Technische Universität Graz
Dekanat für Informatik
Institut für Wissensmanagement

Entwicklung eines Frameworks zur Navigationssimulation

Masterarbeit

von

Michael EDER, BSc.

Vorgelegt zur Erlangung des
akademischen Grades eines Master
der Studienrichtung Informatik

Graz, im März 2013

Betreuer der Masterarbeit:

Assoc.Prof. Dipl.-Ing. Dr.techn. Denis HELIC

.....

Deutsche Fassung:

Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008

Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am 14. März 2013 ……..

(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

14th March 2013 ……..

(signature)

Danksagung

An dieser Stelle möchte ich mich bei Herrn Dipl.-Ing. Dr.techn. Denis Helic bedanken, der als Betreuer immer für Fragen zur Verfügung stand und diese Diplomarbeit möglich gemacht hat.

Außerdem bedanke ich mich bei Herrn Dipl.-Ing. Dr.techn. Markus Strohmaier, der mir durch die Einladung zu wöchentlichen Treffen am Institut ermöglichte, einen Einblick in die Themengebiete der wissenschaftlichen Arbeiten von anderen Studierenden zu gewinnen.

Ein weiterer Dank gilt meinen Studienkollegen Florian Geigl und Markus Horwath mit denen ich das Studium von Anfang an bestritten und gemeinsam viele Lehrveranstaltungen erfolgreich absolviert habe.

Abschließend bedanke ich mich bei meiner Familie, besonders bei meinen Eltern, die mir das Studium ermöglicht haben.

Bei allen Bezeichnungen, die auf Personen bezogen sind, meint die Formulierung beide Geschlechter, unabhängig von der in der Formulierung verwendeten konkreten geschlechtsspezifischen Bezeichnung.

Kurzfassung

Informationssysteme besitzen eine sehr weite Verbreitung und werden in vielen Sparten des täglichen Lebens eingesetzt. Speziell die immer dynamischeren Inhalte aus dem World Wide Web, die im Rahmen des Web 2.0 direkt von Nutzern erstellt werden, gewinnen immer mehr an Bedeutung. Doch die Möglichkeit, dass Benutzter Informationssysteme direkt mitgestalten können, bringt auch neue Herausforderungen mit sich. Da die Hauptaufgabe von Informationssystemen darin besteht Benutzer mit Informationen zu versorgen, ist ein wichtiger Aspekt dabei das Auffinden der gewünschten Information. Einer der integralen Bestandteile dieses Prozesses ist die Navigation durch das vorliegende System. Systeme sollten intuitiv navigierbar und Informationen für den Nutzer leicht erreichbar sein.

Aus diesem Grund wäre es von Vorteil ein Mittel zur Verfügung zu haben mit dem die Navigierbarkeit von Informationssystemen automatisch überprüft werden kann bzw. mit dem die Auswirkungen von Veränderungen an der Struktur von Informationssystemen automatisiert begutachtet werden können. Der erste Schritt, der getan werden muss um diesem Ziel näher zu kommen, umfasst die Simulation von menschlichem Navigationsverhalten in Informationssystemen.

Ein Ziel dieser Arbeit ist es ein Framework zu entwickeln, das die generelle Simulation von Navigationsvorgängen in Informationssystemen ermöglicht und durch Erweiterung und Verfeinerung später die Möglichkeit bietet menschliches Navigationsverhalten zu simulieren.

Diese Arbeit präsentiert die Architektur und Implementierung des Navigationsframeworks sowie einen Anwendungsfall bei dem das Framework bereits dazu eingesetzt wurde um Navigationssimulationen im Netzwerk von Wikipedia durchzuführen.

Abstract

Information systems are wide-spread and are used in many areas of daily life. Specifically, the World Wide Web's increasingly dynamic content that is created directly by the user in the so called Web 2.0 movement is becoming more important. But user created content also brings new challenges. Because the main task of information systems is to provide the user with information, one of the most important aspects is the possibility to find that desired information. One of the integral parts of that process is to be able to navigate through the system. Information should be easily accessible and systems should be intuitive to navigate.

For this reason it would be advantageous to have a means to automatically check the navigability of information systems and to determine the effects of changes to the structure of an information system. The first step towards this goal consists of being able to simulate the human navigation behavior in information systems.

One goal of this thesis is to develop a framework that enables the simulation of general navigation tasks in information systems and can be extended and refined to provide the ability to simulate human navigation behavior later on.

This thesis presents the navigation framework's architecture and implementation and a case study in which the framework was used to navigate through the Wikipedia network.

Inhaltsverzeichnis

1. Einführung

Dieses Kapitel soll einen Überblick darüber geben, mit welchen Themen sich diese Arbeit beschäftigt und worin die Motivation besteht. Außerdem werden die Ziele und die Struktur dieser Arbeit beschrieben.

1.1 Motivation

Seit den Anfangszeiten des World Wide Web [1] im Jahr 1989 hat sich das Medium sehr stark entwickelt und hat einen Beliebtheitsgrad erreicht, mit dem wahrscheinlich nicht einmal der Erfinder Tim Berners-Lee gerechnet hat.

Heute wird das WWW für viel mehr Dinge verwendet als nur um die Aufgabe zu erfüllen, für die es konzipiert wurde, nämlich den Austausch von wissenschaftlichen Arbeiten so bequem wie möglich zu machen.

Die Möglichkeit Dokumente mit Querverweisen, so genannten Hyperlinks, zu versehen macht die Ansammlung an beliebigen Inhalten erst zu diesem großen vernetzten Konstrukt, das wir mittlerweile alltäglich benutzen.

Bestand das Web zu Beginn nur aus statischen Texten, die von Personen mit Programmierkenntnissen erstellt und mit Hyperlinks versehen wurden, so wurden mit der Zeit auch multimediale Inhalte wie Bilder, Audio und Video Teil des WWW.

Zusammen mit den neuen Inhalten wurden auch Methoden verfügbar, die es auch dem durchschnittlichen Nutzer ermöglichten Inhalte im Web zu verfassen. Das Web 2.0 war geboren [2] und damit auch die kollaborative Arbeit an Webinhalten.

Anfangs war Navigation die einzige Möglichkeit die Inhalte aufzufinden, die das Informationsbedürfnis des Benutzers stillten. Mit dem Anstieg der Größe und des Umfangs des Webs wurden verschiedene Systeme zum Durchsuchen des Webs entwickelt.

Trotz der Möglichkeit verschiedene Suchfunktionen zu nutzen, ist die Navigation aber immer noch ein integraler Teil des Prozesses zum Auffinden von Infor-

mation, da in den meisten Fällen eine Suche nicht direkt zu den gewünschten Inhalten führt bzw. manchmal überhaupt nicht dafür geeignet ist.

Aus diesem Grund ist es interessant Informationssysteme nach Benutzerfreundlichkeit und Navigierbarkeit zu bewerten. Vor allem bei Inhalten, die kollaborativ und direkt von Benutzern erstellt werden, wäre eine automatische Bewertung der Struktur der Inhalte und der Navigierbarkeit von großem Nutzen, da nicht mehr alle Inhalte aus einer Hand stammen und nicht mehr zentral verwaltet werden.

In weiterer Folge könnten Webseiten und Systeme im Allgemeinen vollautomatisch so entworfen werden, dass sie benutzerfreundlich und intuitiv navigierbar sind.

Ein wichtiger Schritt in diese Richtung besteht darin, dass die Benutzernavigation in Informationssystemen besser verstanden und schlussendlich künstlich nachgebildet werden kann. Mit diesen Mitteln lässt sich dann überprüfen wie sich die Veränderung der Struktur eines Informationssystems auf die Navigierbarkeit auswirkt, ohne dass Testbenutzer nötig sind.

1.2 Themengebiet

Die Hauptthematik, mit der sich diese Arbeit beschäftigt, ist Benutzernavigation in Informationssystemen und deren Modellierung in Software.

Im Speziellen geht es darum ein Software-Grundgerüst zu erstellen, mit dem es möglich ist, künstliche Pfade zu erstellen, die dann mit vorhandenen Benutzernavigationspfaden verglichen werden können. Wichtig ist dabei, dass die Erweiterbarkeit gegeben ist, sodass das Grundgerüst weiterentwickelt werden kann, um immer bessere Ergebnisse liefern zu können.

Um dieses Thema aufarbeiten zu können, müssen auch verwandte Themen in Betracht gezogen werden. Erst das Verständnis der Grundlagen, die das Thema umgeben, ermöglicht eine genauere Bearbeitung des Hauptthemas.

Die in dieser Arbeit behandelten Inhalte umfassen von der Graphentheorie bis hin zur Betrachtung von Informationssystemen (im Speziellen das Fallbeispiel Wikipedia) ein recht breites Spektrum von Themen.

1.3 Aufgabenstellung

Ein Ziel dieser Arbeit ist es, ein Navigationsframework zu entwickeln, das es ermöglicht, Benutzernavigation in Informationssystemen zu modellieren. Im Speziellen geht es darum, Navigationspfade zu generieren. Diese Navigationspfade besitzen einen vorgegebenen Start- und Zielknoten und verlaufen über die Kanten eines vorgegebenen Graphen. Die Start- und Zielknoten der Pfade sind als Paare gegeben, was aber nicht bedeutet, dass jeder erzeugte Pfad auch den Zielknoten erreichen muss. Es sollen menschliche Navigationspfade nachmodelliert werden, und für die gilt ebenfalls, dass trotz bekanntem Ziel dieses von den Benutzern nicht immer erreicht wird.

Ein wichtiger Punkt ist, dass mit dieser Arbeit nicht versucht wird dem Menschen perfekt nachempfundene Pfade zu erzeugen. Stattdessen geht es um die Implementierung des Grundgerüsts, das in Zukunft in inkrementellen Schritten erweitert werden kann, sodass sich die erzeugten Pfade immer mehr an menschliche Navigationspfade annähern.

Aus diesem Grund werden Schnittstellen festgelegt, über die das Framework mit zusätzlicher Funktionalität erweitert werden kann.

Diese Schnittstellen werden, parallel zu dieser Arbeit, auch schon von Geigl und Horwath in ihren Arbeiten, [3] und [4], eingesetzt.

1.3.1 Übersicht über das Projekt

Diese Arbeit ist im Rahmen eines Projektes entstanden, zu dem neben dieser die zwei weiteren bereits erwähnten Arbeiten [3] und [4] gehören.

In Abbildung 1 ist eine Übersicht über die Gliederung des gesamten Projekts zu sehen.

In der Abbildung sind jene Aufgaben rot gekennzeichnet, die den Kern dieser Arbeit bilden und bearbeitet wurden, um die Ziele dieser Arbeit zu erreichen. Blau sind jene Bereiche gekennzeichnet, bei denen eine Zusammenarbeit mit den beiden anderen Diplomanden notwendig war, um Lösungen zu finden und sicherzustellen, dass die drei Arbeiten jeweils Ergebnisse lieferten, die kompatibel zueinander waren. Die schwarz gedruckten Bereiche fallen in Gebiete, die im Zuge einer der beiden anderen Arbeiten bearbeitet wurden.

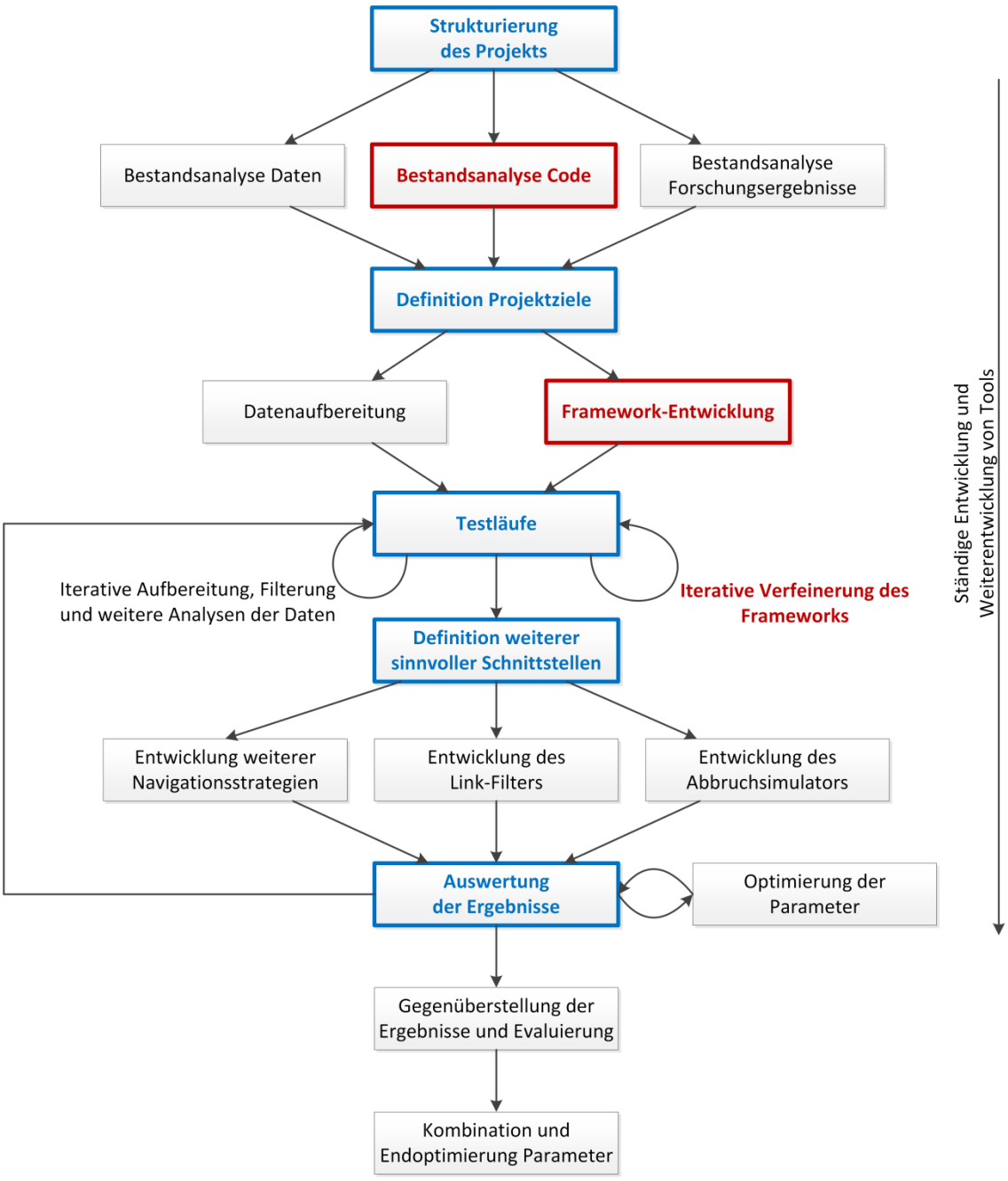

Abbildung 1: Projektübersicht

1.4 Aufbau der Arbeit

Nach der Beschreibung der Motivation sowie einer Übersicht über das Themengebiet, die Ziele und das Projekt im Allgemeinen in diesem Kapitel behandelt Kapitel 2 die Grundlagen der verschiedenen Themen.

