglich ein anderes Maß zu verwenden, wie zum Beispiel die Zeit, die benötigt wird, um sich von Punkt A nach Punkt B zu bewegen.

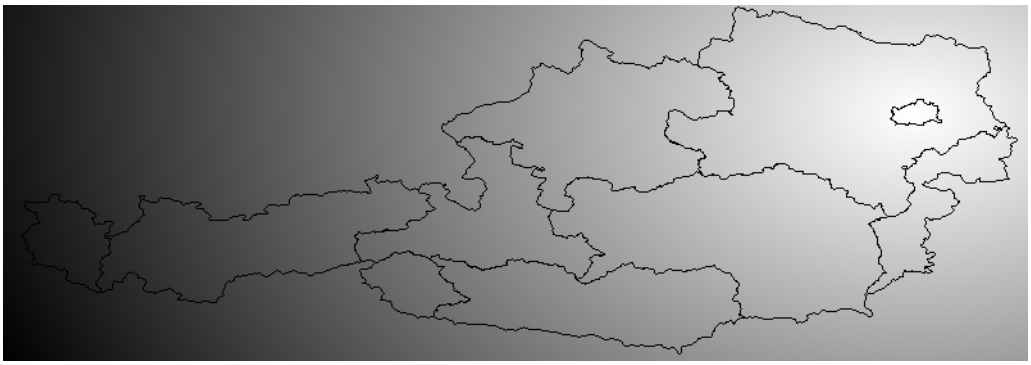

Abbildung 5.4: Entfernungen von Wien im Raum Österreich entlang geodätischer Linien. Ein dunklerer Grauwert bedeutet eine größere Entfernung.

Mit der Genauigkeit der drei genannten Methoden erhöht sich auch deren Komplexität und somit der benötigte Rechenaufwand. Besonders bei der Verwendung von Wegenetzen steigt der Aufwand so stark an, dass diese Berechnungsmethode für die Modellierung inpraktikabel wird. Die Berechnung aller für Abbildung 5.4 benötigten geodätischen Distanzen ($800 \times 277 = 221600$) dauerte nur 826 ms. Im Gegensatz dazu dauerte die Berechnung der $127 \times 45 = 5715$ Routen in Abbildung 5.6 37 Minuten. Damit dauert die Berechnung einer Distanz in einem Wegenetz 104214 mal länger als die Berechnung der entsprechenden geodätischen Entfernung. In den folgenden Abschnitten wird nun eine Methode beschrieben, mit welcher der Aufwand während der Modellierung durch eine einmalige Vorausberechnung ersetzt und approximierte Entfernungen in Netzwerken für die Modellierung verwendet werden können.

5.2.2 Approximation von Distanzen in Netzwerken

Aufgabenstellung

Es soll in effizienter Weise die Distanz entlang eines gegebenen Netzwerkes zwischen zwei beliebigen Punkten einer geographischen Region berechnet werden. Das Netzwerk soll sich über die gesamte betrachtete Region spannen.

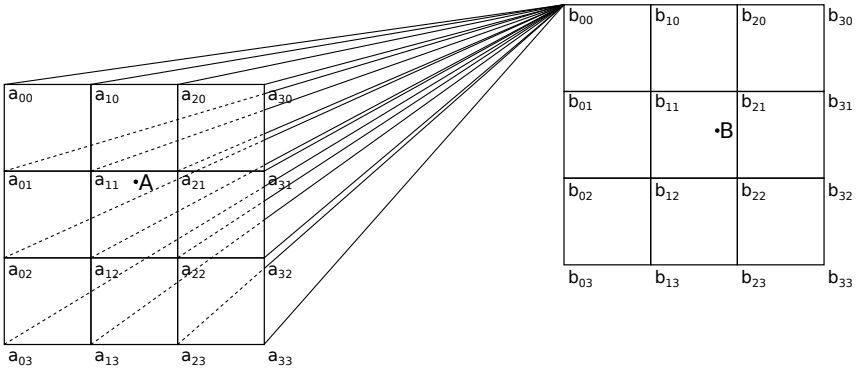

Abbildung 5.5: Die Distanzen von Punkt A zu den Punkten b_{ij} kann durch Interpolation durch die Punkte a_{ij} bestimmt werden. Schließlich kann die Entfernung von Punkt A nach Punkt B durch erneute Interpolation der so gewonnenen Distanzen zu den Punkten b_{ij} approximiert werden.

Methode

Wir definieren ein Raster über der Region und berechnen paarweise alle Distanzen der Knoten des Rasters. Je feiner das Raster ist, desto genauer ist die Entfernungsapproximation. Um schließlich die Entfernung zweier Punkte A und B innerhalb des Rasters zu bestimmen , kann wie folgt vorgegangen werden: Punkt A liegt innerhalb des Raster. Deshalb gibt es in einer Umgebung von A Knotenpunkte a_{ij} des Rasters. Die Punkte a_{ij} sollen Nachbarpunkte von A heißen. Ebenso hat Punkt B Nachbarpunkte b_{mn}. Da alle paarweisen Distanzen des Rasters vorausberechnet wurden, sind die Distanzen $d(a_{ij}, b_{mn})$ zwischen den Nachbarn von A und jenen von B bekannt. Die Entfernung von Punkt A zu einem bestimmten Punkt b aus der Menge der Nachbarpunkte b_{mn} von B kann durch die Distanzen $d(a_{ij}, b)$ interpoliert werden (Abb 5.5. Somit erhält man sämtliche Distanzen $d(A, b_{mn})$. Interpolation im Punkt B mittels dieser neu gewonnenen Entfernungen ergibt schließlich die Distanz zwischen A und B. Wieviele und welche Nachbarpunkte man verwendet hängt von der verwendeten Interpolationsmethode ab. Für die Implementierung in dieser Arbeit wurden Catmull-Rom-Splines [L16] zur Interpolation benutzt, die mit vier Punkten vollständig spezifiziert sind. In zwei Dimensionen werden somit 16 Punkte benötigt (Abb. 5.5).

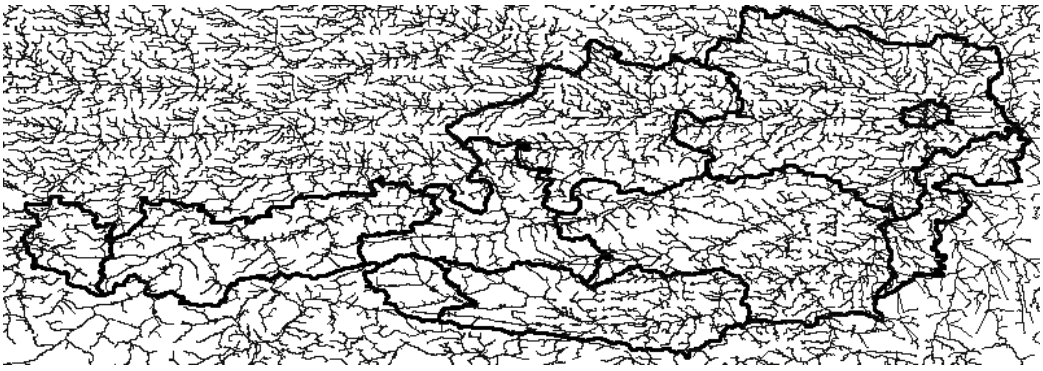

Abbildung 5.6: Zeitlich kürzeste Routen nach Wien ausgehend von einem Raster mit einer 1/16° Unterteilung. Die Grenzen der Bundesländer sind zur Orientierung fett eingezeichnet. Berechnungsdauer: 35 Minuten

Ergebnisse

Die im vorigen Abschnitt präsentierte Methode wurde im Gebiet Österreich mit dem Straßennetz von OpenStreetMap getestet. Als Raster wurden verschieden feine Unterteilungen des Gradnetzes verwendet. Mit dem Routing-Programm Gosmore [W18] wurden die Entfernungen in dem Straßennetz berechnet. Dabei wurden nicht die kürzesten sondern die schnellsten Verbindungen gesucht und die Distanzen entsprechend in Sekunden gemessen. Abbildungen 5.6, 5.7 und 5.8 zeigen Ergebnisse dieser Berechnungen. Aus diesen geht hervor, dass die Berechnung der Routen einen erheblichen Rechenaufwand bedeutet. Definiert man einen Raster durch das Gradnetz auf der Erdkugel mit einer Maschenweite von 1/16° (entspricht ungefähr 7 km), so hat dieser Raster im Raum Österreich $127 \times 45 = 5715$ Punkte. Die Berechnung aller paarweisen Entfernungen erfordert die Bestimmung von $5715^2 = 32661225$ Routen. Angenommen die Berechnung einer Route dauert 400 ms, so würde die Gesamtdauer 150 Tage betragen. Unter der Annahme dass die Wegzeiten (Distanzen) zwischen zwei Punkten symmetrisch sind – dies entspricht der Vernachläßigung von Einbahnstraßen – kann diese Zahl noch halbiert werden. Die Verwendung eines externen Programms wie Gosmore zur Berechnung der Routen führt dazu, dass sämtliche Daten des Wegenetzes für jede Routenberechnung erneut eingelesen werden müssen. Dies

Abbildung 5.7: Relative Reisezeiten nach Wien im Raum Österreich. Weiß entspricht der kürzesten Reisezeit und Schwarz der längsten in der betrachteten Region. Die Berechnung basiert auf einem Raster mit einer 1/16° (entspricht ungefähr 7 km) Unterteilung. Berechnungsdauer: 35 Minuten

Abbildung 5.8: Relative Reisezeiten nach Wien im Raum Österreich. Weiß entspricht der kürzesten Reisezeit und Schwarz der längsten in der betrachteten Region. Die Berechnung basiert auf einem Raster mit einer 1/64° (entspricht ungefähr 1.7 km) Unterteilung. Berechnungsdauer: 9 Stunden 15 Minuten

kann durch eine eigene Implementierung eines Routingalgorithmus verbessert werden, da diese Daten dann nur einmalig eingelesen werden müssen. Die Tatsache, dass alle paarweisen Distanzen zwischen den Rasterknoten berechnet werden sollen, könnte auch ausgenützt werden. Der Algorithmus von Dijkstra [L29, S. 273ff] kann so implementiert werden, dass er nicht nur den kürzesten Pfad zwischen zwei Punkten findet, sondern alle kürzesten Pfade zu einem bestimmten Punkt. Dies ist in der gleichen Laufzeit $O(n^2)$ möglich. Diese Verbesserungsmöglichkeiten wurden in dieser Arbeit nicht umgesetzt, könnten aber Gegenstand zukünftiger Untersuchungen sein. Zusammenfassend gesagt ist die Verwendung von Navigationssoftware zur Vorausberechnung von Wegstrecken nicht geeignet; eine sinnvolle Umsetzung der Vorausberechnung ist aber dennoch mit für diesen Zweck angepassten Algorithmen möglich.