Neben den Grundlagen zur Graphentheorie und Allgemeinem zur Navigation werden auch die Grundkonzepte der Softwarearchitektur und des objektorientieren Paradigmas erläutert. Außerdem wird ganz kurz die Stanford Network Analysis Platform, kurz SNAP, vorgestellt.

In Kapitel 3 werden einige Erkenntnisse und Forschungsergebnisse angeführt, die andere Wissenschaftler in Bereichen erzielt haben, die direkt von dieser Arbeit behandelt werden oder auf die diese Arbeit aufbaut.

Im darauffolgenden Kapitel 4 werden die Grundvoraussetzungen für das Navigationsframework abgeklärt. Neben den Anforderungen, die an das Framework gestellt wurden, wird besprochen, mit welchen Daten umgegangen werden muss und wie die Datenformate, mit denen das Framework arbeitet, aussehen.

Alles zum Design und zur Implementierung des Frameworks wird in Kapitel 5 behandelt. Es wird näher auf die zu Grunde liegende Bibliothek eingegangen und die einzelnen Teile des Frameworks werden erklärt. Außerdem werden die Schnittstellen besprochen, über die eine Erweiterung der Software möglich ist.

In Kapitel 6 wird darauf eingegangen, wie das Framework verwendet werden kann. Dazu wird ein konkretes Fallbeispiel im Detail besprochen.

In weiterer Folge werden eine Zusammenfassung und eine Beurteilung des Frameworks abgegeben.

Am Schluss folgt in Kapitel 8 ein Ausblick in die Zukunft, der beschreibt, in welche Richtungen eine Weiterentwicklung des Frameworks vielversprechend sein könnte und wo unabhängig vom Framework spannende Probleme auf dem Gebiet der Modellierung von Benutzernavigation liegen.

2. Grundlagen

Dieses Kapitel führt einige grundlegende Begriffe ein, die in dieser Arbeit verwendet werden und stellt Konzepte vor, auf die in den späteren Kapiteln aufgebaut wird.

2.1 Graphentheorie

Die Graphentheorie ist ein Teilgebiet der Mathematik, das die Eigenschaften von abstrakten Strukturen beschreibt und untersucht um Objekten und Verbindungen zwischen Objekten modellieren zu können. Das zentrale Konzept der Graphentheorie ist der *Graph*. Im Bezug auf Graphen müssen einige Unterscheidungen getroffen und Begriffe definiert werden.

2.1.1 Allgemeines

Ein Graph besteht aus einer Menge von *Knoten* und *Kanten*. Objekte werden als Knoten und die Verbindungen zwischen Objekten als Kanten abgebildet. Grafisch wird dies meist so aufbereitet, dass Knoten als Punkte oder Kreise und Kanten als Linien zwischen den Punkten bzw. Kreisen dargestellt werde. Auf Grund dieser einfachen Visualisierbarkeit von Graphen und der durch die Graphen modellierten Sachverhalte sind sie vielseitig einsetzbar. Neben Objekten werden durch Knoten oft Zustände und durch Kanten Zustandsübergänge modelliert. Durch diese Darstellung können Graphen zum Beispiel dazu genutzt werden um Automaten nachzubilden und übersichtlich zu visualisieren. [5]

Kanten werden grundsätzlich als Knoten-Paare beschrieben. Das Paar „A, B“ definiert die Kante zwischen den beiden Knoten mit den Bezeichnungen „A“ und „B“. Auf diese Weise wird eine symmetrische Beziehung zwischen den beiden Knoten abgebildet. Kanten sind aber nicht auf die Beschreibung dieser Art von Relation zwischen Knoten beschränkt.

Eine wichtige Unterscheidung ist zwischen *gerichteten* und *ungerichteten* Kanten und infolgedessen zwischen gerichteten Graphen, wie in Abbildung 2e und ungerichteten *Graphen*, wie in Abbildung 2d zu treffen. Gerichtete Graphen

bestehen, gleich wie ungerichtete, aus einer Menge von Knoten und Kanten, jedoch besitzt jede Kante eine zusätzliche Information, die die Richtung dieser Kante angibt. Betrachtet man einen Graphen mit lediglich zwei Knoten, „A“ und „B“, und einer Kante zwischen diesen beiden Knoten, so ist dieser Graph als ungerichteter Graph eindeutig (siehe Abbildung 2a). Für den Fall eines gerichteten Graphen kann man hier aber zwei verschiedene Graphen unterscheiden. [6] [7]

Weil ungerichtete Kanten eine symmetrische Beziehung zwischen zwei Knoten beschreiben, bedeutet dies, dass für ungerichtete Graphen die Paare „A, B“ und „B, A“ dieselbe Kante, nämlich die Kante zwischen den beiden Knoten mit der Bezeichnung „A“ und „B“, beschreiben. Wenn man Kanten in einem gerichteten Graphen angibt, so beinhaltet die Ordnung des Paares die zuvor erwähnte Richtungsinformation. Das Paar „A, B“ ist eine gerichtete Kante, die vom Knoten „A“ nach „B“ geht (Abbildung 2b). Die Kante in die entgegengesetzte Richtung wird durch das Knoten-Paar „B, A“ beschrieben und ist in Abbildung 2c ersichtlich. Kanten in gerichteten Graphen beschreiben also asymmetrische Beziehungen zwischen Knoten. Bei der Visualisierung von gerichteten Graphen werden Knoten weiterhin als Punkte oder Kreise dargestellt. Eine gerichtete Kante wird meist mit einem Pfeil dargestellt um die enthaltene Richtungsinformation zu verdeutlichen. [6]

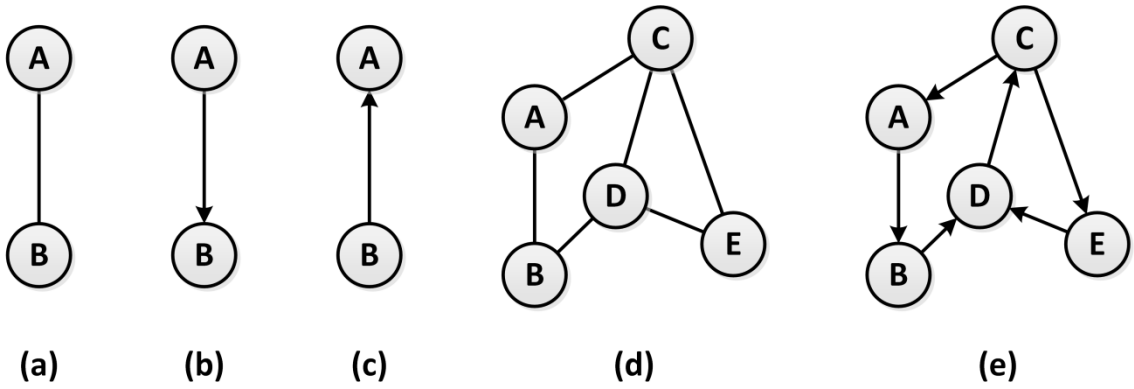

Abbildung 2: Beispiele für ungerichtete (a, d) und gerichtete Graphen (b, c, e)

Jeder ungerichtete Graph kann sehr einfach in einen gerichteten Graphen übergeführt werden indem für jede Kante zwischen Knoten im ungerichteten Graphen, im gerichteten Graphen zwei gerichtete Kanten in die jeweils entge-

gengesetzte Richtung zwischen denselben Knoten eingeführt werden. Gerichtete Graphen in ungerichtete Graphen überzuführen ist im Allgemeinen jedoch nicht ohne Verlust von Information möglich. [7]

In ungerichteten Graphen reicht es aus, wenn ein Knotenpaar maximal eine Kante besitzt um die Relationen zwischen diesem Knotenpaar abzubilden. Bei gerichteten Graphen kann ein Knotenpaar mit keiner, einer oder zwei Kanten verbunden sein, wobei bei einer Verbindung mit zwei Kanten die Richtung dieser Kanten entgegengesetzt ist. Graphen, die mehrere parallele Kanten, sogenannte Multikanten, zwischen einem Knotenpaar besitzen, nennt man *Multigraphen*. In den nachfolgenden Kapiteln sind grundsätzlich keine Multigraphen, wie zum Beispiel in Abbildung 3a dargestellt, gemeint, wenn der Begriff Graph benutzt wird. [7]

Eine weitere Unterscheidung kann zwischen *endlichen* und *unendlichen* Graphen getroffen werden. Als unendliche Graphen werden dabei Graphen mit nicht endlicher Anzahl an Knoten und Kanten bezeichnet. Unendliche Graphen sind für theoretische Probleme relevant, aber im Zuge dieser Arbeit wird grundsätzlich nur mit endlichen Graphen gearbeitet. Aus diesem Grund ist in den nachfolgenden Kapiteln dieser Arbeit unter dem Begriff Graph ein endlicher Graph zu verstehen. [5]

2.1.2 Begriffsdefinitionen

Eine wichtige Eigenschaft von Knoten ist der *Grad*. Damit wird angegeben, zu wie vielen anderen Knoten eine Kante existiert. Ein Knoten mit drei Kanten zu anderen Knoten hat den Grad drei und ist zum Beispiel der Knoten „A“ in Abbildung 3b. Bei gerichteten Graphen unterscheidet man zusätzlich zwischen *Ein-* und *Ausgangsgrad*. Es wird also zwischen Kanten, die bei dem entsprechenden Knoten beginnen bzw. enden, differenziert. Der Eingangsgrad eines Knoten k ist also die Anzahl der Kanten, die im Knoten k enden. Der Ausgangsgrad ist die Anzahl der Kanten die bei k beginnen. In Abbildung 3c hat der Knoten „D“ zum Beispiel den Eingangsgrad drei und den Ausgangsgrad eins. [6]

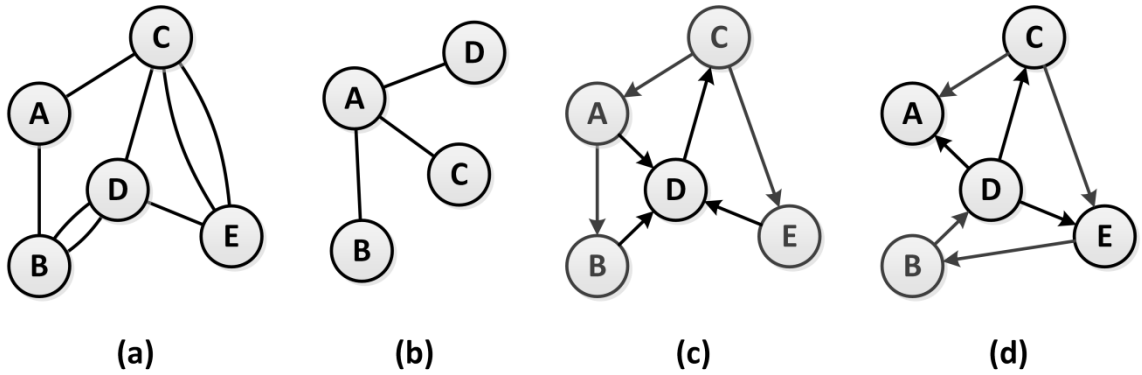

Abbildung 3: Multigraph (a), ungerichteter (b) und gerichtete Graphen (c, d)

Mit dem Begriff der *Nachbarschaft* eines Knoten k definiert man eine Menge von Knoten, die über eine Kante mit k verbunden sind. Im Falle einer ungerichteten Kante zwischen einem Knoten k und Knoten j ist sowohl k ein Nachbar von j als auch j ein Nachbar von k. Eine gerichtete Kante mit dem Startknoten k und dem Endknoten j macht j zu einem Nachbarn von k aber nicht umgekehrt. In Abbildung 3d sind die Knoten „A“, „C“ und „E“, nicht jedoch „B“ ein Nachbar von Knoten „D“. [6]

Aufbauend auf die Nachbarschaft kann man den Begriff des Vorgängers und Nachfolgers definieren. Ein Vorgänger des Knoten j ist der Knoten i, wenn eine Kante von i nach j existiert. Ein Nachfolger des Knoten j ist ein Knoten k, wenn eine Kante von j nach k existiert. „C“ ist in Abbildung 3d also ein Nachfolger von Knoten „D“. „C“ wiederum ist der Vorgänger von „A“ und „E“.

Neben der Nachbarschaft, die über eine einzelne verbindende Kante definiert ist, gibt es in Graphen auch *Pfade*. Ein Pfad ist eine geordnete Liste von Knoten, in der aufeinanderfolgende Knoten über Kanten verbunden sind. Den ersten und letzten Knoten kann man als Start- und End- bzw. Zielknoten ansehen. Auf diese Weise kann man eine Bewegung über die Kanten eines Graphen beschreiben. Mit diesem Konzept kann zum Beispiel die Reiseroute eines Geschäftsreisenden modelliert werden. Hierbei könnten alle internationalen Flugverbindungsstrecken und die Flughäfen die Kanten bzw. Knoten eines Graphen darstellen und die Reiseroute einen Pfad in diesem Graph. Ein solcher Pfad wäre dann zum Beispiel als „London, Wien, Paris, New York“ definiert,

wobei „London“ den Startknoten und „New York“ den Endknoten dieses Pfades darstellt. [6]

Zum Konzept der Pfade in Graphen können genauere Unterscheidungen getroffen werden. Je nachdem, ob Knoten mehrfach besucht werden dürfen oder nicht, wird in der Literatur öfters zwischen Wegen und Pfaden unterschieden. In dieser Arbeit ist diese Unterscheidung aber nicht notwendig und „Pfad“ wird als freier Oberbegriff verwendet und beschreibt grundsätzlich Listen aus Knoten die über Kanten verbunden sind.

Abbildung 4a zeigt zwei verschieden Pfade in grün und rot, die beide denselben Startknoten „A“ und Endknoten „E“ besitzen

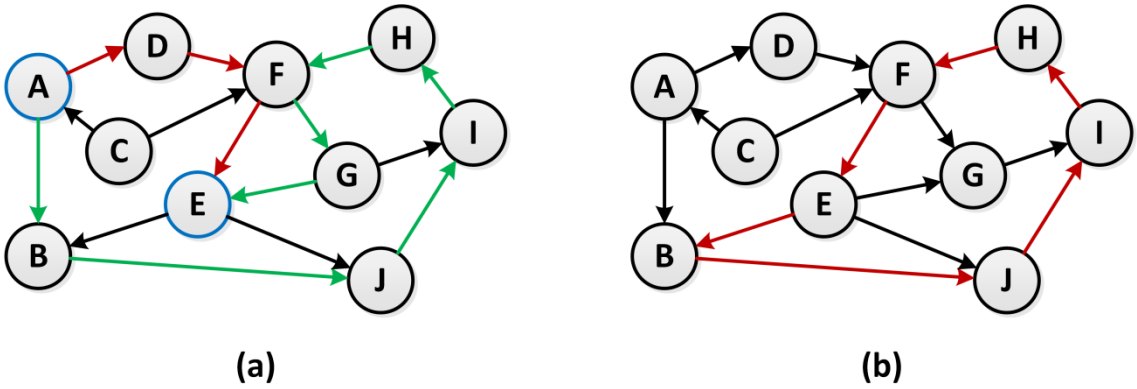

Abbildung 4: Beispiel für Pfade und Kreise in Graphen

Pfade, die so verlaufen, dass sie wieder zum Beginn zurückkehren, also mit identischem Start- und Endknoten, nennt man *Kreise* wenn alle besuchten Knoten außer dem Start- und Endknoten nur ein Mal besucht werden. Werden Knoten mehrfach besucht und der Start- und Endknoten ist derselbe Knoten, so spricht man von einem *Zyklus*. In Abbildung 4b ist in rot ein Kreis über die Knoten „B“, „J“, „I“, „H“, „F“ und „E“ eingezeichnet. [7]

Über die Definition von Pfaden kann man auch *Distanzen* in Graphen definieren. Die *kürzeste Distanz* von zwei Knoten k und j entspricht der Länge des kürzesten Pfades mit dem Startknoten k und dem Endknoten j. Die Länge eines Pfades ist durch die Anzahl der Kanten, die dieser enthält, gegeben. Die zuvor erwähnte Nachbarschaft von Knoten kann also auch als die Menge der Knoten,

zu denen es Pfade der Länge eins gibt bzw. als die Menge der Knoten mit der Distanz eins zum Knoten, definiert werden.

Zur Berechnung der kürzesten Distanz zwischen einem Startknoten und einem Zielknoten oder zu allen anderen Knoten in Graphen gibt es verschieden Algorithmen, wobei die Breitensuche die einfachste Variante darstellt.

Teilgraphen oder *Subgraphen* beschreiben eine Teilmenge der Knoten und Kanten eines Graphen. Die Struktur des Graphen bleibt bei einem Teilgraphen soweit wie möglich unverändert. Die Kanten zwischen den Knoten, die im Teilgraphen enthalten sind, bleiben intakt. Lediglich Kanten zwischen Knoten, die nicht im Teilgraphen enthalten sind sowie Kanten zwischen Knoten im Teilgraphen und außerhalb des Teilgraphen, werden entfernt. Ein Beispiel für einen Teilgraphen, der entsteht, wenn man aus dem ursprünglichen Graphen die Knoten „A“, „B“, „C“ und „D“ entfernt, ist in Abbildung 5a zu sehen. [5]

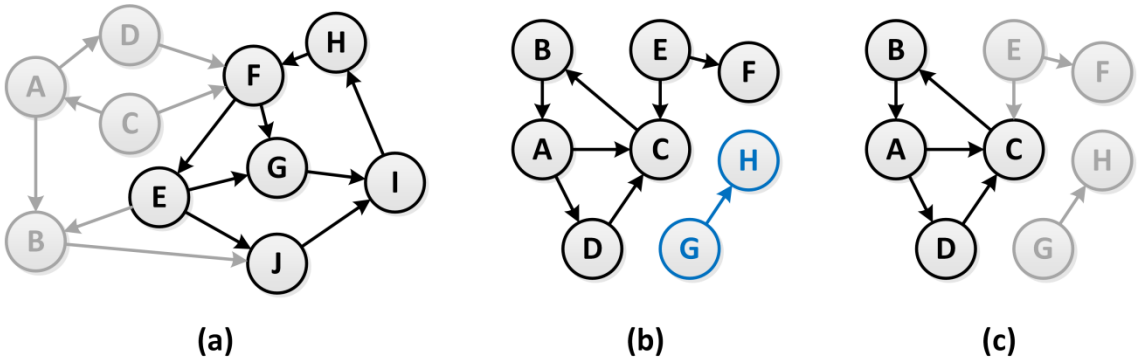

Abbildung 5: Teilgraphen und Zusammenhang von Graphen

Ein Graph ist *zusammenhängend*, wenn im Graph für beliebige Knoten j und k ein Pfad zwischen j und k existiert. Bei gerichteten Graphen kann an dieser Stelle noch zwischen starkem und schwachem Zusammenhang unterschieden werden. Existiert für jedes beliebe Knotenpaar (j, k) im gerichteten Graphen ein Pfad von j nach k über die gerichteten Kanten, so bezeichnet man diesen gerichteten Graphen als *stark zusammenhängend*. Existieren für alle beliebigen Knotenpaare (j, k) im ungerichteten Graph, der durch Ersetzen aller gerichteten Kanten mit ungerichteten Kanten entsteht, Pfade, so ist der gerichtete Graph *schwach zusammenhängend*. Der in Abbildung 5b abgebildete Graph besteht

aus zwei schwach zusammenhängenden Teilgraphen. Der dazugehörige stark zusammenhängende Teilgraph ist in Abbildung 5c hervorgehoben. [5] [6]

Eine spezielle Form von Graphen bezeichnet man als *Baum*. Bäume können ebenfalls gerichtet oder ungerichtet sein. Ungerichtete Bäume sind ungerichtete Graphen, die keine Kreise enthalten und zusammenhängend sind. In Abbildung 6a ist ein Beispiel für einen solchen ungerichteten Baum zu sehen. Knoten mit dem Grad eins (in der Abbildung in blau) werden dabei als *Blätter* bezeichnet. Höhergradige Knoten werden meist innere Knoten genannt und sind in der Abbildung grün dargestellt.

Gerichtete Bäume sind jene gerichtete Graphen, die kreisfrei sind und genau einen Knoten mit Eingangsgrad null besitzen. Der Knoten mit keinen eingehenden Kanten wird als *Wurzel* bezeichnet und ist in Abbildung 6b rot markiert. Knoten mit einem Ausgangsgrad von null sind wiederum die Blätter des Baumes. Sind der Eingangs- und Ausgangsgrad eines Knoten größer null, so ist dieser Knoten ein innerer Knoten. [5]

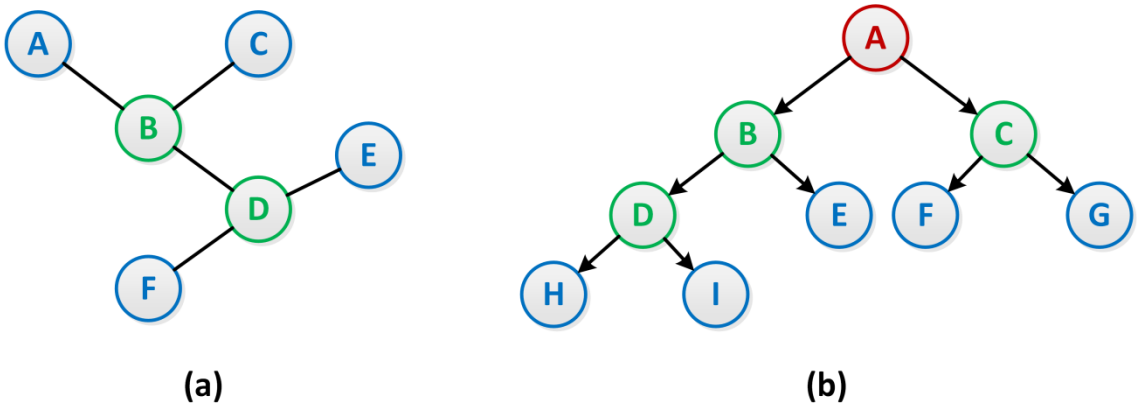

Abbildung 6: Beispiele für ungerichtete (a) und gerichtete Bäume (b)

2.2 Navigation

Mit dem Begriff des Pfades wurde bereits definiert, wie man Bewegung über die Kanten eines Graphen beschreiben kann. Startend bei einem Knoten in einem gerichteten oder ungerichteten Graphen werden vorhandene Kanten zu weiteren Knoten verfolgt und so entsteht eine Liste aus Kanten bzw. ein Pfad über

Knoten, die durch Kanten verbunden sind. Diese „Reise“ von Knoten zu Knoten über die vorhandenen Kanten kann man als Navigation in Graphen betrachten.

Navigation in Graphen ist für theoretische Betrachtungen, wie zum Beispiel dem Problem des Handlungsreisenden, sehr interessant. Bei diesem Problem geht es darum, alle Knoten eines Graphen möglichst effizient zu besuchen und wieder zurück zum Startknoten zu gelangen. [8]

Das Problem des Handlungsreisenden bildet dabei ein Verkehrsnetzwerk, das Städte miteinander verbindet, als Graphen ab. Die Städte sind Knoten im Graphen und Verkehrsverbindungen sind Kanten. Die durch die Graphen abstrahierten zugrundeliegenden Systeme sind aber natürlich nicht immer Verkehrsnetzwerke. Sehr häufig werden Informationssysteme als Graphen modelliert um Navigation in diesen untersuchen zu können.

2.2.1 Informationssysteme

Ein Informationssystem ist ein System, das die Aufgabe hat, Informationen zur Verfügung zu stellen um Informationsbedürfnisse zu decken.

Die Navigierbarkeit von Informationssystemen ist ein besonders interessantes Thema, da diese ein ausschlaggebender Faktor dafür ist, wie gut ein Informationssystem seine Aufgabe erfüllt. Egal ob Menschen auf der Suche nach einem ganz bestimmten Stück Information sind oder sich nur mit einer groben Vorstellung eines Zieles durch ein Informationssystem bewegen. Jede Art der Navigation wird von bestimmten Eigenschaften des Systems beeinflusst.

Aus diesem Grund ist es geradezu ideal, dass sich Informationssysteme sehr gut mittels Graphen modellieren lassen. Basierend auf diesen Modellen lassen sich die Systemeigenschaften und deren Einfluss auf die Navigierbarkeit sehr gut untersuchen.