Kapitel 6

Integration von GIS-Daten in ein Modell des österreichischen Gesundheitssystems

Dieses Kapitel befasst sich nun damit, wie GIS-Daten mit Hilfe der in Kapitel 4 vorgestellten Methoden in ein agentenbasiertes Modell [L9] des österreichischen Gesundheitssystems integriert werden können. Die Erstellung des Modells selbst war nicht Aufgabe dieser Arbeit, sondern es konnte aus einem Projekt von dwh Simulation Services in Zusammenarbeit mit dem Hauptverband der österreichischen Sozialversicherungsträger übernommen werden. Dieses Modell ist in AnyLogic implementiert.

6.1 Modellbeschreibung

Es gibt zwei Klassen von Agenten: Personen (Patienten) und medizinische Leistungserbringer (Ärzte, Labore, etc.). Ausgehend von realen statistischen Daten zur Inzidenz verschiedener medizinischer Probleme erkranken die Personen. In der verwendeten Implementierung des Modells wurde nur die Erkrankung an Diabetes berücksichtigt. Jeder Erkrankung ist eine Menge an benötigten Leistungen zugeordnet.

Tabelle 6.1: Umgebungsradien für Facharztgruppen

	Umgebungsradius
Praktischer Arzt	5 km
Institute[1]	∞
Alle anderen Fachärzte	25 km

[1] Rehabilitationszentren, Physiotherapiezentren, Labore

Tabelle 6.2: Patienten, die pro Wochentag behandelt werden können

	Mo	Di	Mi	Do	Fr	Sa	So
Praktischer Arzt	50	50	50	50	50	5	5
Medizinisch-Chemisches Labor	500	500	500	500	500	0	0
Alle anderen Fachärzte	50	50	50	50	50	0	0

Für das Verständnis der später präsentierten Ergebnisse ist es besonders wichtig zu wissen, wie die Wahl der Leistungserbringer erfolgt: Ausgehend von den benötigten Leistungen bestimmt der Patient jene Fachärzte, die alle seine benötigten Leistungen optimal abdecken und sich innerhalb einer bestimmten Umgebung befinden. Die Größe dieser Umgebung ist dabei abhängig vom Typ des Facharztes. Eine Auflistung der verwendeten Umgebungsradien findet sich in Tabelle 6.1. Bieten mehrere Fachärzte dieselben benötigten Leistungen an, so wird zufällig zwischen ihnen entschieden. Außerdem werden Fachärzte, bei denen die Wartezeit bis zum nächsten Termin mehr als 10 Tage beträgt, aus der Wahl ausgeschlossen. Bei den gewählten Fachärzten meldet sich der Patient dann zu einem Termin an. Die Wartezeit bei einem Facharzt resultiert aus der Anzahl der bereits angemeldeten Patienten und der Anzahl an Patienten, die ein Arzt pro Tag behandeln kann (Tabelle 6.2). Nimmt ein Patient schließlich einen Facharzttermin wahr, so bekommt er nur Leistungen die er auch benötigt und für die der Leistungserbringer berechtigt ist. Zum Schluss der Behandlung übergibt der Leistungserbringer noch die Informationen zur Behandlung an das Bezahlsystem. Dieses erhält somit Informationen zu sämtlichen erbrachten Leistungen und kann diese entsprechend abrechnen.

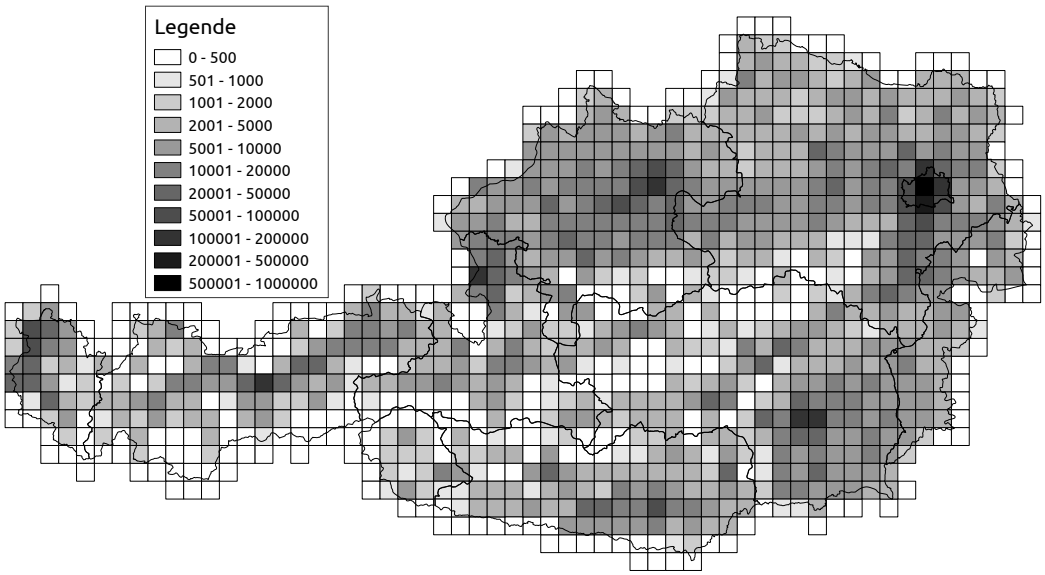

Abbildung 6.1: Bevölkerungsverteilung Österreichs in einem Raster mit 10×10 km großen Zellen. Die Legende ordnet den einzelnen Grauwerten Bereiche von Bevölkerungszahlen zu. Die Umrisse der Bundesländer sind zur besseren Orientierung auch dargestellt.

6.2 Verwendete Daten

Die folgenden Daten wurden in einzelne Shapefiles konvertiert und diese in das Modell integriert. Dadurch ist das Modell unabhängig von einer Datenbank ausführbar und kann daher leichter auch auf andere Rechner kopiert und dort simuliert werden. Sollte die Menge der Daten beträchtlich zunehmen, so dass sie sich nicht mehr als einzelen Shapefiles verwalten lassen, kann in Zukunft immer noch auf die Verwendung einer Datenbank umgestiegen werden.

Populationsdaten Die Bevölkerungsdaten werden in Bezug zum MGI-Lambert-Raster von der Statistik Austria veröffentlicht. Die frei verfügbaren Daten haben eine Auflösung von 10 km und geben den Bevölkerungsstand im Jahr 2011 an [W2]. Der MGI-Lambert-Raster wird in Abbildung 6.1 dargestellt.

Dauersiedlungsraum Dieser Raum fasst verschiedene Landnutzungsgebiete zusammen und wird auf der Webseite der Statistik Austria folgendermaßen beschrieben:

> „Der Dauersiedlungsraum umfasst den für Landwirtschaft, Siedlung und Verkehrsanlagen verfügbaren Raum. Die Abgrenzung des Dauersiedlungsraumes lässt einen relativ großen Spielraum zu, je nachdem welche Datengrundlagen herangezogen werden bzw. in welcher räumlichen Bezugsbasis diese zur Verfügung stehen. Datenquelle für die Dauersiedlungsraumabgrenzung sind die CORINE-Landnutzungsdaten 2000, sowie die Bevölkerung- und Beschäftigtendaten 2001 auf der Grundlage von 250 m-Rastereinheiten.
>
> Der Dauersiedlungsraum besteht aus einem Siedlungsraum mit den Nutzungskategorien städtisch geprägte Flächen, Industrie-, und Gewerbeflächen und aus einem besiedelbaren Raum mit den Nutzungskategorien Ackerflächen, Dauerkulturen, Grünland, heterogene landwirtschaftliche Flächen, Abbauflächen und den künstlich angelegten nicht landwirtschaftlich genutzten Flächen (z.B. städtische Grünflächen, Sport- und Freizeitflächen).
>
> Die auf der Grundlage von 250 m-Rastereinheiten abgegrenzte Dauersiedlungsraumfläche dient zur Berechnung von Dichtewerten (z.B.: Bevölkerungsdichte).“ [W1]

Der Dauersiedlungsraum ist in Abbildung 6.2 dargestellt.

Politische Gliederung Daten zu Bundesländern, Bezirken und Gemeinden wurden dem OpenStreetMap-Projekt entnommen.

6.3 Integration

Im Wesentlichen fand die Integration so wie in Kapitel 4 statt. Abb. 6.3 zeigt das Modell mit ca. 82364 Patienten, die als schwarze Punkte dargestellt

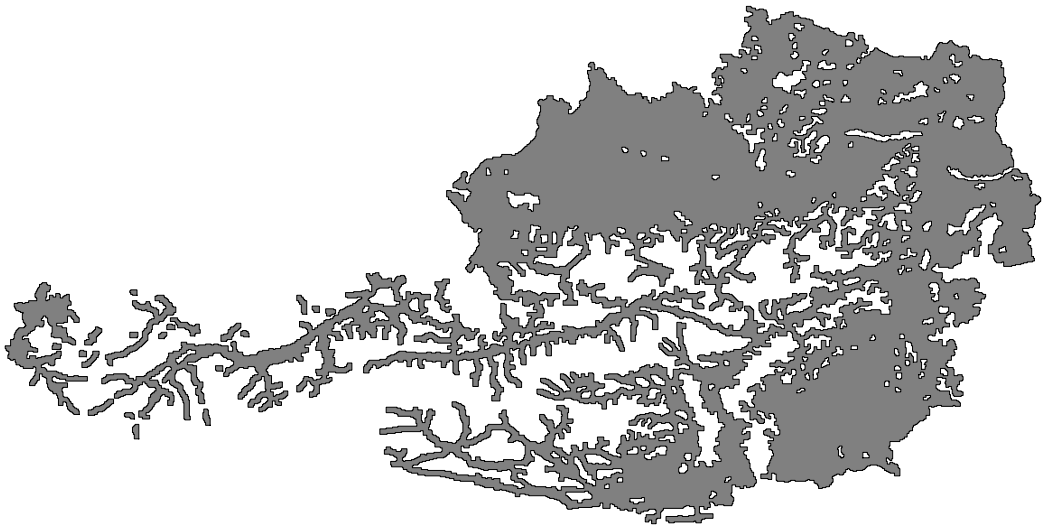

Abbildung 6.2: Dauersiedlungsraum Österreichs

werden. Deren Verteilung wurde aus den Populationsdaten abgeleitet, wobei der Lebensraum der Patienten durch den Dauersiedlungsraum eingeschränkt wird. Wie diese Initialisierung der Agenten im Detail funktioniert, wurde bereits in Abschnitt 5.1 beschrieben.

6.4 Ergebnisse

Das ursprünliche Modell verwendet als Umgebung für die Agenten einen kontinuierlichen quadratischen Raum mit einer Seitenlänge von 290 km. Diese Maße wurden deshalb gewählt, da dies ungefähr dem Flächeninhalt Österreichs entspricht. Die Verteilung der Patienten erfolgt zufällig gleichverteilt in diesem Quadrat. Mit den Ärzten wird in gleicher Weise verfahren.