Des Weiteren kann die Navigation von Menschen durch ein Informationssystem sehr gut durch Pfade in Graphen repräsentiert werden.

Mit all diesen Mitteln ist es also möglich menschliches Navigationsverhalten so zu formulieren, dass es analysiert werden kann. Außerdem kann durch diese Art der Repräsentation Navigation durch Informationssysteme auch simuliert werden, da ein Navigationspfad durch das Informationssystem ganz einfach einem Pfad durch einen Graphen entspricht. Auf Grund der abstrakten Darstellungsweise als Graph kann die Navigation im Informationssystem recht unabhängig vom ursprünglichen System untersucht werden.

2.2.2 Hintergrundwissen

Wenn Menschen durch ein Informationssystem navigieren, so geschieht dies nicht wahllos. Grundsätzlich steht hinter der Navigation durch ein Informationssystem eine Motivation. Im Allgemeinen ist dies oft der Wunsch eine bestimmte Information ausfindig zu machen oder zu einem Themengebiet zu navigieren um grundsätzlich mit dem gewünschten Thema verwandte Informationen aufzufinden.

Neben der Motivation ist bei der Navigation auch eine Art von Hintergrundwissen vorhanden, die es erst ermöglicht, den Navigationsvorgang so zu steuern, dass kein rein willkürlicher Navigationspfad durch das Informationssystem entsteht.

Natürlich ist das menschliche Hintergrundwissen sehr komplex und, wenn überhaupt, nur sehr schwer modellierbar. Stark verallgemeinert kann man Teile des Hintergrundwissens, das während der Navigation durch Informationssysteme zum Einsatz kommt, aber zum Beispiel durch hierarchische Strukturen näherungsweise beschreiben, wie auch einige Forschungsergebnisse, die in Kapitel 3 vorgestellt werden vorschlagen. Durch eine hierarchische Einteilung aller Knoten im Graphen, der das Informationssystem abbildet, werden alle, durch das System abgebildete Konzepte in Relation zueinander gestellt.

Die Hierarchie ist dabei einer der zuvor besprochenen speziellen Graphen, nämlich ein Baum.

2.3 Objektorientiertes Design und Softwarearchitektur

Neben den Grundlagen der Navigation und den von den in Kapitel 3 vorgestellten Forschungsergebnissen abgeleiteten Anforderungen sind beim Entwurf eines Frameworks zur Navigationssimulation natürlich die Grundlagen des Objektorientierten Designs und der Softwarearchitektur weitere wichtige Punkte.

Eine bekannte Analogie, mit der zum Beispiel auch in [9] in das Thema der Softwarearchitektur eingeführt wird, stammt von Grady Booch. Er stellte einen Vergleich der Größenausmaße zwischen Bauprojekten und Softwareentwicklungsprojekten an.

Beim Bau eines Hundehauses ist es nicht unbedingt nötig genaue Pläne anzufertigen, um das Projekt erfolgreich zu Ende bringen zu können. Ist man handwerklich begabt, so reichen schon einige schnelle Abmessungen, die sicherstellen, dass die Hundehütte am Ende groß genug ist und der Hund auch tatsächlich durch die Tür des Hundehauses passt.

Beim Bau eines Wohnhauses ist wahrscheinlich Planungsarbeit notwendig und vor dem Baubeginn müssen einige Überlegungen gemacht werden. Aber auch ohne die detailierten Pläne eines Architekten kann der Bau eines Hauses funktionieren, wenn Bauarbeiter am Werk sind, die ihr Handwerk sehr gut beherrschen und genug Erfahrung haben.

Beim Erbauen eines Wolkenkratzers besteht jedoch kein Zweifel, dass eine Planung durch Architekten notwendig ist.

Bei Softwareprojekten verhält es sich sehr ähnlich. Auch wenn kleinere Projekte wahrscheinlich ohne Planung rein durch Erfahrung bezwingbar sind, so steigt mit der Größe des Projekts auch die Notwendigkeit Softwarearchitektur in den Entwicklungsprozess mit aufzunehmen.

2.3.1 Grundlagen des objektorientierten Paradigmas

Bevor man sich jedoch mit der Planung von Software beschäftigen kann, muss geklärt sein, mit welchen Mitteln gearbeitet wird. Im Speziellen ermöglicht der objektorientierte Softwareentwicklungs-Ansatz gute Designs und ist deshalb oftmals Grundlage von Softwaresystemen und infolgedessen Thema vieler Bücher wie zum Beispiel [10] und [11].

Bei der Methode des objektorientierten Ansatzes werden Daten und Methoden bzw. Funktionen, die mit diesen Daten arbeiten, in eine Einheit, die man als Objekt bezeichnet, zusammengefasst.

Vor diesem Ansatz wurde größtenteils die prozedurale Programmierung eingesetzt. Diese kann man im Allgemeinen auch als das Gegenteil der objektorientierten Programmierung ansehen. Die prozedurale Programmierung ergab sich aus dem Umfang bzw. den Einschränkungen der zur Verfügung stehenden Programmiersprachen. Auch der prozedurale Ansatz bietet Daten und Funktionen, jedoch besteht im Gegenteil zur objektorientierten Programmierung dabei kein Zusammenhalt zwischen diesen.

Neben den bereits erwähnten Objekten gibt es noch unzählige weitere Konzepte, die mit der objektorientierten Programmierung eingeführt wurden.

Mit der Frage, welche Konzepte die wichtigsten sind und daher die absoluten Grundlagen des objektorientierten Ansatzes darstellen, beschäftigt sich zum Beispiel Armstrong in [12]. Die Basiskonzepte des objektorientierten Paradigmas werden in der Arbeit identifiziert und beschrieben. Es wurde untersucht, welche Konzepte in der Literatur, die im Zusammenhang mit objektorientierter Entwicklung steht, am häufigsten vorkommen. Die acht laut [12] wichtigsten Konzepte bzw. Grundlagen des objektorientierten Paradigmas sind die folgenden:

- *Objekt*
  Wie schon zuvor erwähnt ist ein Objekt eine Einheit, die Daten und Methoden, die mit diesen Daten arbeiten, enthält. Objekte können identifi-

ziert werden und besitzen einen Zustand. Der Zustand von Objekten wird durch die Gesamtheit aller im Objekt befindlichen Daten bestimmt.

- *Klasse*
  Zur abstrakten Beschreibung von Objekten bzw. einem Set von Objekten wird das Konzept der Klasse eingesetzt. Klassen legen fest, welche Daten Objekte enthalten können und bestimmt außerdem, welche Funktionen bzw. Methoden ein Objekt umfasst. Klassen, die mit Daten gefüllt sind und deshalb einen Zustand besitzen, sind Objekte.

- *Vererbung*
  Das Konzept der Vererbung beschreibt den Umstand, dass die Implementierung einer Klasse auf der einer anderen existierenden Klasse basiert. Durch Vererbung entsteht also eine Hierarchie aus Klassen, wobei Klassen weiter unten in der Hierarchie speziellere Implementierungen bieten, als die abstrakteren Klassen die sich in der Hierarchie höher befinden. Wie der Name schon sagt, erben Klassen Funktionalität von Klassen, die in der Hierarchie über ihnen liegen und können diese für die eigene Implementierung nutzen.

- *Kapselung*
  Mit der Kapselung beschreibt man das Vorgehen, bei dem man die Idee der Klassen bzw. Objekte zum vollen Potential ausnutzt und Daten wirklich mit genau den Funktionen zusammen in Klassen verpackt, die sie manipulieren. Kapselung beschreibt also einen Design-Ansatz für Klassen, bei dem es um die Verbergung von Information, im Speziellen implementierungsspezifischer Information geht. Interaktion mit anderen Objekten von außen geschehen dabei über wohl definierte Schnittstellen.

- *Methode*
  Der Vorgang die Daten eines Objekts auszulesen oder zu verändern, basiert auf Methoden. Methoden oder Funktionen selbst umfassen den (Code-/Implementierungs-)Teil eines Objekts, der dafür zuständig ist, eine bestimmte Aufgabe zu erfüllen, wie zum Beispiel das bereits erwähnte Manipulieren von Daten des Objekts. Außerdem ist auch die oben be-

sprochene Schnittstelle zur Interaktion zwischen Objekten durch Methoden definiert.

- *Message Passing*
  Der sogenannte „Nachrichtenaustausch“ beschreibt die Kommunikation zwischen Objekten. Eine der einfachsten Formen der Kommunikation ist zum Beispiel ein Funktions- bzw. Methodenaufruf. Durch einen Funktionsaufruf wird ein Objekt von einem anderen Objekt benachrichtigt, dass die mit der Funktion verbundene Aktion ausgeführt werden soll. Mit einem Nachrichtenaustausch können auch Daten von einem Objekt zu einem anderen weitergegeben werden. Dies geschieht zum Beispiel durch die Parameter bei einem Methodenaufruf.

- *Polymorphismus*
  Unter dem Begriff Polymorphismus versteht man das Konzept, dass verschiedene Objekte trotz einer unterschiedlichen Implementierung auf dieselbe Nachricht (z.B. Methodenaufruf) reagieren. Verschiedene Objekte bieten also dieselbe Schnittstelle an, jedoch behandelt jedes Objekt die Nachricht auf seine eigene Art und Weise.

- *Abstraktion*
  Durch Abstraktion wird versucht Komplexität zu verbergen. Dies geschieht mittels Verallgemeinerung durch das Weglassen von Details, die für die aktuelle Betrachtungsweise nicht notwendig sind. Mit diesem Ansatz können auch unterschiedliche Objekte gemeinsam betrachtet werden indem man Gemeinsamkeiten identifiziert und Unterschiede weglässt.

Mit Hilfe des objektorientierten Ansatzes, dessen Kern-Konzepte hier auszugsweise erklärt wurden, stehen die Mittel zur Verfügung um strukturierte Softwaresysteme zu entwerfen. Beim Entwurf eines sehr komplexen Softwaresystems, wie es etwa ein Framework zur Modellierung von Nutzernavigation darstellt, ist das Zusammenspiel all dieser Konzepte von zentraler Wichtigkeit.

2.3.2 Softwarearchitektur

Am Beginn eines Softwareprojekts stehen immer Anforderungen. In irgendeiner Form bestehen Ansprüche bezüglich Funktionalität, die von der Software erfüllt werden sollen. Außerdem gibt es nicht-funktionale Anforderungen, die zum Beispiel die Performance oder ähnliches betreffen.

Die Aufgabe von Software-Design oder dem Design der Softwarearchitektur besteht darin, Softwareteile zu schaffen, die jeweils eine Aufgabe haben und im Zusammenspiel die Anforderungen erfüllen.

Beim Designen von Software gibt es einige Prinzipien, die befolgt werden sollten. Diese Prinzipien gibt es schon immer, aber erst über die Zeit hat man diese Design-Prinzipien unter einem Begriff, nämlich Softwarearchitektur, zusammengefasst.

Mittlerweile befasst sich auch sehr viel Literatur mit diesem Teilgebiet der Softwareentwicklung. Die Grundlagen der Softwarearchitektur werden zum Beispiel in [9] besprochen.

Softwarearchitektur befasst sich mit der Struktur von Softwaresystemen, also der Zusammensetzung der einzelnen Systemteile bzw. Komponenten. Durch das Zusammenspiel aller Komponenten entsteht ein Systemverhalten, das im Idealfall die gestellten Anforderungen erfüllt.

Die Wichtigkeit von Softwarearchitektur im Feld der Softwareentwicklung ist heute weithin anerkannt. [13] zum Beispiel hält fest, dass bei komplexen und teuren Softwaresystemen die Analyse und das Design der Architektur Schlüsselfaktoren im Softwareentwicklungsprozess sein müssen. Wenn nicht genug Wert auf eine gute Softwarearchitektur gelegt wird, hat dies oft im Nachhinein zur Folge, dass große Teile der Software überarbeitet werden müssen. [14] beschreibt, wie das Design von Softwaresystemen so angepasst werden kann, dass es aktuellen Standards entspricht. Krikhaar et al. beschreiben dazu eine Zwei-Phasen-Methode. Im Zuge der Beschreibung des Prozesses zur Architekturverbesserung stellen sie dabei die Frage wie man Softwarequalität und deren Verbesserung überhaupt messen kann.

Um die Entwicklung von Architekturen zu unterstützen stehen auch verschiedene Hilfsmittel zur Verfügung. Neben den verschiedensten UML[1]-Diagrammen werden auch verschiedene Werkzeuge zur Architektur-Prototyp-Erstellung, wie zum Beispiel in [15] vorgestellt, eingesetzt.

2.3.3 Architekturmuster

Da durch eine Architektur die grundlegende Struktur bzw. der grundlegende Stil von Softwarekomponenten beschrieben wird, kann Software auf diese Weise aus einem recht abstrakten Blickwinkel untersucht werden. Je nach Zweck der Software existieren einige immer wiederkehrende Softwarearchitekturen, die man als Architekturmuster bezeichnet.

Für interaktive Anwendungen wie zum Beispiel Web-Applikationen wird immer wieder das Model-View-Controller-Muster eingesetzt. Dabei wird die Software in drei Hauptkomponenten, Datenrepräsentation/Datenmodell, Präsentation und Programmlogik/Steuerung, aufgeteilt.