In dem neuen Modell mit integrierten GIS-Daten wird das Land Österreich als Umgebung für die Agenten herangezogen. Die Verteilung der Patienten entspricht der Bevölkerungsverteilung. Mit diesem Modell wurden zwei Szenarien gerechnet: In einem wurden die Ärzte als in Österreich gleichverteilt angenommen, in dem anderen wurden diese ebenfalls entsprechend der Populationsdichte verteilt. Die Idee hinter dem ersten Szenario

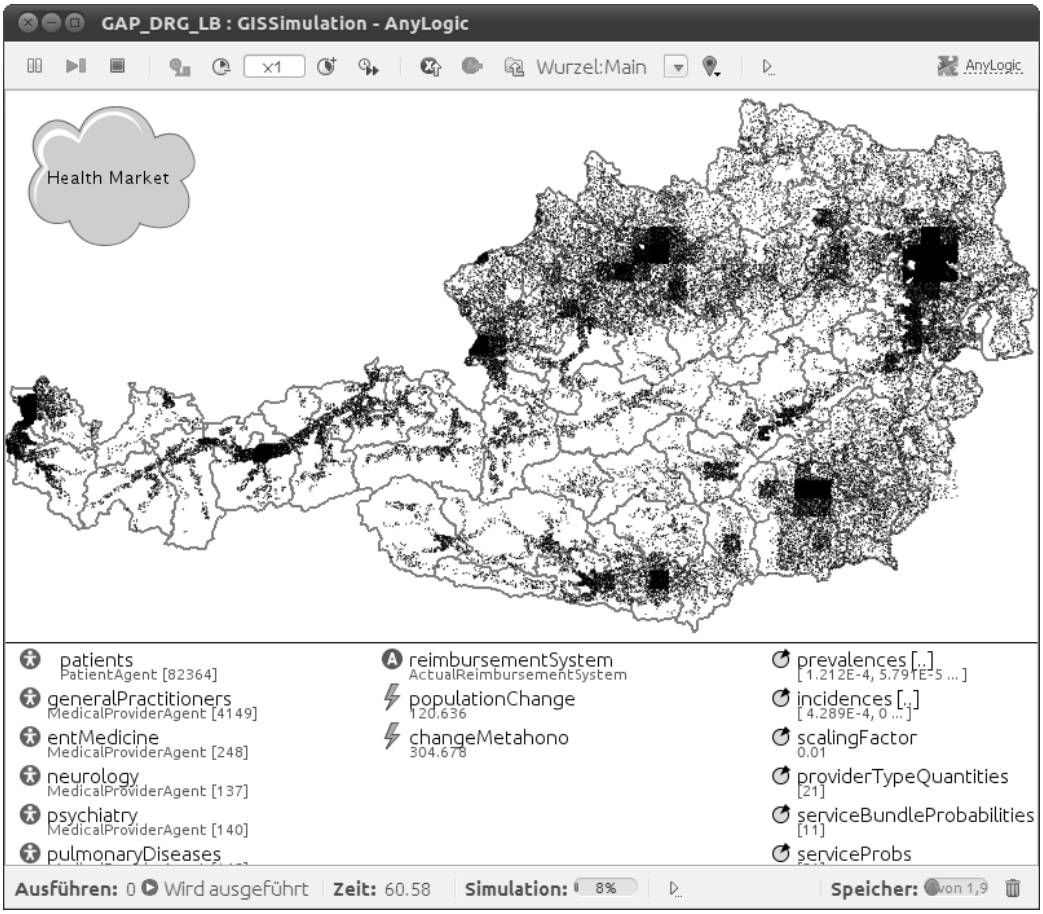

Abbildung 6.3: Graphische Darstellung des Modells. Im Hintergrund ist eine Karte der Bezirke Österreichs. Im Vordergrund sind die Patienten als schwarze Punkte abgebildet. In den Ballungsräumen ist das grobe Raster der frei verfügbaren Populationsdichte-Daten zu erkennen.

ist, dass die Versorgung mit Leistungserbringern in ganz Österreich gleichermaßen gegeben sein soll, während das zweite Szenario mehr den Umstand berücksichtigt, dass in Ballungsräumen mehr Ärzte benötigt werden als in ländlichen Gebieten.

Für die folgenden Ergebnisse (Abb. 6.4–6.12) wurden beide Modelle mit 82363 Patienten initialisiert. Das Modell benötigt auch noch die Anzahlen der Ärzte, wie zum Beispiel Dermatologen, Neurologen oder Radiologen. Die zur Initialisierung der verschiedenen Facharztgruppen benötigten Daten wurden aus einer für Österreich spezifischen Datenbank entnommen. In den Ergebnissen werden jedoch nur praktische Ärzte, innere Mediziner und Augenärzte betrachtet. Während der Simulation, die für einen Zeitraum von 2 Jahren durchgeführt wird, erkranken die Patienten an Diabetes – wie schon erwähnt wird im Modell momentan nur die Krankheit Diabetes berücksichtigt – und besuchen daraufhin verschiedenste Ärzte. Die folgenden Abbildungen stellen die Häufigkeitsverteilungen der Entfernungen zu den genannten Ärzten in beiden Modellen dar. Diese Arztgruppen wurden gewählt, da sich diese in der Anzahl der Ärzte beträchtlich unterscheiden, im Modell aber dennoch für diese Arztgruppen genügend Daten erzeugt werden, so dass repräsentative Auswertungen möglich sind. So gibt es zum Beispiel wesentlich mehr praktische Ärzte als innere Mediziner oder Augenärzte.

In den Auswertungen ist erkennbar, dass es immer eine bestimmte Entfernung gibt, ab der die Häufigkeit rapide abnimmt. Diese Entfernung entspricht genau dem für den Facharzt spezifischen Umgebungsradius, in dem dieser Arzt gesucht wird. Erst wenn in dieser Umgebung kein Arzt gefunden wird, werden auch weitere Entfernungen akzeptiert. Dadurch lässt sich diese Eigenschaft der Graphen erklären. Auffallend ist weiters, dass die Entfernungshäufigkeiten bis zu diesem Umgebungsradius annähernd linear zunehmen. Dies wird durch die Gleichverteilung der Ärzte und die Methode der Arztwahl bedingt. Wenn die Ärzte innerhalb des Umgebungsradius gleichverteilt sind, befinden sich am Rand der Kreisscheibe mehr Ärzte als im Zentrum. Dass zwischen der Häufigkeit und der Entfernung ein linearer Zusammenhang besteht, lässt sich auf den linearen Zusammenhang von Radius und Umfang zurückführen.

Die Ergebnisse für das Modell mit GIS-Daten, aber mit gleichverteilten Ärzten, unterscheiden sich kaum von jenen ohne GIS-Daten. Das bedeutet, dass die alleinige Verwendung von geographischen Daten für die Bevölkerung keinen oder nur einen sehr kleinen Einfluss auf die Häufigkeitsverteilung der Entfernungen zu den Ärzten hat. Anders gesagt ist die Entfernung zu den Ärzten unabhängig von der Bevölkerungsdichte. So gesehen ist das Ergebnis dieses Szenarios nicht überraschend, da die Ärzte gleichverteilt sind.

Im Szenario, in dem die Ärzte entsprechend der Populationsdichte verteilt sind, fällt auf, dass die Häufigkeiten der Entfernungen innerhalb des Umgebungsradius bei Augenärzten und inneren Medizinern wesentlich ausgeglichener sind als bei praktischen Ärzten oder in den anderen Szenarien. Die Ursache dafür liegt in der Arztverteilung. In Gebieten mit hoher Bevölkerungsdichte ist auch die Arztdichte entsprechend hoch und die Entfernungen sind somit kleiner. Dadurch verschieben sich die Häufigkeiten auf die in Abbildungen 6.9 und 6.12 dargestellte Art und Weise. Nun stellt sich die Frage, warum sich dieser Effekt bei praktischen Ärzten nicht einstellt, wie in Abbildung 6.6 ersichtlich ist. Der Grund dafür liegt in der Wahl des Umgebungsradius. Bei praktischen Ärzten liegt dieser bei 5 km und somit unterhalb der Auflösung des Populationsdichte-Rasters (10 km). Damit sind die Ärzte innerhalb des Umgebungsradius wieder gleichverteilt und die Situation entspricht jener in den anderen Szenarien.

Für die tatsächliche Inanspruchnahme des Gesundheitssystems gemessen an den auftretenden Kosten oder Behandlungsfällen kann die Integration von GIS-Daten in dieses Modell keine Auswirkung haben, da die Patienten nicht die Möglichkeit haben auf eine Behandlung auf Grund einer zu großen Entfernung zum Leistungserbringer zu verzichten oder alternative kostengünstigere Behandlungsmöglichkeiten zu erwägen.

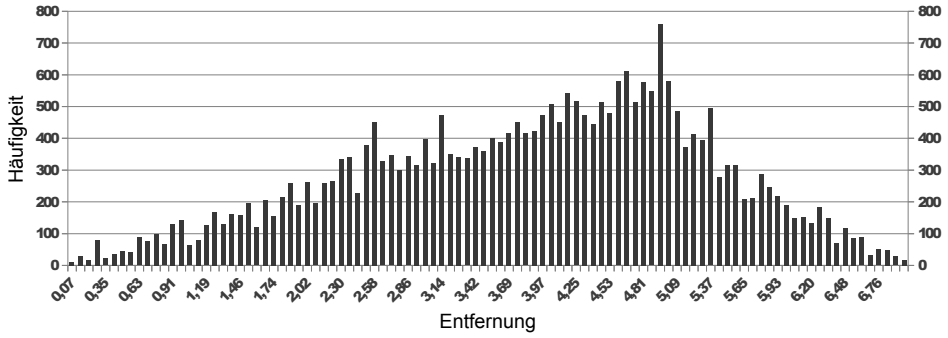

Abbildung 6.4: Häufigkeitsverteilung der Entfernung zu praktischen Ärzten im Modell ohne GIS-Daten

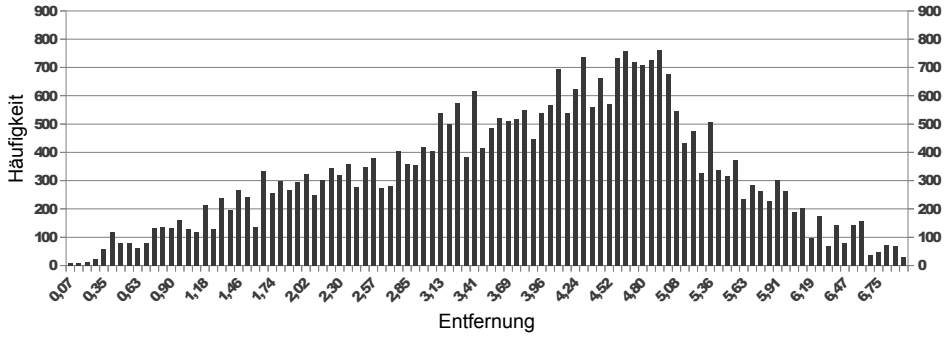

Abbildung 6.5: Häufigkeitsverteilung der Entfernung zu praktischen Ärzten im Modell mit GIS-Daten und gleichverteilten Ärzten

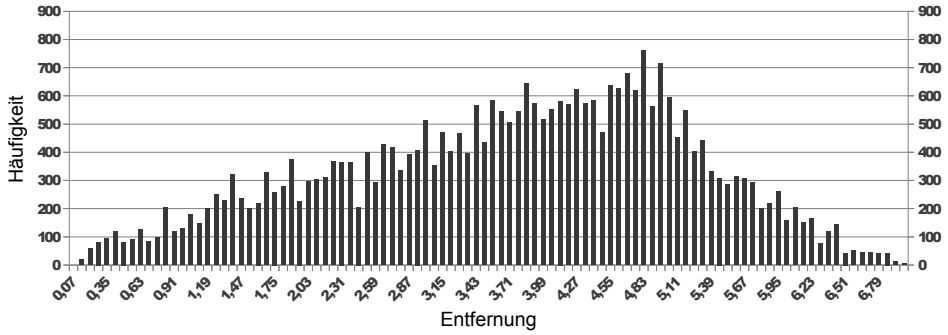

Abbildung 6.6: Häufigkeitsverteilung der Entfernung zu praktischen Ärzten im Modell mit GIS-Daten und Verteilung der Ärzte entsprechend der Bevölkerungsdichte

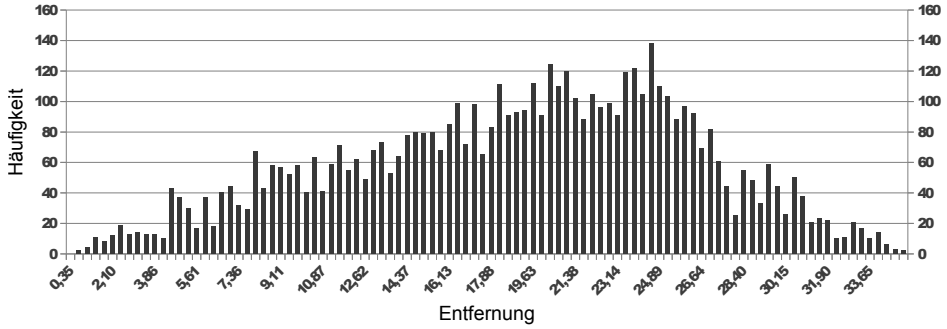

Abbildung 6.7: Häufigkeitsverteilung der Entfernung zu Augenärzten im Modell ohne GIS-Daten

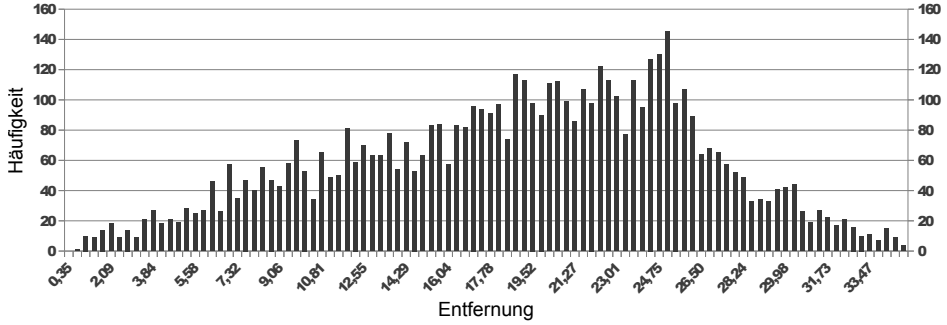

Abbildung 6.8: Häufigkeitsverteilung der Entfernung zu Augenärzten im Modell mit GIS-Daten und gleichverteilten Ärzten

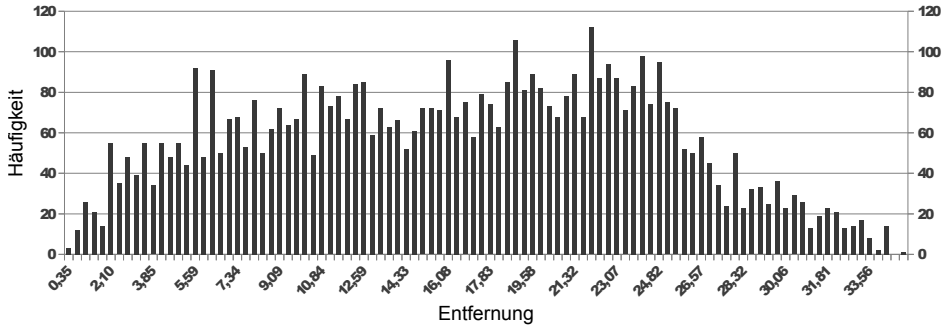

Abbildung 6.9: Häufigkeitsverteilung der Entfernung zu Augenärzten im Modell mit GIS-Daten und Verteilung der Ärzte entsprechend der Bevölkerungsdichte

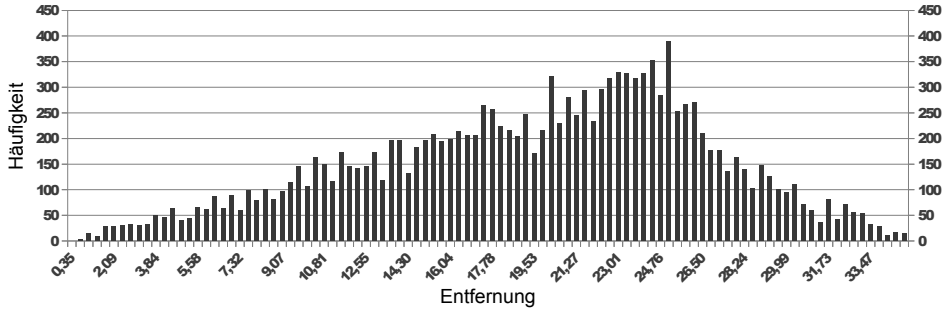

Abbildung 6.10: Häufigkeitsverteilung der Entfernung zu Fachärzten für Innere Medizin im Modell ohne GIS-Daten

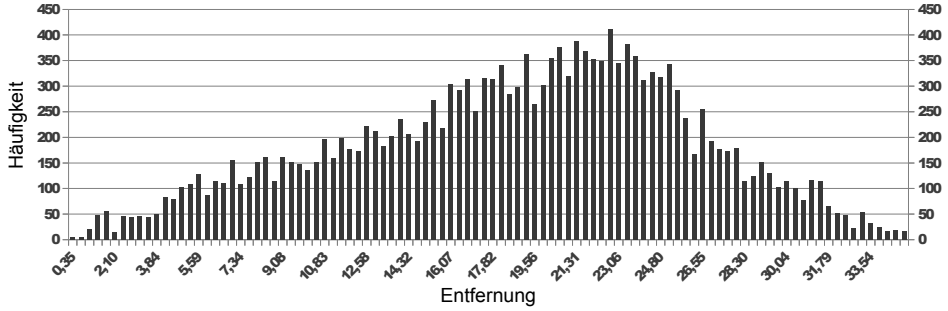

Abbildung 6.11: Häufigkeitsverteilung der Entfernung zu Fachärzten für Innere Medizin im Modell mit GIS-Daten und gleichverteilten Ärzten

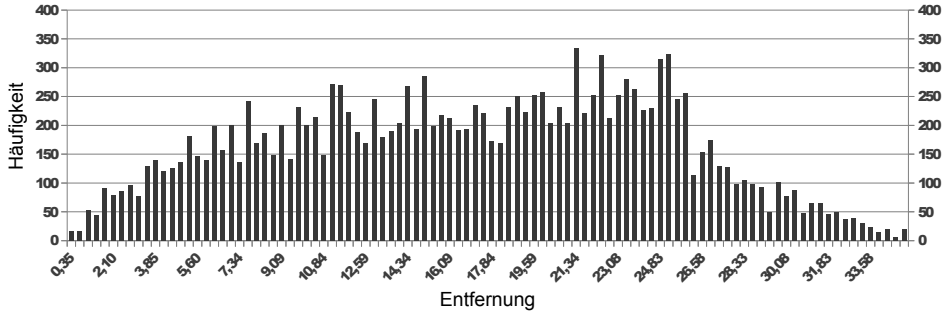

Abbildung 6.12: Häufigkeitsverteilung der Entfernung zu Fachärzten für Innere Medizin im Modell mit GIS-Daten und Verteilung der Ärzte entsprechend der Bevölkerungsdichte

Kapitel 7

Fazit und Ausblick

Das Ziel dieser Arbeit war es geographische Daten in ein bestehendes agentenbasiertes Modell des österreichischen Gesundheitssystems zu integrieren und eine Möglichkeit zur graphischen Darstellung dieser in Simulationsausgaben zu schaffen. Um diese Aufgabe zu erfüllen wurde zuerst untersucht, ob in der Literatur schon ähnliche Arbeiten vorhanden sind. Dies führte zu dem Schluß, dass es zwar einige Modelle zur Ausbreitung von Epidemien mit der Verwendung von GIS-Daten gibt, aber so gut wie keine umfassenden Modelle ganzer Gesundheitssysteme.

Obwohl die Modellierungsmethode von Anfang an feststand, da auf einem bereits bestehenden Modell aufgebaut werden sollte, wurde dennoch eine weitere Methode als Alternative zu agentenbasierter Modellierung untersucht: Diskrete Event Simulation. Es stellte sich heraus, dass der agentenbasierte Ansatz für die Modellierung von einzelnen Patienten und Leistungserbringern geeigneter ist.

Da Geographische Informationssysteme in dynamischen Modellen – zumindest im Gesundheitsbereich – nicht standardmäßig verwendet werden, wurde diesem Bereich ein eigenes Kapitel gewidmet, das eine Einführung in GI-Systeme gibt und näher auf Koordinaten-Bezugssysteme, gängige Datenformate und Datenquellen eingeht. Im Speziellen wurden auch für Österreich frei verfügbare Datenquellen untersucht.

Um Daten aus verschiedenen Quellen in ein Modell integrieren zu können,

mussten diese zusammengeführt und entsprechend homogenisiert werden. Dabei stellte sich heraus, dass es von Vorteil ist, wenn alle Daten in ein gemeinsames Koordinaten-Bezugssystem konvertiert werden. Für die Verwendung von Geodaten gibt es verschiedene Open Source Software-Bibliotheken, für die ebenfalls ein Überblick gegeben wurde. Da das vorhandene Modell in AnyLogic und Java implementiert worden war und auch viele andere Simulationswerkzeuge wie Repast oder MASON auf Java basieren, wurde der Schwerpunkt bei der Analyse der Integrationsmöglichkeiten auf diese Programmiersprache gelegt. Verweise und Vergleiche zu anderen Sprachen und Modellierungswerkzeugen wurden an entsprechenden Stellen eingebracht. Schließlich wurde für AnyLogic eine eigene Komponente $ShapeGeoToolsMap$ entwickelt, mit der es möglich ist geographische Daten aus verschiedensten Datenquellen in das Modell zu laden und als Umgebung für die Agenten zu verwenden. Die Darstellung des Simulationsoutputs findet damit auf einer interaktiven Karte der integrierten Geodaten statt. Während der Simulation können detailiertere Bereiche der Karte vergrößert werden und die Position der Agenten im Kontext zu verschiedenen geographischen Daten betrachtet werden. Sowohl bei der Erstellung des Leitfadens zur Integration von Geodaten in Java als auch bei der Implementierung der Schnittstelle in AnyLogic wurde besonders auf die Wiederverwendbarkeit – nicht nur für Modelle im Gesundheitsbereich – geachtet.