Zur Verarbeitung von Datenströmen wird oft das Pipes-und-Filter-Muster eingesetzt. Dabei handelt es sich um eine Abfolge von Verarbeitungseinheiten mit Dateneingaben und Ausgaben, die man als Filter bezeichnet. Diese Filter sind über die sogenannten Pipes verbunden.

Für Netzwerk-Anwendungen gibt es unter anderem das Peer-To-Peer und das Client-Server-Muster. Bei der Peer-To-Peer-Architektur ist das System so konzipiert, dass alle teilnehmenden Systemkomponenten gleichberechtigt sind. Das heißt jeder kann sowohl Dienste von anderen in Anspruch nehmen als auch selbst Dienste anbieten. Im Gegensatz dazu ist im Client-Server-Modell festgelegt, dass Server Dienste zur Verfügung stellen und Clients diese lediglich benutzen.

Auch für Frameworks existieren wiederkehrende Muster auf die im Abschnitt 2.4 eingegangen wird.

---

[1] Unified Modeling Language

2.3.4 Entwurfsmuster

Neben den Architekturmustern gibt es auch die etwas weiter verbreiteten Software-Entwurfsmuster oder, wie sie auf Englisch genannt werden, „design patterns“. Diese beschreiben bewährte Lösungen von Softwareproblemen auf Klassenebene. Eine große Auswahl an „klassischen“ Entwurfsmustern werden von der sogenannten Viererbande in [16] vorgestellt.

Einige der Muster, die im Zuge dieses Projekts zum Einsatz kamen, sind die folgenden:

- *Fabrikmethode*
  Beim Muster der Fabrikmethode wird ein Objekt, anstatt direkt über die Benutzung eines Konstruktors, über einen Methodenaufruf erzeugt. Ein großer Vorteil dieses Musters liegt darin, dass die Klasse, die das Objekt von der anderen Klasse, der sogenannten Fabrik, erzeugen lässt, kein Wissen über die Konstruktion des Objektes benötigt. Alleine die Fabrik kümmert sich um die Konstruktion des Objektes. Dies geht soweit, dass nur die Fabrik überhaupt die konkrete Klasse des zu erzeugenden Objektes kennen muss, was in Abbildung 7 durch die Klassen „Produkt“ und „KonkretesProdukt“ dargestellt ist. Auf Grund seiner Eigenschaften wird dieses Entwurfsmuster sehr häufig im Design von Frameworks eingesetzt.

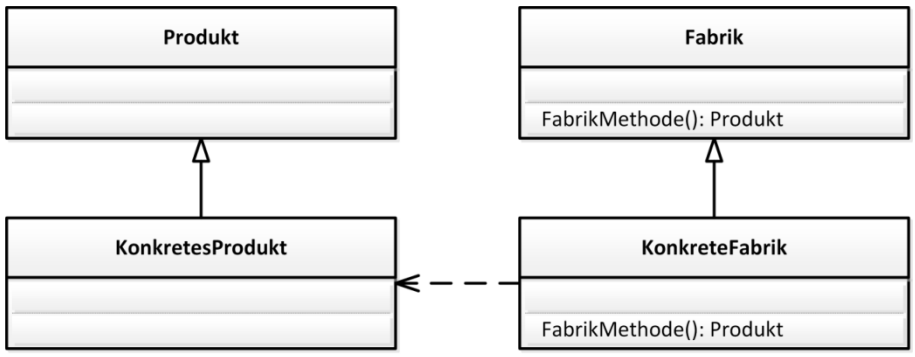

Abbildung 7: Fabrikmethode

- *Adapter*
  Beim Adapter handelt es sich um eine Klasse, die sich um eine andere Klasse, Schnittstelle, Bibliothek etc. hüllt. Durch einen Adapter können

somit normalerweise inkompatible oder unbekannte Klassen über eine bekannte Schnittstelle benutzt werden. In Abbildung 8 ist zu sehen, dass ein Adapter die Funktionalität einer unbekannten Klasse nutzbar macht.

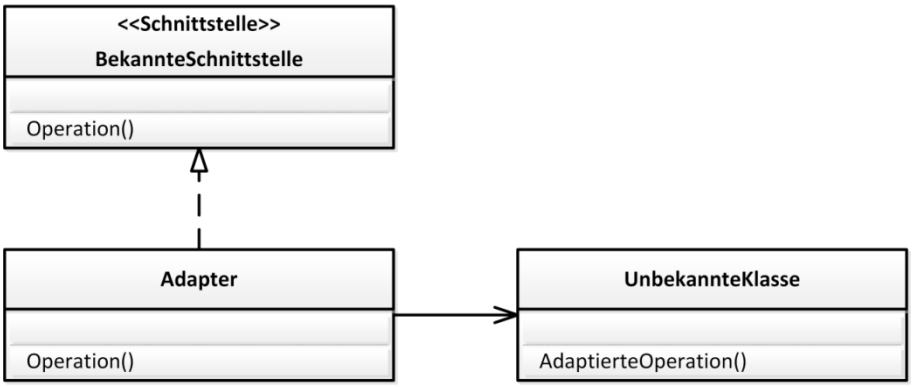

Abbildung 8: Adapter

- *Iterator*

  Der Iterator ist eine Klasse, die eine Schnittstelle zur Verfügung stellt über die eine Sammlung von Objekten durchlaufen werden kann ohne über die zu Grunde liegende Struktur der Sammlung Bescheid wissen zu müssen. Damit können zum Beispiel alle Knoten eines Graphen durchlaufen werden ohne nähere Informationen über die Struktur des Graphen zu benötigen. Abbildung 9 zeigt eine mögliche Form in der das Iterator-Muster auftreten kann.

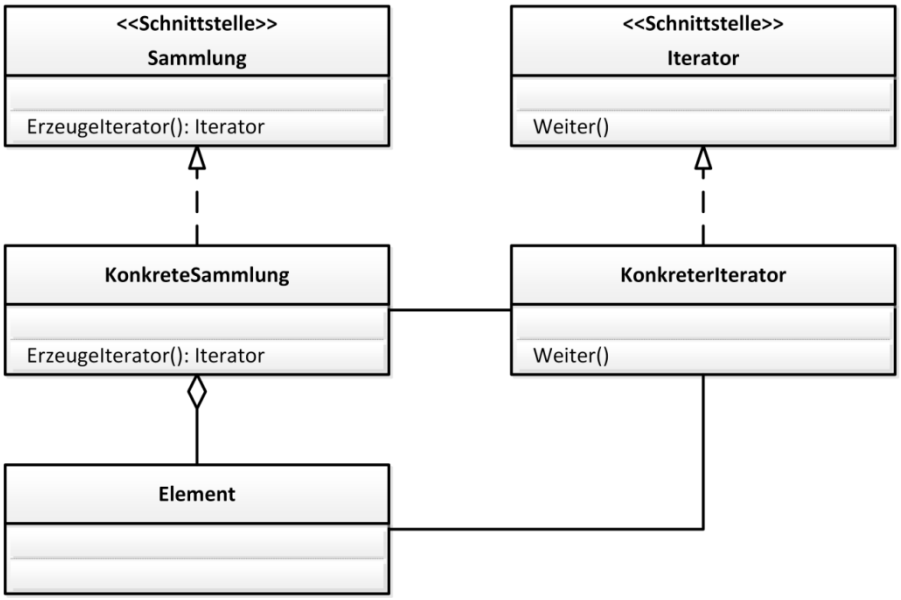

Abbildung 9: Iterator

- *Schablonenmethode*

  Eine abstrakte Klasse definiert den Ablauf eines Algorithmus, wobei die einzelnen Teile des Algorithmus erst später durch eine konkrete Implementierung festgelegt werden und in der abstrakten Klasse variabel sind. Abbildung 10 zeigt das Schablonenmethoden-Muster anhand der beiden Klassen „AbstrakteBasis“ und „KonkreteImplementierung“. Die Funktion „SchablonenMethode“ nutzt hier die beiden Methoden „Schritt1“ und „Schritt2“. Diese beiden Methoden sind jedoch erst in der Klasse „KonkreteImplementierung“ implementiert.

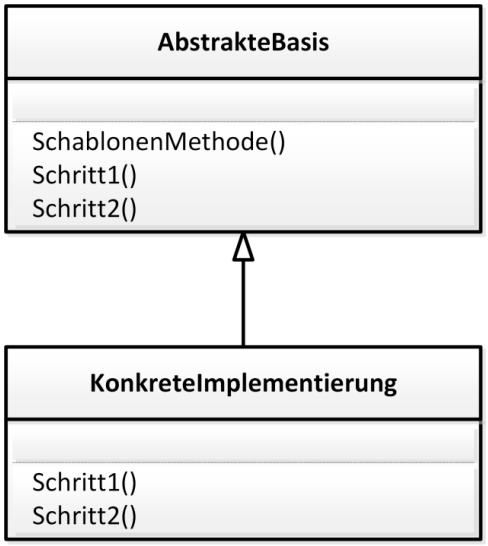

Abbildung 10: Schablonenmethode

- *Strategie*

  Beim Strategie-Muster wird ein Algorithmus, der in einer Klasse benötigt wird, nicht durch die Klasse selbst implementiert, sondern in ein eigenes Objekt, das als Member gehalten wird, ausgelagert. Auf diese Weise ist der Algorithmus austauschbar und es reicht wenn die Klasse, die den Algorithmus benötigt, ein Interface kennt. Ein Wissen über die konkrete Implementierung des Algorithmus ist nicht nötig. Dieses Muster wird öfters mit dem Fabrikmethoden-Muster kombiniert, wobei die Fabrik dazu benutzt wird, um eine Strategie zu konstruieren. Die Klasse „Kontext“ aus Abbildung 11 zum Beispiel benötigt für die „NutzeAlgorithmus“-Methode eine Implementierung eines Algorithmus, der durch ein „Strategie“-Objekt zur Verfügung gestellt wird.

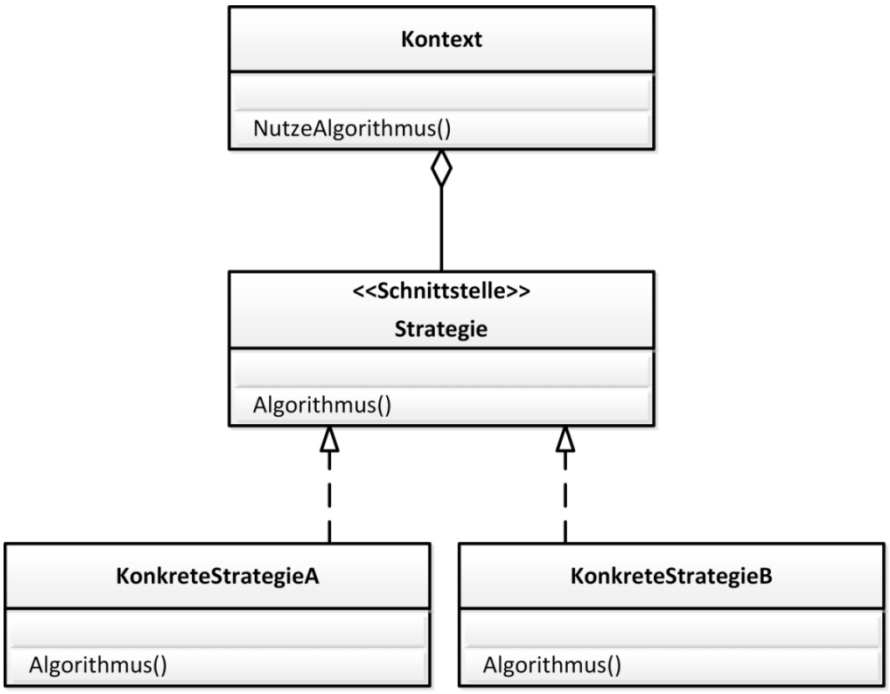

Abbildung 11: Strategie

2.4 Frameworks

Der Begriff Framework beschreibt im Bereich der Softwareentwicklung, wie der Name schon sagt, ein Grundgerüst. Diese spezielle Softwareform ist selbst kein komplettes, selbstständiges Softwareprodukt, sondern bietet in abstrahierter, generischer Form eine bestimmte Funktionalität. Diese Funktionalität kann durch Programmcode, den der Nutzer des Frameworks schreibt, so abgeändert und an vorhandene Bedürfnisse angepasst werden, dass ein Softwareprodukt entsteht, das eine bestimmte Aufgabe erfüllt.

Der Vorteil liegt darin, dass durch die vorhandene Codebasis, die ein Framework bietet, schon viele Bereiche abgedeckt sind, die ohne die Verwendung eines Frameworks vom Benutzer selbst zu implementieren wären. Außerdem fällt durch die Verwendung eines Frameworks das Erlernen einer bestimmten Technologie oft leichter, da nicht jeder kleine Teilaspekt dieser Technologie sofort erlernt werden muss, wenn das Framework diesen schon ausreichend abdeckt.

Des Weiteren bieten Frameworks durch ihre Wiederverwendbarkeit für mehrere ähnliche Projekte oder Aufgaben eine große Ersparnis im Arbeitsaufwand. Im

Idealfall bieten Frameworks genau diese Funktionalität, die all diese ähnlichen Projekte gemeinsam haben, und somit bleibt als Implementierungsaufwand nur genau der Teil der Anwendungen übrig, der sich von den anderen Projekten unterscheidet. Frameworks folgen also in großem Ausmaß dem Prinzip der Wiederverwendbarkeit aus dem objektorientierten Softwareentwicklungs-Paradigma.

Auch im Bereich der Sicherheit können Frameworks eine große Erleichterung darstellen. Unerfahrene Entwickler können zum Beispiel weniger sicherheitsrelevante Fehler machen, wenn diese Dinge bereits vom Framework bereitgestellt werden und robust implementiert sind.

2.4.1 Merkmale

Über die Bezeichnung Framework ist ein sehr breites Gebiet abgedeckt und es existieren auch einige verwandte Begriffe wie zum Beispiel die Programmbibliothek. Im Unterschied zu Bibliotheken wird die Struktur von Programmen, die auf Basis von einem Framework entstehen, jedoch durch das Framework beeinflusst. Durch das Framework ist eine Architektur vorgegeben und der Nutzer des Frameworks kann an bestimmten Stellen Codeteile bereitstellen, die dann vom Framework in den Kontrollfluss eingebunden werden. Bei Bibliotheken hat der Benutzer die Kontrolle über den gesamten Ablauf des Programms und nutzt lediglich die von der Bibliothek bereitgestellten Klassen und Funktionen. Dieses Grundprinzip der Funktionsweise von Frameworks bezeichnet man als „Inversion of Control“, zu Deutsch „Umkehrung der Steuerung“. Oft spricht man in diesem Zusammenhang auch vom Hollywood-Prinzip [17]. Dieses lautet „Rufen Sie uns nicht an, wir rufen Sie an“. Nach diesem Prinzip stellt der Nutzer eine konkrete Implementierung von bestimmten Programmteilen zur Verfügung, die dann vom Framework benutzt werden.

Ein Framework ist immer für die Erweiterbarkeit konzipiert. Meist geschieht dies durch Vererbung und das gezielte Ersetzen, Überschreiben und Erweitern von Funktionalitäten durch den Benutzer. Außerdem bieten Frameworks Schnittstellen, die implementiert werden können. Diese Implementierungen werden dann

zum Beispiel beim Framework registriert und dann vom Framework gesteuert und an der entsprechenden Stelle im Ablauf des Frameworks eingesetzt.

2.4.2 Arten

Da die grundlegende Architektur vom Framework selbst festgelegt wird, sind Frameworks immer für bestimmte Aufgaben in einer bestimmten Problemdomäne konzipiert. Auf Grund dieser Spezialisierung auf ein bestimmtes Anwendungsgebiet bzw. auf einen bestimmten Anwendungstyp gibt es viele verschiedene Frameworks, die jeweils unterschiedliche Zwecke erfüllen.

Ein Beispiel für einen sehr weit verbreiteten Framework-Typ sind Webframeworks. Diese stellen ein Softwaregrundgerüst zur Erstellung von dynamischen Webseiten bzw. Webanwendungen zur Verfügung.

Ein Beispiel eines solchen Webframeworks ist CodeIgniter[2]. CodeIgniter ist ein Framework für die Erstellung von Webanwendungen in PHP. Es ist bewusst klein gehalten um gute Performance zu bieten und gleichzeitig leichter erlernbar zu sein. Die eingesetzte Softwarearchitektur ist im Falle von CodeIgniter das Model-View-Controller-Muster. Außerdem bietet das Framework zahlreiche Funktionen, wie zum Beispiel Datenbankzugriff und Eingabeüberprüfung, die bei der Webentwicklung häufig benötigt werden.

Ein weiteres sehr bekanntes Webframework ist Ruby on Rails[3]. Auch Ruby on Rails, kurz Rails, folgt dem Model-View-Controller-Muster und ist ein Gerüst zur Erstellung von Webanwendungen in der Programmiersprache Ruby.

Testframeworks wie zum Beispiel JUnit[4] sind ein weiterer Framework-Typ mit spezieller Funktion. Sie dienen dazu Software mit möglichst geringem Aufwand von Seiten des Programmierers zu testen. Durch die Verwendung eines Testframeworks müssen lediglich Testfälle definiert werden. Die Ausführung und Auswertung der Tests erfolgt durch das Framework.

---

2 http://codeigniter.com/

3 http://rubyonrails.org/

4 http://junit.sourceforge.net/

Ein weiterer Typ sind allgemeine Anwendungs- bzw. domänenspezifische Frameworks. Diese bieten das Gerüst für Anwendungen, die zum Beispiel aus derselben Problemdomäne stammen.

2.4.3 Muster

Wie bereits erwähnt, findet sich in Webframeworks oft das Model-View-Controller-Muster wieder. Außerdem basieren die meisten Frameworks auf dem zuvor besprochenen Prinzip der „Inversion of Control“. Ein weiteres Muster im Bereich der Frameworks ist die so genannte Dependency Injection, zu Deutsch Abhängigkeits-Injektion. Die Bezeichnung dieses Musters wurde von Martin Fowler eingeführt. [18]

Das Muster beschreibt das Konzept, dass Objekte Daten bzw. andere Objekte von denen sie abhängig sind zur Laufzeit zur Verfügung gestellt bekommen und nicht selbst erstellen. Dadurch ist es möglich, dass die Verwaltung und Konstruktion von Objekten an einem zentralen Ort zusammengefasst werden kann.

Dependency Injection ist stark mit dem Muster der „Inversion of Control“ verwandt, bezieht sich aber eher auf die Erstellung von Objekten als auf den allgemeinen Programmfluss.

Frameworks, die Dependency Injection benutzen, sind unter anderen Spring[5], PicoContainer[6] und das Zend Framework[7].

2.5 Stanford Network Analysis Platform

Zusätzlich zum Entwurf einer passenden Softwarearchitektur muss das Navigationsframework auch in der Lage sein mit großen Datenmengen umgehen zu können.

---

5 http://www.springsource.org/

6 http://picocontainer.codehaus.org/

7 http://framework.zend.com/

Ein effizienter Umgang mit Netzwerken, die eine Dateigröße von mehreren Gigabyte haben und mehrere Millionen Knoten und mehrere hundert Millionen Kanten besitzen, ist wünschenswert.

Es ist somit Software nötig, die mit solchen Datenmengen effizient umgehen kann bzw. die es überhaupt schafft, Informationen in diesen Größenordnungen zu bewältigen.

Eine Bibliothek, die dafür geeignet ist, ist die Stanford Network Analysis Platform[8], kurz SNAP. SNAP wird seit dem Jahr 2004 an der Universität von Stanford entwickelt. Die Bibliothek ist in C++[9] geschrieben und wuchs seit ihren Anfängen durch die Anforderungen, die verschiedene Forschungsprojekte stellten.

Die Version 1.0 von SNAP wurde im November 2009 auf der Projekt-Webseite veröffentlicht. Die Bibliothek wird unter der BSD-Lizenz zum Herunterladen angeboten.

Mittlerweile ist SNAP auch als Projekt auf GitHub[10] verfügbar.

SNAP ist so konzipiert, dass die Skalierbarkeit zu Netzwerken in den Ausmaßen von hundert Millionen Knoten und Milliarden an Kanten gegeben ist.

Ein Beispiel, bei dem SNAP für ein derartig großes Netzwerk eingesetzt wurde, ist die Untersuchung des Microsoft Sofortnachrichtendienst-Netzwerks mit 240 Millionen Knoten und 1,3 Milliarden Kanten. [19]

SNAP bietet unter anderem Funktionalitäten zum Umgang mit gerichteten und ungerichteten Graphen. Wichtig dabei ist, dass, wie bereits erwähnt, auch recht große Graphen unterstützt werden. Dieser Umstand hat dazu geführt, dass SNAP als die Bibliothek ausgewählt wurde auf die das Navigationsframework aufbaut.

---

8 http://snap.stanford.edu/

9 Programmiersprache - http://isocpp.org/

10 https://github.com/snap-stanford/snap

3. Related Work

In diesem Kapitel werden einige Ergebnisse und Erkenntnisse angeführt, die von anderen Wissenschaftlern auf dem Gebiet der Nutzernavigation erzielt wurden. Durch das Verständnis bisheriger Forschungsergebnisse kann ein Ansatz für den Entwurf eines Frameworks zur Simulation von menschlicher Navigation auf Basis der in Kapitel 2 vorgestellten Grundlagen gefunden werden, der es auch tatsächlich ermöglicht realistische Ergebnisse zu liefern.

3.1 Benutzernavigationsverhalten

Menschen, die auf der Suche nach Informationen sind, stehen einige grundlegende Strategien zur Verfügung um an die gewünschten Daten zu gelangen. In spezialisierten Informationssystemen oder dem World Wide Web im Allgemeinen kann man hier zwei besonders häufige Ansätze unterscheiden. Zum einen kann eine eventuell vorhandene Suchfunktion benutzt werden. Dabei versucht man durch Ausdrücken seines Informationsbedürfnisses, zum Beispiel durch Angabe von Schlagworten, zur gewünschten Information zu gelangen. Je nach Informationssystem, das durchsucht werden soll, stehen hierzu verschieden Möglichkeiten offen. Im Falle des World Wide Webs stehen hier unter anderem Suchmaschinen wie zum Beispiel Google[11] zur Verfügung.

Zum Thema Suchverhalten von Menschen wurde eine Studie in [20] vorgestellt. Die Nutzerstudie mit 2000 Teilnehmern untersuchte über einen Zeitraum von fünf Monaten die Unterschiede und Gemeinsamkeiten von Suchanfragen von verschieden Personen. Die Ergebnisse der Studie sollen dazu beitragen bessere Werkzeuge zur Unterstützung von Suchprozessen im Web zu entwickeln.

Der zweite Ansatz neben der Suche ist die direkte Navigation durch die Verfolgung von Hyperlinks. Diese Methode ist die ältere der beiden Strategien und besteht schon seit Beginn des World Wide Web, als noch keine Suchmaschinen existierten. [1]

[11] http://www.google.com/

In [21] wird dazu ein Modell von menschlicher Navigation vorgestellt, bei dem davon ausgegangen wird, dass ein Nutzer so lange Links verfolgt, solange die Wertigkeit der nächsten Seite einen gewissen Schwellenwert überschreitet. Die Wertigkeit von Seiten wird dabei von den Nutzern vor jedem Schritt abgeschätzt. Wenn man zum Beispiel auf der Suche nach Informationen zur Stadt London ist, wird ein Link auf eine Seite über Großbritannien eine hohe Wertigkeit besitzen, da Großbritannien mit London assoziiert werden kann.

Der Grund, warum die Suche hier als erstes, noch vor der Navigation angeführt wurde, liegt darin, dass es seit den Anfängen des WWW eine Verschiebung von der reinen Navigation hin zur suchgestützten Navigation gegeben hat, sodass diese heute die dominante Form der Wissensbeschaffung darstellt.

Meist wird der Ansatz über die Suche jedoch mit der eben besprochenen „klassischen“ Navigation kombiniert (siehe [22]), da eine Suche den Nutzer nicht immer direkt zur gewünschten Information bringt. Ein beispielhafter Ablauf könnte so aussehen, dass der Nutzer zuerst eine Suche ausführt. Durch die Suche gelangt er in die Nähe der gewünschten Information und navigiert daher über vorhandene Links zum Ziel oder er bewegt sich in der Umgebung des Suchergebnisses, um genug Information zu sammeln um danach eine besser formulierte Suchanfrage starten zu können.

Genau dieser Umstand, nämlich, dass eine Suche alleine nicht immer die gewünschten Ergebnisse liefert oder überhaupt unmöglich ist, wird in [23] und [20] aufgezeigt. In der in [23] vorgestellten Studie benutzten nur 39% der Teilnehmer eine Stichwortsuche zum Auffinden einer Information. Dies wird auch durch die Untersuchung, in welchem Umfang Navigation und Suche kombiniert werden, in [24] bestätigt. Laut diesen Ergebnissen wird im Web mit einer mittleren Wahrscheinlichkeit zwischen 60% und 72% „klassisch“ navigiert, indem Links angeklickt werden.

3.2 Hintergrundwissen

Laut [25] kann man die Aufgabe, die ein Mensch hat, wenn er in einem Informationssystem navigiert, als das Finden eines Pfades zwischen zwei Knoten

auffassen. Menschen besitzen zwar ein komplexes Hintergrundwissen, das sie während der Navigation leitet, aber die Autoren zeigen auf, dass eine automatische Navigation auch ohne ein Hintergrundwissen solcher Komplexität möglich ist bzw. sogar bessere Ergebnisse liefert als Nutzer.

Auch [26] beschäftigt sich mit versteckten Hintergrundmetriken, die ein Netzwerk begleiten. Laut den Autoren würden die versteckten Informationen aus dem „hidden metric space“ sehr gute Suchstrategien für Suchen liefern, die nur auf lokalen Informationen basieren.

Kleinberg beschreibt in [27] hierarchische Strukturen und wie Distanzen in diesen eine wertvolle Information darstellen. Hierarchien stellen gute Modelle dar, mit denen eine bestimmte Art von Hintergrundwissen abgebildet werden kann.