Während der Integration von GIS-Daten sind verschiedene Probleme aufgetreten, wie die Positionierung von Agenten entsprechend einer Populationsverteilung oder die Berechnung von Distanzen zwischen Agenten. Für die Lösung dieser Probleme wurden verschiedene Methoden betrachtet und in Hinblick auf ihre Effizienz untersucht. Dabei hat sich gezeigt, dass die Erhöhung der Genauigkeit durch die Verwendung von Wegdistanzen zu erheblichen Rechenaufwänden führen, die deren Verwendung momentan nicht ermöglichen. Lösungen für dieses Problem wurden angedacht, konnten in dieser Arbeit aber nicht weiter verfolgt werden.

Schließlich wurden Daten der Statistik Austria zur Bevölkerungsdichte und dem Dauersiedlungsraum sowie Daten aus OpenStreetMap in das bestehende Modell des Gesundheitssystems integriert und verschiedene Szenarien

mit dem Modell gerechnet. Diese Szenarien umfassten das Modell ohne GIS-Daten und jenes mit GIS-Daten einmal unter der Verwendung von gleichverteilten Leistungserbringern und einmal mit Verteilung der Leistungserbringer entsprechend der Bevölkerungsdichte. Ziel dieser Untersuchungen war es den Einfluss von geographischen Information auf die Entfernung zwischen Ärzten und Patienten festzustellen. Als Ergebnis konnte gezeigt werden, dass die Integration von Geodaten einen nicht zu vernachlässigenden Einfluss auf das Modell hat. Weiters haben die Ergebnisse gezeigt, dass eine zu grobe Auflösung des verwendeten Datenmaterials sich negativ auf das Modell auswirken kann und zwar in dem Sinne, dass bestimmte Effekte dadurch nicht wirksam werden.

Aufbauend auf dieser Arbeit könnten noch weitere Szenarien mit dem Modell simuliert werden, die andere Aspekte des Modells betrachten. Die Verteilung von Arztbesuchen und damit verbundenen Kosten auf administrative Regionen wie Bezirke oder Gemeinden ist ein Beispiel dafür. Ebenso könnten auch noch andere Daten in das Modell integriert werden wie zum Beispiel die reale Verteilung der Ärzte, soweit diese bekannt ist.

Anhang A

Quellcode von ShapeGeoToolsMap

```
package at.dwh.modelling.gis;

import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.GraphicsEnvironment;
import java.awt.Rectangle;
import java.awt.Toolkit;
import java.awt.Transparency;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseWheelEvent;
import java.awt.geom.AffineTransform;
import java.awt.geom.Point2D;
import java.awt.image.BufferedImage;

import org.geotools.geometry.DirectPosition2D;
import org.geotools.geometry.jts.ReferencedEnvelope;
import org.geotools.map.MapContent;
import org.geotools.map.Layer;
import org.geotools.map.MapViewport;
import org.geotools.map.event.MapBoundsEvent;
import org.geotools.map.event.MapBoundsListener;
import org.geotools.map.event.MapLayerEvent;
import org.geotools.map.event.MapLayerListEvent;
import org.geotools.map.event.MapLayerListListener;
import org.geotools.referencing.CRS;
import org.geotools.referencing.GeodeticCalculator;
import org.geotools.referencing.crs.DefaultGeographicCRS;
```

```
import org.geotools.referencing.datum.DefaultEllipsoid;
import org.geotools.renderer.GTRenderer;
import org.geotools.renderer.lite.StreamingRenderer;
import org.opengis.geometry.BoundingBox;
import org.opengis.geometry.Envelope;
import org.opengis.referencing.FactoryException;
import org.opengis.referencing.crs.CoordinateReferenceSystem;
import org.opengis.referencing.operation.MathTransform;
import org.opengis.referencing.operation.TransformException;

import com.xj.anylogic.engine.AbstractShapeGISMap;
import com.xj.anylogic.engine.presentation.Panel;
import com.xj.anylogic.engine.presentation.ReplicatedShape;
import com.xj.anylogic.engine.presentation.Shape;
import com.xj.anylogic.engine.presentation.ShapeGroup;
import com.xj.anylogic.engine.presentation.ShapeRectangle;
import com.xj.anylogic.engine.presentation.Texture;

import com.vividsolutions.jts.geom.Coordinate;

public class ShapeGeoToolsMap
    extends AbstractShapeGISMap
    implements MapLayerListListener, MapBoundsListener {

    static final long serialVersionUID = 1L;

    private static final int earthCircumference = 40075000;

    private ShapeRectangle mapFrame;

    private GTRenderer renderer;
    private BufferedImage baseImage;

    private MathTransform wgs84ToWorldTransform;
    private MathTransform worldToWGS84Transform;
    private CoordinateReferenceSystem displayCRS;
    private Point2D panDirection;
    private GeodeticCalculator calc = new GeodeticCalculator();

    private double screenResolution; // in Pixels per Meter

    private boolean lazyPanning = true;

    /*
     * If the user sets the display area before the pane is shown on screen we
     * store the requested envelope with this field and refer to it when the
     * pane is shown.
     */
    private ReferencedEnvelope pendingDisplayArea;
```

```
/*
 * This field is used to cache the full extent of the combined map layers.
 */
private ReferencedEnvelope fullExtent;
private MapContent mapContent;

/**
 * Creates a new map pane. Any or all arguments may be null
 *
 * @param panel The panel where the map will be drawn
 */
public ShapeGeoToolsMap(ShapeRectangle mapFrame) {
    this.mapFrame = mapFrame;
}

@Override
public double getX() {
    return mapFrame.getX();
}

@Override
public void setX(double x) {
    mapFrame.setX(x);
}

@Override
public double getY() {
    return mapFrame.getY();
}

@Override
public void setY(double y) {
    mapFrame.setY(y);
}

@Override
public void setPos(double x, double y) {
    mapFrame.setPos(x, y);
}

public double getWidth() {
    return mapFrame.getWidth();
}

public void setWidth(double width) {
    mapFrame.setWidth(width);
}

public double getHeight() {
    return mapFrame.getHeight();
```

```
}

public void setHeight(double height) {
    mapFrame.setHeight(height);
}

public void setSize(double width, double height) {
    mapFrame.setSize(width, height);
}

public Color getFillColor() {
    return mapFrame.getFillColor();
}

public void setFillColor(Color fillColor) {
    mapFrame.setFillColor(fillColor);
}

public void setFillColor(Object fillColor) {
    mapFrame.setFillColor(fillColor);
}

public Texture getFillTexture() {
    return mapFrame.getFillTexture();
}

public Color getLineColor() {
    return mapFrame.getLineColor();
}

public void setLineColor(Color lineColor) {
    mapFrame.setLineColor(lineColor);
}

public void setLineColor(Object lineColor) {
    mapFrame.setLineColor(lineColor);
}

public int getLineStyle() {
    return mapFrame.getLineStyle();
}

public void setLineStyle(int style) {
    mapFrame.setLineStyle(style);
}

public Texture getLineTexture() {
    return mapFrame.getLineTexture();
}
```

```
public double getLineWidth() {
    return mapFrame.getLineWidth();
}

public void setLineWidth(int width) {
    mapFrame.setLineWidth(width);
}

@Override
public double getScaleX() {
    return mapFrame.getScaleX();
}

@Override
public double getScaleY() {
    return mapFrame.getScaleY();
}

@Override
public boolean isVisible() {
    return mapFrame.isVisible();
}

@Override
public void setVisible(boolean v) {
    mapFrame.setVisible(v);
}

@Override
public double getRotation() {
    return mapFrame.getRotation();
}

@Override
public void setRotation(double r) {
    mapFrame.setRotation(r);
}

@Override
public void setScale(double s) {
    mapFrame.setScale(s);
}

@Override
public void setScale(double sx, double sy) {
    mapFrame.setScale(sx, sy);
}
```

```
@Override
public void setScaleX(double sx) {
    mapFrame.setScaleX(sx);
}

@Override
public void setScaleY(double sy) {
    mapFrame.setScaleY(sy);
}

@Override
public boolean canHandleClick(boolean arg0) {
    return true;
}

@Override
public boolean onClick(double clickx, double clicky) {
    try {
        Point2D p = new Point2D.Double(clickx, clicky);
        getScreenToFrameTransform().transform(p, p);
        getFrameToWorldTransform().transform(p, p);
        DirectPosition2D dp = new DirectPosition2D(
            getDisplayArea().getCoordinateReferenceSystem(), p.getX(), p.getY());
        getWorldToWGS84Transform().transform(dp, dp);
        mapFrame.onClick(dp.x, dp.y);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return true;
}

@Override
public boolean onClickAt(double px, double py, boolean publicOnly) {
    return onClick(px, py);
}

public boolean getLazyPanning() {
    return lazyPanning;
}

public void setLazyPanning(boolean lazyPanning) {
    this.lazyPanning = lazyPanning;
}
```

```
private void addToPresentation() {
    ShapeGroup group = mapFrame.getGroup();
    group.add(this);
    int i = group.indexOf(mapFrame) + 1;
    while (group.indexOf(this) != i) {
        Object shape = group.get(i);
        if (shape instanceof Shape) {
            group.remove((Shape) shape);
            group.add((Shape) shape);
        } else if (shape instanceof ReplicatedShape<?>) {
            group.remove((ReplicatedShape<?>) shape);
            group.add((ReplicatedShape<?>) shape);
        } else {
            group.remove((Integer) shape);
            group.add((Integer) shape);
        }
    }

    MouseAdapter mouseAdapter = new MouseAdapter() {
        private Point2D p;

        @Override
        public void mousePressed(MouseEvent ev) {
            if (contains(ev.getX(), ev.getY())) {
                p = getScreenToFrameTransform().transform(
                    new Point2D.Double(ev.getX(), ev.getY()), null);
            }
        }

        @Override
        public void mouseDragged(MouseEvent ev) {
            Point2D np = new Point2D.Double(ev.getX(), ev.getY());

            if (p != null) {
                getScreenToFrameTransform().transform(np, np);
                if (lazyPanning) {
                    panDirection = new Point2D.Double(
                        np.getX() - p.getX(),
                        np.getY() - p.getY());
                } else {
                    pan(
                        (p.getX() - np.getX())/getWidth()*4,
                        (p.getY() - np.getY())/getHeight()*4);
                    p = np;
                }
            }
        }
```

```
        @Override
        public void mouseClicked(MouseEvent ev) {
            if (ev.getClickCount() > 1) {
                reset();
            }
        }

        @Override
        public void mouseReleased(MouseEvent ev) {
            if (lazyPanning && p != null) {
                Point2D np = new Point2D.Double(ev.getX(), ev.getY());

                if (contains(np.getX(), np.getY())) {
                    getScreenToFrameTransform().transform(np, np);
                    panDirection = new Point2D.Double(
                        np.getX() - p.getX(),
                        np.getY() - p.getY());
                }

                if (panDirection != null) {
                    pan(
                        -panDirection.getX()/getWidth()*4,
                        panDirection.getY()/getHeight()*4);
                }
            }
            panDirection = null;
            p = null;
        }

        @Override
        public void mouseWheelMoved(MouseWheelEvent ev) {
            Point2D p = new Point2D.Double(ev.getX(), ev.getY());
            if (contains(p.getX(), p.getY())) {
                getScreenToFrameTransform().transform(p, p);
                double dx = p.getX()/getWidth()*4-2;
                double dy = 2-p.getY()/getHeight()*4;
                pan(dx, dy);
                zoom(-ev.getWheelRotation());
                pan(-dx, -dy);
            }
        }
    };

    getPanel().addMouseListener(mouseAdapter);
    getPanel().addMouseWheelListener(mouseAdapter);
    getPanel().addMouseMotionListener(mouseAdapter);

    Toolkit tk = getPanel().getToolkit();
    screenResolution = tk.getScreenResolution()/0.0254;
}
```

```
private Rectangle getVisibleRect() {
    return new Rectangle(0, 0, (int) getWidth(), (int) getHeight());
}

private Panel getPanel() {
    return mapFrame.getPresentable().getPresentation().getPanel();
}

private void invalidate() {
    baseImage = null;
}

public ReferencedEnvelope getDisplayArea() {
    if (mapContent != null) {
        return mapContent.getViewport().getBounds();
    } else if (pendingDisplayArea != null) {
        return new ReferencedEnvelope(pendingDisplayArea);
    } else {
        return new ReferencedEnvelope();
    }
}

public void reset() {
    if (fullExtent != null) {
        setDisplayArea(fullExtent);
    }
}

public void setCoordinateReferenceSystem(CoordinateReferenceSystem crs) {
    if (mapContent != null) {
        mapContent.getViewport().setCoordinateReferenceSystem(crs);
    }
}

public void setDisplayArea(Envelope envelope) {
    doSetDisplayArea(envelope);
    if (mapContent != null) {
        invalidate();
    }
}
```

```
/**
 * Helper method for setDisplayArea which is also called by
 * other methods that want to set the display area without provoking
 * repainting of the display
 *
 * @param envelope requested display area
 */
private void doSetDisplayArea(Envelope envelope) {
    if (mapContent != null) {
        CoordinateReferenceSystem crs = envelope.getCoordinateReferenceSystem();
        if (crs == null) {
            // assume that it is the current CRS
            crs = mapContent.getCoordinateReferenceSystem();
            if (crs == null) {
                crs = DefaultGeographicCRS.WGS84;
            }
        }

        ReferencedEnvelope refEnv = new ReferencedEnvelope(
                envelope.getMinimum(0), envelope.getMaximum(0),
                envelope.getMinimum(1), envelope.getMaximum(1),
                crs);

        mapContent.getViewport().setBounds(refEnv);

    } else {
        pendingDisplayArea = new ReferencedEnvelope(envelope);
    }
}

public BoundingBox getMaxBounds() {
    ReferencedEnvelope bounds = fullExtent;
    if (bounds.getCoordinateReferenceSystem() == null) {
        return bounds;
    } else {
        try {
            return fullExtent.toBounds(DefaultGeographicCRS.WGS84);
        } catch (Exception e) {
            e.printStackTrace();
            // return new ReferencedEnvelope(-180, 180, -90, 90,
            // DefaultGeographicCRS.WGS84);
            return bounds;
        }
    }
}
```

```
public AffineTransform getFrameToWorldTransform() {
    if (mapContent == null) {
        return null;
    } else {
        MapViewport viewport = new MapViewport(mapContent.getViewport());
        if (panDirection != null) {
            AffineTransform transform = viewport.getWorldToScreen();
            ReferencedEnvelope bounds = viewport.getBounds();
            bounds.translate(
                -panDirection.getX()/transform.getScaleX(),
                -panDirection.getY()/transform.getScaleY());
            viewport.setBounds(bounds);
        }
        return viewport.getScreenToWorld();
    }
}

public AffineTransform getWorldToFrameTransform() {
    if (mapContent == null) {
        return null;
    } else {
        MapViewport viewport = new MapViewport(mapContent.getViewport());
        if (panDirection != null) {
            AffineTransform transform = viewport.getWorldToScreen();
            ReferencedEnvelope bounds = viewport.getBounds();
            bounds.translate(
                -panDirection.getX()/transform.getScaleX(),
                -panDirection.getY()/transform.getScaleY());
            viewport.setBounds(bounds);
        }
        return viewport.getWorldToScreen();
    }
}

public AffineTransform getScreenToFrameTransform() {
    AffineTransform transform = new AffineTransform();
    transform.scale(1 / getScaleX(), 1 / getScaleY());
    transform.rotate(-getRotation());
    transform.translate(-getX(), -getY());
    return transform;
}

public AffineTransform getFrameToScreenTransform() {
    AffineTransform transform = new AffineTransform();
    // transform.translate(getX(), getY());
    transform.rotate(getRotation());
    transform.scale(getScaleX(), getScaleY());
    return transform;
}
```

```
private void checkForNewSize() {
    if (mapContent != null) {
        /*
         * Compare the new pane screen size to the viewport's screen area
         * and skip further action if the two rectangles are equal. This
         * check avoid extra rendering requests when redundant resize events
         * are received (e.g. on mouse button release after drag resizing).
         */
        if (mapContent.getViewport().getScreenArea().equals(getVisibleRect())) {
            return;
        }

        mapContent.getViewport().setScreenArea(getVisibleRect());

        if (pendingDisplayArea != null) {
            doSetDisplayArea(pendingDisplayArea);
            pendingDisplayArea = null;

        } else if (mapContent.getViewport().getBounds().isEmpty()) {
            setFullExtent();
            doSetDisplayArea(fullExtent);
        }
        invalidate();
    }
}

/**
 * Gets the renderer, creating a default one if required.
 *
 * @return the renderer
 */
public GTRenderer getRenderer() {
    if (renderer == null) {
        setRenderer(new StreamingRenderer());
    }
    return renderer;
}

/**
 * Sets the renderer to be used by this map pane.
 *
 * @param renderer
 * the renderer to use
 */
public void setRenderer(GTRenderer renderer) {
    if (renderer != null && mapContent != null) {
        renderer.setMapContent(mapContent);
    }
    this.renderer = renderer;
}
```

```
public void setMapContent(MapContent content) {
    if (mapContent != content) {
        if (mapContent != null) {
            mapContent.removeMapLayerListListener(this);
        }

        mapContent = content;

        if (mapContent != null) {
            MapViewport viewport = mapContent.getViewport();
            viewport.setMatchingAspectRatio(true);
            Rectangle rect = getVisibleRect();
            if (!rect.isEmpty()) {
                viewport.setScreenArea(rect);
            }

            mapContent.addMapLayerListListener(this);
            mapContent.addMapBoundsListener(this);

            if (!mapContent.layers().isEmpty()) {
                // set all layers as selected by default for the info tool
                for (Layer layer : mapContent.layers()) {
                    layer.setSelected(true);
                }

                setFullExtent();
                doSetDisplayArea(fullExtent);
            }
        }
        invalidate();
    }
    if (content != null && renderer != null) {
        renderer.setMapContent(content);
    }

    addToPresentation();
}

public MapContent getMapContent() {
    return mapContent;
}

private boolean setFullExtent() {
    return setFullExtent(mapContent.getCoordinateReferenceSystem());
}
```

```
/**
 * Determines the full extent
 * @return true if full extent was set successfully
 */
private boolean setFullExtent(CoordinateReferenceSystem crs) {
    if (mapContent != null && !mapContent.layers().isEmpty()) {
        try {
            fullExtent = mapContent.getMaxBounds();
            /*
             * Guard against degenerate envelopes (e.g. empty map layer or
             * single point feature)
             */
            if (fullExtent == null) {
                // set arbitrary bounds centred on 0,0
                fullExtent = new ReferencedEnvelope(-1, 1, -1, 1, crs);
            } else {
                double w = fullExtent.getWidth();
                double h = fullExtent.getHeight();
                double x = fullExtent.getMinimum(0);
                double y = fullExtent.getMinimum(1);

                double xmin = x;
                double xmax = x + w;
                if (w <= 0.0) {
                    xmin = x - 1.0;
                    xmax = x + 1.0;
                }

                double ymin = y;
                double ymax = y + h;
                if (h <= 0.0) {
                    ymin = y - 1.0;
                    ymax = y + 1.0;
                }

                if (crs == null) {
                    fullExtent = new ReferencedEnvelope(
                            xmin, xmax, ymin, ymax, DefaultGeographicCRS.WGS84);
                } else {
                    fullExtent = new ReferencedEnvelope(xmin, xmax, ymin, ymax, crs);
                }
            }
        } catch (Exception ex) {
            throw new IllegalStateException(ex);
        }
    } else {
        fullExtent = null;
    }
    return fullExtent != null;
}
```

```
/**
 * MapLayerListListener Implementation
 */

/**
 * Called when a new map layer has been added. Sets the layer as selected (for queries) and,
 * if the layer table is being used, adds the new layer to the table.
 */
@Override
public void layerAdded(MapLayerListEvent event) {
    setFullExtent();
    MapViewport viewport = mapContent.getViewport();
    if (viewport.getBounds().isEmpty()) {
        viewport.setBounds(fullExtent);
    }
    invalidate();
}

/**
 * Called when a map layer has been removed
 */
@Override
public void layerRemoved(MapLayerListEvent event) {
    if (mapContent.layers().isEmpty()) {
        fullExtent = null;
    } else {
        setFullExtent();
    }
    invalidate();
}

/**
 * Called when a map layer has changed, e.g. features added to a displayed feature collection
 */
@Override
public void layerChanged(MapLayerListEvent event) {
    if (event.getMapLayerEvent().getReason() == MapLayerEvent.DATA_CHANGED) {
        setFullExtent();
    }
    invalidate();
}

@Override
public void layerMoved(MapLayerListEvent event) {
    invalidate();
}

@Override
public void layerPreDispose(MapLayerListEvent event) {}
```

```
/**
 * MapBoundsListener Implementation
 */

/**
 * Called by the map content's viewport when its bounds have changed. Used
 * here to watch for a changed CRS, in which case the map is re-displayed at
 * full extent.
 */
@Override
public void mapBoundsChanged(MapBoundsEvent event) {
    int type = event.getType();
    if ((type & MapBoundsEvent.COORDINATE_SYSTEM_MASK) != 0) {
        /*
         * The coordinate reference system has changed. Set the map to
         * display the full extent of layer bounds to avoid the effect of a
         * shrinking map
         */
        setFullExtent();
        reset();
    }
}

/**
 * AbstractShapeGISMap Implementation
 */

@Override
public double getCenterLongitude() {
    try {
        return getDisplayArea().toBounds(DefaultGeographicCRS.WGS84).getMedian(0);
    } catch (Exception e) {
        return 0;
    }
}

@Override
public void setCenterLongitude(double centerLongitude) {
    setProjectionCenter(centerLongitude, getCenterLatitude());
}

@Override
public double getCenterLatitude() {
    try {
        return getDisplayArea().toBounds(DefaultGeographicCRS.WGS84).getMedian(1);
    } catch (Exception e) {
        return 0;
    }
}
```

```
@Override
public void setCenterLatitude(double centerLatitude) {
    setProjectionCenter(getCenterLongitude(), centerLatitude);
}

@Override
public void setProjectionCenter(double centerLongitude, double centerLatitude) {
    try {
        DirectPosition2D dp = new DirectPosition2D(centerLongitude, centerLatitude);
        getWGS84ToWorldTransform().transform(dp, dp);
        ReferencedEnvelope env = getDisplayArea();
        env.translate(dp.getX() − env.centre().x, dp.getY() − env.centre().y);
        setDisplayArea(env);
    } catch (TransformException e) {
        // do nothing
    } catch (FactoryException e) {
        // do nothing
    }
}

@Override
public float[] createTrajectory(double lonFrom, double latFrom, double lonTo, double latTo) {
    calc.setStartingGeographicPoint(lonFrom, latFrom);
    calc.setDestinationGeographicPoint(lonTo, latTo);
    double azimuth = calc.getAzimuth();
    double distance = calc.getOrthodromicDistance();
    float[] trajectory = new float[1026];
    trajectory[0] = (float) lonFrom;
    trajectory[1] = (float) latFrom;
    for (int i = 1; i < 512; i++) {
        calc.setDirection(azimuth, distance / 512 ∗ i);
        Point2D dest = calc.getDestinationGeographicPoint();
        trajectory[2 ∗ i] = (float) dest.getX();
        trajectory[2 ∗ i + 1] = (float) dest.getY();
    }
    trajectory[1024] = (float) lonTo;
    trajectory[1025] = (float) latTo;
    return trajectory;
}

@Override
public double convertXForward(double longitude, double latitude) {
    return convertXYForward(longitude, latitude).getX();
}

@Override
public double convertYForward(double longitude, double latitude) {
    return convertXYForward(longitude, latitude).getY();
}
```

```
private MathTransform getWGS84ToWorldTransform() throws FactoryException {
    CoordinateReferenceSystem crs = getDisplayArea().getCoordinateReferenceSystem();
    if (crs == null) {
        crs = DefaultGeographicCRS.WGS84;
    }
    if (displayCRS != crs || wgs84ToWorldTransform == null) {
        displayCRS = crs;
        wgs84ToWorldTransform = CRS.findMathTransform(
            DefaultGeographicCRS.WGS84, displayCRS, true);
    }
    return wgs84ToWorldTransform;
}

private MathTransform getWorldToWGS84Transform() throws FactoryException {
    CoordinateReferenceSystem crs = getDisplayArea().getCoordinateReferenceSystem();
    if (crs == null) {
        crs = DefaultGeographicCRS.WGS84;
    }
    if (displayCRS != crs || worldToWGS84Transform == null) {
        displayCRS = crs;
        worldToWGS84Transform = CRS.findMathTransform(
            displayCRS, DefaultGeographicCRS.WGS84, true);
    }
    return worldToWGS84Transform;
}

private Point2D convertXYForward(double longitude, double latitude) {
    try {
        DirectPosition2D dp = new DirectPosition2D(
            DefaultGeographicCRS.WGS84, longitude, latitude);
        if (displayCRS != null) {
            getWGS84ToWorldTransform().transform(dp, dp);
            dp.setCoordinateReferenceSystem(displayCRS);
        }

        Point2D dest = new Point2D.Double(dp.getOrdinate(0), dp.getOrdinate(1));
        getWorldToFrameTransform().transform(dest, dest);
        getFrameToScreenTransform().transform(dest, dest);
        return dest;
    } catch (Exception e) {
        e.printStackTrace();
        return new Point2D.Double();
    }
}

@Override
public double getDistance(double lonFrom, double latFrom, double lonTo, double latTo) {
    return DefaultEllipsoid.WGS84.orthodromicDistance(lonFrom, latFrom, lonTo, latTo);
}
```

```
@Override
public double convertRotationAngleForward(double longitude, double latitude, double angle) {
    double azimuth = (angle − Math.PI / 2) / Math.PI ∗ 180;
    if (azimuth > 180) {
        azimuth −= 360;
    } else if (azimuth < −180) {
        azimuth += 360;
    }
    calc.setStartingGeographicPoint(longitude, latitude);
    calc.setDirection(azimuth, 1000);
    Point2D dest = calc.getDestinationGeographicPoint();

    Point2D startXY = convertXYForward(longitude, latitude);
    Point2D destXY = convertXYForward(dest.getX(), dest.getY());

    return Math.atan2(
        destXY.getY() − startXY.getY(),
        startXY.getX() − destXY.getX())+2∗getRotation();
}

@Override
public double getMapScale() {
    ReferencedEnvelope displayArea = getDisplayArea();
    double w = displayArea.getMaxX()−displayArea.getMinX();
    return (getWidth()/screenResolution)/(w/(fullExtent.getWidth())∗earthCircumference);
}

@Override
public void setMapScale(double mapScale) {
    if (mapScale < getMinMapScale()) {
        mapScale = getMinMapScale();
    } else if (mapScale > getMaxMapScale()) {
        mapScale = getMaxMapScale();
    }

    ReferencedEnvelope displayArea = getDisplayArea();

    double w2 = (getWidth()/screenResolution)/
        (mapScale/(fullExtent.getWidth())∗earthCircumference)/2;
    double h2 = w2/displayArea.getWidth()∗displayArea.getHeight();

    Coordinate centre = displayArea.centre();
    displayArea.init(centre.x − w2, centre.x + w2, centre.y − h2, centre.y + h2);

    setDisplayArea(displayArea);
}
```

```
@Override
public double getMinMapScale() {
    if (fullExtent != null) {
        double w = fullExtent.getMaxX()−fullExtent.getMinX();
        return (getWidth()/screenResolution)/(w/(fullExtent.getWidth())∗earthCircumference);
    } else {
        return getWidth()/screenResolution/earthCircumference;
    }
}

@Override
public double getMaxMapScale() {
    return 1;
}

@Override
public void zoomOut() {
    zoom(−1);
}

@Override
public void zoomIn() {
    zoom(1);
}

public void zoom(int amount) {
    setMapScale(getMapScale()∗Math.pow(2, amount));
}

@Override
public void pan(int toEast, int toNorth) {
    pan((double)toEast, (double)toNorth);
}

public void pan(double toEast, double toNorth) {
    ReferencedEnvelope env = getDisplayArea();
    env.translate(toEast∗env.getWidth()/4, toNorth∗env.getHeight()/4);
    setDisplayArea(env);
}

@Override
public boolean contains(double px, double py) {
    return mapFrame.contains(px, py);
}
```

```
    @Override
    public boolean projectionContains(double longitude, double latitude) {
        try {
            DirectPosition2D dp = new DirectPosition2D(
                DefaultGeographicCRS.WGS84, longitude, latitude);
            getWGS84ToWorldTransform().transform(dp, dp);
            dp.setCoordinateReferenceSystem(displayCRS);

            return getDisplayArea().contains(dp);
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    @Override
    public void draw(Panel panel, Graphics2D g, AffineTransform xform, boolean publicOnly) {

        if (mapContent != null && !mapContent.getViewport().isEmpty()) {
            checkForNewSize();
            if (baseImage == null) {
                baseImage = GraphicsEnvironment.getLocalGraphicsEnvironment().
                        getDefaultScreenDevice().getDefaultConfiguration().
                        createCompatibleImage((int) getWidth(), (int) getHeight(),
                            Transparency.TRANSLUCENT);

                Graphics2D baseImageGraphics = baseImage.createGraphics();

                baseImageGraphics.setComposite(
                    java.awt.AlphaComposite.getInstance(java.awt.AlphaComposite.CLEAR, 0.0f));
                java.awt.geom.Rectangle2D.Double rect = new java.awt.geom.Rectangle2D.Double(
                    0, 0, (int) getWidth(), (int) getHeight());
                baseImageGraphics.fill(rect);

                getRenderer().paint(
                    baseImageGraphics,
                    new Rectangle(0, 0, (int) getWidth(), (int) getHeight()),
                    mapContent.getViewport().getBounds());

                baseImageGraphics.dispose();
            }
            if (panDirection == null) {
                g.drawImage(baseImage, 0, 0, null);
            } else {
                g.setClip(0, 0, (int) getWidth(), (int) getHeight());
                g.drawImage(baseImage, (int) panDirection.getX(), (int) panDirection.getY(), null);
            }
        }
    }
}
```