3.3 Kleine-Welt-Phänomen

Viele der Forschungsarbeiten im Bereich der Benutzernavigation in den verschiedensten Netzwerken gehen auf die Arbeit von Milgram zurück. In [28] stellte der amerikanische Psychologe seine Untersuchungen zur sozialen Verflechtung unserer Gesellschaft vor. In seinem Experiment ging es darum einen Brief über eine Kette von Personen an eine Zielperson weiterzuleiten. Von einer Startperson ausgehend musste jeder Teilnehmer den Brief an eine Person aus seinem Bekanntenkreis weiterleiten, von der er glaubte, dass sie den Brief näher zur Zielperson befördern kann. Der Empfänger des Briefes musste ihn wieder an jemanden aus seinem Bekanntenkreis weiterleiten. Auf diese Art entstand eine Kette aus Personen, die idealerweise bei der Zielperson endete.

Das Kleine-Welt-Phänomen entstand aus den Ergebnissen dieses Experiments. Die durchschnittliche Länge der Briefketten betrug sechs, obwohl sich die Start-Personen über die USA verteilt befanden. Die Person, die den Brief als erste zur Weiterleitung erhielt, war also nur durch sechs Schritte von der Zielperson getrennt. Aus diesem Grund bezieht man sich auf das Kleine-Welt-Phänomen (englisch „small world phenomenon“) oft auch unter der Bezeichnung „six

degrees of seperation“. Alle stark verknüpften Netzwerke, in denen es trotz der Größe des Netzwerks recht kurze Pfade zwischen zwei beliebigen Knoten gibt, können also in Anlehnung an Milgrams Experiment als „kleine Welt“ eingestuft werden.

2003 wurde Milgrams Experiment von Dodds, Muhamad und Watts unter leicht veränderten Bedingungen wiederholt. Die Veränderungen bestanden zum Beispiel darin, dass die Start- und Zielpersonen zufällig auf der ganzen Welt verteilt waren. Die in [29] veröffentlichten Ergebnisse bestätigten die Resultate von Milgram im Allgemeinen. Um genauere Analysen machen zu können, mussten die Teilnehmer des Experiments bei jeder Weiterleitung den Grund für die Auswahl der Person, an die die Nachricht weitergeleitet wurde, angeben. So konnten die Forscher feststellen, dass vor allem am Beginn einer Kette die Weiterleitung sehr stark durch geografische Faktoren bestimmt war. Das heißt eine Weiterleitung erfolgte an Personen, die geografisch näher an der Zielperson waren.

Weiters beschäftigte sich [29] auch mit der Abbruchrate, „attrition rate“ genannt. Es wurde untersucht, aus welchem Grund Briefketten unterbrochen wurden und in welchem Schritt. Basierend auf den Ergebnissen dieser Untersuchungen ist es für die Simulation von Nutzerverhalten während einer Navigation also wünschenswert auch eine Abbruchrate zu modellieren.

Auch Kleinberg beschäftigte sich in [27] mit dem Kleine-Welt-Phänomen und der Struktur von Kleine-Welt-Netzwerken. Außerdem bespricht er auch, welche Methoden angewandt wurden, um die Briefe in Milgrams Experiment weiterzuleiten und wie die kürzesten Briefketten gefunden werden könnten. Sollte der Brief das Ziel schnellstmöglich erreichen, so hätte jeder Briefempfänger eine Weiterleitung an all seine Kontakte ausführen müssen. Dies war aber nicht möglich und es wurde immer nur eine Weiterleitung pro Person durchgeführt.

3.4 Decentralized Search

Die Teilnehmer der zuvor erwähnten Experimente waren in der Entscheidung, an wen sie die Nachricht weiterleiten, auf ihren Bekanntenkreis eingeschränkt.

Die Zielperson war zum Beispiel bei Milgrams Experiment [28] jeder Person in der Briefkette inklusive der Adresse bekannt. Der Brief musste aber an eine Person aus dem eigenen Bekanntenkreis weitergeschickt werden und durfte nicht direkt an die Zielperson gesendet werden, außer diese zählte zu den eigenen Bekanntschaften.

Diese Einschränkung der möglichen Weiterleitungs-Kandidaten an lokale Auswahlmöglichkeiten ist das Konzept der dezentralen Suche und wird unter anderem in [27] besprochen. Dabei ist keine globale Information über das Netzwerk vorhanden. In jedem Schritt wird eine der vom aktuellen Knoten ausgehenden Kanten ausgewählt, solange bis das Ziel erreicht ist. Zur Auswahl der ausgehenden Kante gibt es die unterschiedlichsten Strategien.

Adamic und Adar haben in [30] zwei Experimente zum Kleine-Welt-Phänomen durchgeführt und dabei verschiedene Navigationsstrategien getestet. In einem der beiden Experimente haben sie ein Netzwerk aus E-Mail-Kontakten in einem Unternehmen benutzt um Nachrichten zu einer Zielperson weiterleiten zu lassen. Dann wurde überprüft, wie gut die von den Teilnehmern berichteten Strategien zur Weiterleitung einer Nachricht auf das entsprechende Netzwerk anwendbar sind. Das Netzwerk der E-Mail-Kontakte umfasste 430 Personen, in dem jedes mögliche Personenpaar im Schnitt durch drei Kanten getrennt war. Die Kanten im Netzwerk wurden immer zwischen zwei Personen eingefügt, die innerhalb von drei Monaten zumindest sechs E-Mails an die andere Person geschickt und mindestens sechs E-Mails von der anderen Person erhalten haben. Durch diese Abschätzung von sozialen Kontakten ergaben sich im Durchschnitt 13 Bekanntschaften pro Person.

Basierend auf der tatsächlichen Anzahl von E-Mails, die zwischen zwei Personen in den drei Monaten ausgetauscht wurden, kann man die Kanten im Netzwerk wie auch in [31] beschrieben in schwache und starke Verbindungen einordnen. Schwache Verbindungen stellten sich dabei als sehr wichtig heraus, da sie sehr stark verknüpfte Teile des Netzwerks (z.B. die einzelnen Abteilungen des Unternehmens) miteinander verbanden.

Es wurden drei Strategien zur Nachrichtenweiterleitung untersucht und die Ergebnisse in [30] vorgestellt. Die Strategien unterschieden sich in der Auswahl der Personen an die eine Nachrichtenweiterleitung erfolgte. Als Empfänger der Nachricht kamen Personen basierend auf diesen Kriterien in Frage:

- Person mit den meisten Bekanntschaften.
- Person, die sich in der hierarchischen Struktur des Unternehmens am nächsten zur Zielperson befindet.
- Person, deren Arbeitsplatz physikalisch bzw. geographisch zur Zielperson die kleinste Distanz hat.

Die Strategie der Weiterleitung der Nachricht zu der Person mit dem höchsten Grad bzw. der Person mit den meisten Bekanntschaften stellte sich als nicht geeignet heraus. Nachrichten an Personen, die nicht gut verlinkt waren und keinen Nachbar mit hohem Grad hatten, erreichten nur selten das Ziel.

Das Leiten der Nachrichten über Personen, die in der Unternehmenshierarchie die geringste Distanz zur Zielperson hatten, funktionierte sehr gut.

Die Orientierung an der physikalischen Distanz funktionierte, benötigt im Schnitt aber länger, d. h. die entstandenen Pfade vom Startpunkt der Nachricht zur Zielperson waren länger.

Es gibt im Zuge der dezentralen Suche also verschieden Strategien, die teilweise unterschiedliches Hintergrundwissen voraussetzen, aber auch verschieden gute Ergebnisse liefern.

3.5 Wikipedia

Da das in dieser Arbeit entwickelte Framework unter anderem auch für den Einsatz zur Simulation von Navigation im Umfeld von Wikipedia gedacht ist (siehe Fallbeispiel in Kapitel 6), sind Forschungsergebnisse auch in diesem Bereich durchaus interessant um Anregungen zum Design des Frameworks zu liefern.

2005 analysierte Voss in [32] den Stand der Forschung zum Wikipedia-Netzwerk. Er gab einen Überblick über die Grundideen und die Struktur, untersuchte das Wachstum sowie weitere Metriken.

Ein Jahr nach Voss wurde Wikipedia wiederum untersucht. In [33] befassten sich die Autoren mit dem Graphen, der entsteht, wenn Artikel mit Knoten und Hyperlinks mit Kanten abstrahiert werden. Vor allem die Entwicklung und die zeitlichen Unterschiede im Graphen wurden von den Autoren erforscht.

Nur wenn die Inhalte, die Wikipedia bietet, qualitativ hochwertig sind, ist der Wikipedia-Graph interessant und keine zufällige Ansammlung an Knoten und Kanten. Ein Ansatz, um die Qualität von Wikipedia Artikeln zu bestimmen, wird dafür in [34] präsentiert.

3.5.1 Navigation im Wikipedia-Netzwerk

West und Leskovec [35] haben 2012 menschliches Navigationsverhalten in einem Teilgraphen von Wikipedia mit rund 4000 Artikeln und 120000 Links untersucht. Sie analysierten dazu 30000 Nutzerpfade von ungefähr 9400 verschiedenen Benutzern, die durch das Spielen eines Navigationsspieles über den Wikipedia-Teilgraphen entstanden sind. Ihre Analysen ergaben unter anderem, dass sogenannte Hubs, also Seiten mit hohen Ein- und Ausgangsgraden am Beginn der Navigation eine wichtige Rolle spielen. Neben dem Beginn wurde auch das Ende von Navigationspfaden genau analysiert. Mit den Ergebnissen der Analysen wurden einige Modelle, mit denen die Vorhersage des Zieles eines Navigationspfades bereits während der Navigation möglich ist, vorgeschlagen.

Auch Helic hat sich in [36] mit Nutzerklickpfaden im Wikipedia-Netzwerk beschäftigt. Ein sehr interessanter Punkt aus dieser Arbeit ist die Erkenntnis, dass Nutzer, die auf der Suche nach einem vorgegebenen Ziel mit einer Zwei-Phasen-Strategie durch das Wikipedia-Netzwerk navigieren. In der ersten Phase wird versucht Hubs, also Knoten mit sehr hohem Grad, zu erreichen. Ausgehend von diesen Knoten, die im Kern des Netzwerks liegen, startet die

zweite Phase, in der die Nutzer in die nähere Umgebung des Zieles und schlussendlich zum Zielknoten navigieren.

4. Daten und Anforderungen

Aus den in Kapitel 3 vorgestellten Ergebnissen von bisherigen Arbeiten kann ein Ansatz für den Entwurf des Navigationsframeworks abgeleitet werden.

Die dezentrale Suche unter Berücksichtigung von hierarchischem Hintergrundwissen stellt sich als vielversprechender Ansatz heraus.

4.1 Anforderungen an das Navigations-Framework

Wie schon in Abschnitt 2.3.2 besprochen, bestehen am Beginn von Softwareprojekten immer Anforderungen. Es wird angenommen, dass die Softwarelösung diese Anforderungen zum Abschluss des Projektes erfüllt.

Bei diesem Projekt entstand ein großer Teil der Anforderungen durch die Analyse der in Kapitel 3 vorgestellten Forschungsergebnisse. Aus dem Wissen, welche Ansätze schon versucht wurden und auf welche Art es bisher gelang Nutzernavigation annähernd nachzubilden, können Anforderungen abgeleitet werden.

Demnach soll das Framework dezentrale Suche und hierarchisches Hintergrundwissen unterstützen.

Weitere Forderungen entstanden auf Grund der Projekte von Geigl in [3] und Horwath in [4]. Außerdem mussten Datenformate definiert werden, mit denen das Framework arbeiten soll. Die kommenden Abschnitte beschreiben diese vom Framework benutzten Datenformate.

4.2 Navigations-Netzwerk

Um die Navigationssimulation durchführen zu können, muss ein Netzwerk aus Knoten und Kanten gegeben sein, durch das navigiert werden kann. Um einen solchen Graphen als Input für das Framework liefern zu können, muss ein bestimmtes Datenformat eingehalten werden.

Dieses Datenformat für Graphen, das vom Framework benutzt wird, ist ein Klartext-Format. Graphen werden als sogenannte Kantenlisten abgespeichert.

Die Datei besteht dabei, wie der Name schon sagt, aus einer Liste von Kanten. Mit jeweils einer Kante aus dem Graphen pro Zeile sind nur die Kanten des Graphen in dieser Datei repräsentiert. Die Knoten des Graphen ergeben sich implizit aus der Liste, da jede Kante durch das Knotenpaar, bestehend aus Startknoten und Endknoten, gegeben ist, wobei Start- und Zielknoten durch einen Tabulator in der Datei getrennt sind.