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[L1] Jerry Banks und John S. Carson. *Discrete-Event System Simulation*. Prentice-Hall International Series in Industrial and Systems Engineering. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1984. ISBN: 0-13-215582-6.

[L2] Norbert Bartelme. *Geoinformatik : Modelle, Strukturen, Funktionen*. 4. Aufl. Berlin; Heidelberg; New York: Springer, 2005. ISBN: 3-540-20254-4.

[L3] Norbert Bartelme. *GIS-Technologie : Geoinformationssysteme, Landinformationssysteme und ihre Grundlagen*. Berlin; Heidelberg; New York; London; Paris; Tokyo: Springer, 1989. ISBN: 3-540-50410-9.

[L4] Sally C. Brailsford. „Tutorial: Advances and Challenges in Healthcare Simulation Modeling“. In: *Proceedings of the 2007 Winter Simulation Conference*. Washington, D.C., USA: Omnipress, 2007, S. 1436–1448. ISBN: 1-4244-1306-0.

[L5] François E. Cellier. *Continuous system modeling*. Springer, 1991. ISBN: 3540975020.

[L6] Wai Kin Victor Chan, Young-Jun Son und Charles M. Macal. „Agent-based Simulation Tutorial - Simulation of Emergent Behaviour and Differences between Agent-based Simulation and Discrete-Event Simulation“. In: *Proceedings of the 2010 Winter Simulation Conference*. New York, N.Y.: Association for Computing Machinery, 2010, S. 135–150. ISBN: 9781424498642. URL: `http://www.informs-sim.org/wsc10papers/014.pdf` (besucht am 09. 02. 2012).

[L7] Eliseo Clementini, Jayant Sharma und Max J. Egenhofer. „Modelling topological spatial relations: Strategies for query processing“. In: *Computers & Graphics* 18 (6) (1994), S. 815–822.

[L8] W. Reynolds Craig. „Flocks, herds and schools: A distributed behavioral model“. In: *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. New York, N.Y.: Association for Computing Machinery, 1987, S. 25–34. ISBN: 0-89791-227-6. DOI: 10.1145/37401.37406.

[L9] Patrick Einzinger, Reinhard Jung und Nina Pfeffer. „Modeling Health Care Systems – An Approach Using Routine Health Care Data“. 2012.

[L10] ESRI. *ESRI Shapefile Technical Description*. 1998. URL: http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf (besucht am 09. 02. 2012).

[L11] Jay Wright Forrester. *Industrial dynamics*. Student edition. Cambridge, MA: Productivity Press, 1961. ISBN: 9780915299881.

[L12] Martin Gardner. „The fantastic combinations of John Conway's new solitaire game „Life“ “. In: *Scientific American* 223 (1970), S. 120–123.

[L13] Timothy C. Germann u. a. „Mitigation strategies for pandemic influenza in the United States“. In: *Proceedings of the National Academy of Sciences* 103 (Apr. 2006). URL: http://www.pnas.org/content/103/15/5935 (besucht am 09. 02. 2012).

[L14] Antonin Guttman. „R-Trees: A Dynamic Index Structure for Spatial Searching“. In: *Proceedings of the 1984 ACM SIGMOD international conference on Management of data - SIGMOD '84*. University of California, Berkeley: Association for Computing Machinery, 1984. ISBN: 0897911288. URL: http://www-db.deis.unibo.it/courses/SI-LS/papers/Gut84.pdf (besucht am 09. 02. 2012).

[L15] Eric Haines. „Point in Polygon Strategies“. In: *Graphics Gems IV*. Academic Press, 1994, S. 24–46.

[L16] Donald Hearn und M. Pauline Baker. *Computer Graphics with OpenGL*. New Jersey: Pearson Prentice Hall, 2004. ISBN: 0131202383.

[L17] International Association of Oil & Gas Producers (OGP). „Coordinate Conversions and Transformations including Formulas“. In: *Geomatics Guidance Note number 7, part 2*. OGP Publications, 2011. URL: `http://www.epsg.org/guides/docs/G7-2.pdf` (besucht am 09. 02. 2012).

[L18] Ryan C. Kennedy u. a. „A GIS Aware Agent-Based Model of Pathogen Transmission“. In: *International Journal of Intelligent Control and Systems* 14/1 (März 2009), S. 51–61. URL: `http://www.asmemesa.org/IJICS/files/154/5\%20\%20ijics.pdf` (besucht am 09. 02. 2012).

[L19] Joshua Michael Lloyd King. „Malaria in the Amazon: An Agent-Based Approach to Epidemiological Modeling of Coupled Systems“. Master Thesis. Waterloo, Ontario, Canada: University of Waterloo, 2009. URL: `http://hdl.handle.net/10012/4576` (besucht am 09. 02. 2012).

[L20] Patrick Kuckertz u. a. „Agent based modeling and simulation of a pastoral-nomadic land use system“. In: *ASIM 2011 - 21. Symposium Simulationstechnik Tagungsband*. Winterthur: Pabst Science Publishers, Sep. 2011. ISBN: 978-3-905745-44-3.

[L21] Sean Luke u. a. „MASON: A Multi-Agent Simulation Environment“. In: *Simulation: Transactions of the society for Modeling and Simulation International* 82(7) (2005), S. 517–527. URL: `http://cs.gmu.edu/~sean/papers/simulation.pdf` (besucht am 09. 02. 2012).

[L22] Charles M. Macal und Michael J. North. „Agent-based Modeling and Simulation“. In: *Proceedings of the 2009 Winter Simulation Conference*. New York, N.Y.: Association for Computing Machinery, 2009, S. 86–98. ISBN: 9781424457724. URL: `http://www.informs-sim.org/wsc09papers/009.pdf` (besucht am 09. 02. 2012).

[L23] Open Geospatial Consortium, Inc. (OGC). *OpenGIS® Catalogue Services Specification*. Feb. 2007. URL: http://portal.opengeospatial.org/files/?artifact_id=20555 (besucht am 09. 02. 2012).

[L24] Open Geospatial Consortium, Inc. (OGC). *OpenGIS® Coordinate Transformation Service Implementation Specification*. Jän. 2001. URL: http://portal.opengeospatial.org/files/?artifact_id=999 (besucht am 09. 02. 2012).

[L25] Open Geospatial Consortium, Inc. (OGC). *OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option*. Aug. 2010. URL: http://portal.opengeospatial.org/files/?artifact_id=25354 (besucht am 09. 02. 2012).

[L26] Open Geospatial Consortium, Inc. (OGC). *OpenGIS® Web Map Server Implementation Specification*. März 2006. URL: http://portal.opengeospatial.org/files/?artifact_id=14416 (besucht am 09. 02. 2012).

[L27] Open Geospatial Consortium, Inc. (OGC). *Styled Layer Descriptor profile of the Web Map Service Implementation Specification*. Juni 2007. URL: http://portal.opengeospatial.org/files/?artifact_id=22364 (besucht am 09. 02. 2012).

[L28] Open Geospatial Consortium, Inc. (OGC). *Symbology Encoding Implementation Specification*. Juli 2006. URL: http://portal.opengeospatial.org/files/?artifact_id=16700 (besucht am 09. 02. 2012).

[L29] Joseph O'Rourke. *Computational geometry in C*. Cambridge: Cambridge University Press, 1994. ISBN: 0521440343.

[L30] Liliana Perez und Suzana Dragicevic. „An agent-based approach for modeling dynamics of contagious disease spread". In: *International Journal of Health Geographics 2009* 8 (Aug. 2009).

[L31] Paul Ramsey. *The State of Open Source GIS*. Sep. 2007. URL: `http://www.refractions.net/expertise/whitepapers/opensourcesurvey/survey-open-source-2007-12.pdf` (besucht am 09. 02. 2012).

[L32] Refractions Research Inc. *PostGIS 1.5.3 Manual*. 2011. URL: `http://postgis.refractions.net/download/postgis-1.5.3.pdf` (besucht am 09. 02. 2012).

[L33] Philippe Rigaux, Michel O. Scholl und Agnes Voisard. *Spatial Databases With Application To GIS*. San Francisco: Morgan Kaufmann, 2002. ISBN: 1558605886.

[L34] Philip J. Schneider und David H. Eberly. *Geometric Tools for Computer Graphics*. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. San Francisco: Morgan Kaufmann Publishers, 2003. ISBN: 1-55860-594-0.

[L35] Keith Sullivan, Mark Coletti und Sean Luke. *GeoMason: GeoSpatial Support for MASON*. 2010. URL: `http://cs.gmu.edu/~tr-admin/papers/GMU-CS-TR-2010-16.pdf` (besucht am 09. 02. 2012).

[L36] Uri Wilensky. *NetLogo*. Center for Connected Learning und Computer-Based Modeling, Northwestern University, Evanston, 1999. URL: `http://ccl.northwestern.edu/netlogo/` (besucht am 09. 02. 2012).

[L37] Bernard P. Zeigler. „DEVS toda: recent advances in discrete event-based information technology“. In: *11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems 2003*, S. 148–161. ISBN: 0769520391.

Webverzeichnis

[W1] Statistik Austria. *Dauersiedlungsraum*. URL: http://www.statistik.at/web_de/klassifikationen/regionale_gliederungen/dauersiedlungsraum (besucht am 30. 01. 2012).

[W2] Statistik Austria. *Regionalstatistische Rastereinheiten MGI-Lambert–Raster*. URL: http://www.statistik.at/web_de/klassifikationen/regionale_gliederungen/raster_mgi_lambert (besucht am 30. 01. 2012).

[W3] Statistik Austria. *Statistik Austria*. URL: http://www.statistik.at (besucht am 30. 01. 2012).

[W4] OGP Geomatics Committee. *EPSG Geodetic Parameter Registry*. URL: http://www.epsg-registry.org (besucht am 30. 01. 2012).

[W5] OGP Geomatics Committee. *Geomatics Committee*. URL: http://www.epsg.org (besucht am 30. 01. 2012).

[W6] XJ Technologies Company. *Simulation Software Tool - Anylogic*. URL: http://www.xjtek.com (besucht am 30. 01. 2012).

[W7] Jean-Marie Dautelle. *JScience*. URL: http://jscience.org (besucht am 30. 01. 2012).

[W8] Martin Davis. *JTS Topology Suite*. URL: http://tsusiatsoftware.net/jts/main.html (besucht am 30. 01. 2012).

[W9] Inc. (Esri) Environmental Systems Research Institute. *Esri - The GIS Software Leader*. URL: http://www.esri.com (besucht am 30. 01. 2012).

[W10] Inc. Free Software Foundation. *GNU General Public License, version 2 (GPL-2.0)*. URL: http://www.opensource.org/licenses/gpl-2.0.php (besucht am 30. 01. 2012).

[W11] Geoland. *Geoland*. URL: http://www.geoland.at (besucht am 30. 01. 2012).

[W12] GeoTools. *GeoTools The Open Source Java GIS Toolkit*. URL: http://www.geotools.org (besucht am 30. 01. 2012).

[W13] Google. *Google Maps*. URL: http://maps.google.com (besucht am 30. 01. 2012).

[W14] Argonne National Laboratory. *Repast Suite*. URL: http://repast.sourceforge.net (besucht am 30. 01. 2012).

[W15] Microsoft. *Bing Maps*. URL: http://www.bing.com/maps (besucht am 30. 01. 2012).

[W16] International Association of Oil & Gas producers (OGP). *OGP - The International Association of Oil & Gas producers*. URL: http://www.ogp.org.uk (besucht am 30. 01. 2012).

[W17] OpenStreetMap. *Country and area extracts*. URL: http://wiki.openstreetmap.org/wiki/Planet.osm\#Country_and_area_extracts (besucht am 30. 01. 2012).

[W18] OpenStreetMap. *Gosmore*. URL: http://wiki.openstreetmap.org/wiki/Gosmore (besucht am 30. 01. 2012).

[W19] OpenStreetMap. *Nominatim*. URL: http://nominatim.openstreetmap.org (besucht am 30. 01. 2012).

[W20] OpenStreetMap. *OpenStreetMap*. URL: http://www.openstreetmap.org (besucht am 30. 01. 2012).

[W21] OpenStreetMap. *Planet.osm*. URL: http://wiki.openstreetmap.org/wiki/Planet.osm (besucht am 30. 01. 2012).

[W22] Open Source Geospatial Foundation (OSGeo). *Willkommen beim Quantum GIS Projekt*. URL: http://www.qgis.org (besucht am 30. 01. 2012).

[W23] Refractions Research. *PostGIS 1.5.3 Manual*. URL: `http://www.postgis.org/documentation/manual-1.5` (besucht am 30. 01. 2012).

[W24] Refractions Research. *Refractions Research*. URL: `http://refractions.net` (besucht am 30. 01. 2012).

[W25] BBN Technologies. *OpenMap®*. URL: `http://openmap.bbn.com` (besucht am 30. 01. 2012).

[W26] Frank Warmerdam. *PROJ.4*. URL: `http://trac.osgeo.org/proj` (besucht am 30. 01. 2012).

[W27] Stadt Wien. *Stadtplan Wien*. URL: `http://www.wien.gv.at/stadtplan` (besucht am 30. 01. 2012).

[W28] Stadt Wien. *Web Map Service (WMS)*. URL: `http://data.wien.gv.at/katalog/wms.html` (besucht am 30. 01. 2012).

[W29] Yahoo! *Yahoo! Maps, Driving Directions, and Traffic*. URL: `http://maps.yahoo.com` (besucht am 30. 01. 2012).