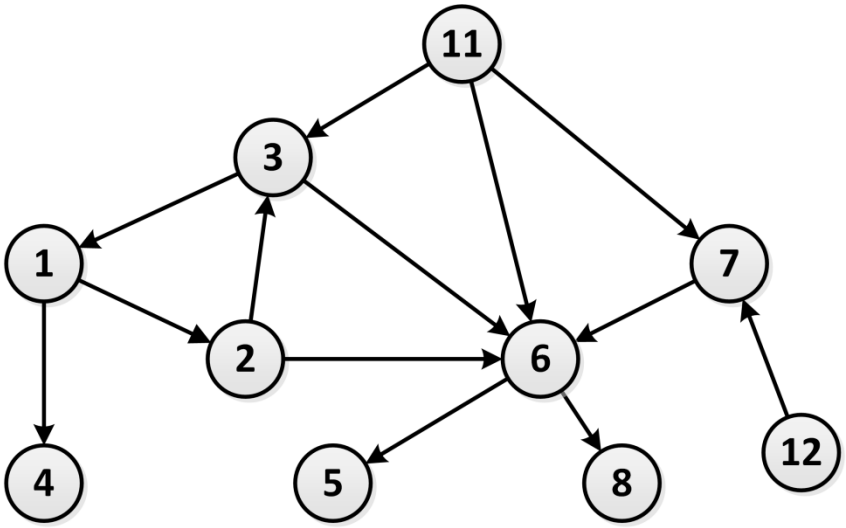

Abbildung 12: Einfacher gerichteter Graph

Als Beispiel ist für den Graphen aus Abbildung 12 die Kantenliste in Listing 1 angeführt. Die Liste besteht aus 13 Zeilen, was mit der Anzahl der Kanten in der Abbildung übereinstimmt. Die Identifikationsnummern (IDs) der zehn Knoten des Graphen aus Abbildung 12 sind nicht explizit als Knoten angegeben, kommen aber alle als Start- bzw. End-Knoten in der Kantenliste vor. Die Reihenfolge der Kanten bzw. Zeilen in der Datei ist nicht ausschlaggebend, aber die Reihung des Start- und Zielknoten muss richtig sein, falls die Kantenliste einen gerichteten Graphen korrekt repräsentieren soll. Wären die Positionen der zwei Knoten-IDs in jeder Zeile genau umgedreht, so würde diese Liste einen Graphen repräsentieren, bei dem jede gerichtete Kante genau in die umgekehrte Richtung zur Kante im ursprünglichen Graphen zeigt.

```
11    3
11    6
11    7
3     1
3     6
6     5
6     8
7     6
1     4
1     2
2     3
2     6
12    7
```

Listing 1: Darstellung eines Graphen im Kantenlisten-Format

Die ersten drei Zeilen aus Listing 1 beschreiben hier zum Beispiel die drei ausgehenden Kanten vom Knoten mit der ID 11 (vgl. Abbildung 12).

Zum Zweck der Einheitlichkeit besitzen Knoten grundsätzlich eine positive ganzzahlige ID und werden in den Kantenlisten und auch in den weiteren folgenden Datenformaten auch immer durch diese IDs identifiziert.

Da Dateien bei Graphen mit sehr vielen Kanten entsprechend groß werden, bietet das Framework auch die Möglichkeit Graphen in einem binären Format abzuspeichern und zu laden. Durch das Benutzen des binären Formats anstatt des ineffizienten Klartextformats kann beim Laden von Graphen sehr viel Zeit gewonnen werden und bei großen Graphen kann auch erheblich Speicherplatz gespart werden.

4.3 Hierarchisches Hintergrundwissen

Das vom Framework benutzte Hintergrundwissen ist als hierarchische Struktur gegeben. Da Hierarchien eine Spezialform von Graphen, nämlich Bäume, sind, kann dasselbe Datenformat verwendet werden, das schon in Abschnitt 4.2 besprochen wurde.

Auf Grund der Struktur einer Hierarchie sind die Anzahl der Kanten in der Kantenliste durch die Anzahl der Knoten im Graphen beschränkt. Jeder Knoten in der Hierarchie (ausgenommen die Wurzel) besitzt genau eine eingehende

Kante. Aus diesem Grund entstehen keine so großen Kantenlisten, wie es bei normalen Graphen möglich ist.

```
11    3
11    6
11    7
3     1
6     5
6     8
1     4
1     2
```

Listing 2: Kantenliste einer Beispiel-Hierarchie

Die Kantenliste einer möglichen Hierarchie die zum Graphen aus Abbildung 12 passt, ist in Listing 2 angeführt. Eine solche Hierarchie wird im einfachsten Fall zum Beispiel durch eine Breitensuche ausgehend von einem der Knoten (in diesem Fall 11) erstellt. Die visuelle Repräsentation dieser Hierarchie ist in Abbildung 13 zu sehen. Auffällig dabei ist, dass die Hierarchie im Vergleich zum Graphen aus Abbildung 12 einen Knoten weniger besitzt. Der Knoten mit der ID 12 fehlt, weil dieser nicht vom Knoten 11 aus erreichbar war. Alle anderen Knoten sind im Graphen und in der Hierarchie von der Wurzel aus erreichbar.

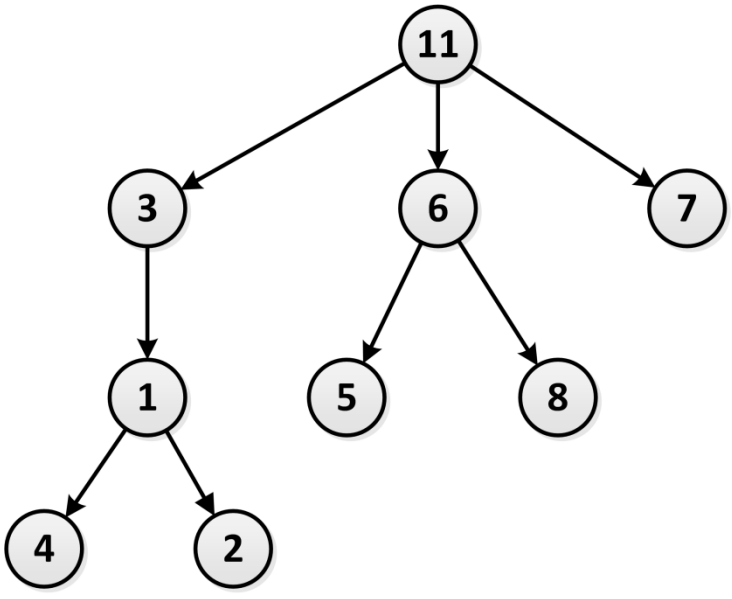

Abbildung 13: Visualisierung einer Beispiel-Hierarchie

4.4 Navigationspfad-Datenformat

Für die Speicherung von Navigationspfaden wurde ebenfalls ein Klartext-Format gewählt. Listing 3 zeigt ein Beispiel mit sechs Navigationspfaden, wobei jede Zeile einen Pfad darstellt. Die ersten beiden Zahlen sind die IDs der Start- und Zielknoten und durch einen Tabulator getrennt. Die ersten vier Zeilen aus Listing 3, sowie die Zeile fünf und sechs beschreiben also jeweils Navigationspfade mit demselben Start- und Zielknoten. Nach dem Start und Ziel folgt direkt eine Liste aus Knoten, die den Pfad beschreibt und wieder durch Tabulatoren getrennt ist. Die erste ID in dieser Liste ist immer die ID des Startknotens, weil die Navigation immer bei diesem Knoten begonnen wird. Anhand der letzten ID kann man zwischen Pfaden, die das Ziel erreicht haben (erfolgreiche Pfade) und Pfaden, die das Ziel nicht erreicht haben, unterscheiden. Stimmt die letzte Knoten-ID mit dem zweiten überein, so wurde der Zielknoten von diesem Navigationspfad erreicht.

```
11    6     11    7
11    6     11    3     1     2     6
11    6     11    7     6
11    6     11    3     1
1     5     1     2     6     5
1     5     1     2     3
```

Listing 3: Beispiel für das Format von Klick-Pfad-Daten

Von den in Listing 3 angeführten Pfaden waren der zweite, dritte und fünfte erfolgreich. Die restlichen Pfade haben den in der jeweiligen Zeile als Zielknoten angegebenen Knoten nicht erreicht und sind deshalb als erfolglos einzustufen.

Abbildung 14 zeigt den zweiten Pfad aus Listing 3. Der Startknoten 11 ist blau und der Zielknoten 6 grün markiert. Die im Navigationspfad enthaltenen Kanten sind rot gekennzeichnet.

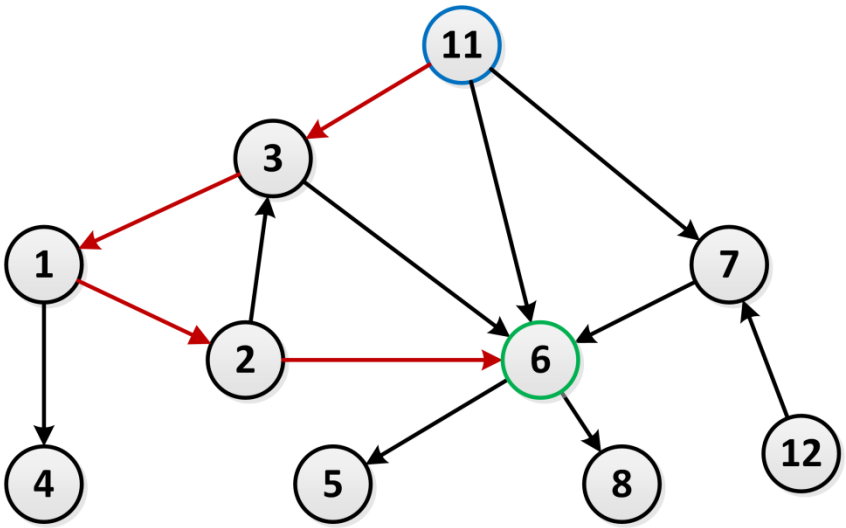

Abbildung 14: Möglicher Navigationspfad zwischen Knoten 11 und 6

4.5 Suchpaare

Die im vorigen Abschnitt besprochenen Navigationspfad-Daten enthalten Informationen über die Start- und Ziel-Knoten der Navigationspfade. Es ist wünschenswert die Start- und Ziel-Knoten-Paare, auch Suchpaare genannt, von den Navigationspfaden getrennt zur Verfügung zu haben, um, basierend auf diesen Informationen, dann neue Pfade durch Simulation zu generieren.

Wenn man zum Beispiel vom Framework Suchpaare generieren lässt, bringt ein Abspeichern der generierten Paare den Vorteil, dass bei Experimenten mit mehreren Simulationen hintereinander die Paare einmal generiert werden und diese dann für alle Experimente verwendet werden können, um die Ergebnisse der verschiedenen Simulationsdurchläufe besser vergleichen zu können.

Aus diesem Grund gibt es ein Daten-Format, das nur Suchpaare unabhängig von Navigationspfaden enthält. Neben der ID des Startknoten und der ID des Zielknotens sind optional auch die jeweiligen Eingangsgrade angegeben.

In Listing 4 ist ein Beispiel mit fünf Suchpaaren angeführt. Jede Zeile repräsentiert dabei immer ein Paar, beginnend mit den Start- und Ziel-Knoten. Die letzen beiden Zahlen jeder Zeile sind der Eingangsgrad des Start-Knotens und der Eingangsgrad des Ziel-Knotens. Die vier Werte jeder Zeile sind jeweils durch Tabulatoren getrennt.

```
11    6     0     4
1     5     1     1
3     8     1     1
7     5     2     1
2     4     1     1
```

Listing 4: Fünf beispielhafte Suchpaare

Die erste Zeile aus Listing 4 beschreibt zum Beispiel das Paar mit dem Startknoten 11 und dem Zielknoten 6, wobei 11 keine eingehenden Kanten besitzt und 6 einen Eingangsgrad von vier hat.

4.6 Kürzeste Distanzen

Die kürzesten Distanzen werden zu Vergleichs- bzw. Auswertungszwecken vom Framework benötigt. Sie beschreiben die Länge der kürzesten Pfade im Navigationsnetzwerk für alle Suchpaare.

Da es sehr rechenaufwändig ist, die kürzesten Distanzen zu erzeugen, ist es vorteilhaft ein Dateiformat zur Verfügung zu haben, in dem die berechneten Distanzen abgespeichert und später wieder geladen werden können.

In Listing 5 ist ein Auszug aus einer Datei zu sehen, die kürzeste Distanzen im entsprechenden Format beinhaltet. Neben den kürzesten Distanzen für die fünf Suchpaare aus Listing 4 sind noch acht weitere Distanzen angeführt.

Die Datei ist so in Abschnitte eingeteilt, dass jeweils die Distanzen für Suchpaare mit dem gleichen Startknoten zusammengefasst werden. Jede Zeile besteht aus zwei Zahlen bzw. IDs. Der Beginn eines Abschnittes ist immer durch eine spezielle Zeile gekennzeichnet. Diese Zeile beginnt immer mit dem Wert -1 und enthält durch einen Tabulator getrennt als zweiten Wert die ID des Startknotens.

So beginnt zum Beispiel der Abschnitt, der alle kürzesten Distanzen ausgehend vom Knoten mit der ID 1 umfasst, mit einer Zeile mit den Werten -1 und 1, wie auch in der ersten Zeile von Listing 5 zu sehen ist.

```
-1    1
2     1
6     2
5     3
-1    2
8     2
4     3
-1    3
4     2
5     2
8     2
-1    7
6     1
5     2
-1    11
3     1
6     1
7     1
```

Listing 5: Datei-Format für kürzeste Distanzen

Nach der Zeile, die einen Abschnitt beginnt, folgen jene Zeilen, die die Werte für die kürzesten Distanzen zu verschiedenen Zielknoten beinhalten. Diese Zeilen bestehen wiederum aus zwei durch einen Tabulator getrennten Werten. Der erste Wert ist dabei die ID des Zielknoten und der zweite Wert entspricht der errechneten kürzesten Distanz.

Die zweite Zeile in Listing 5 enthält beispielsweise die Werte 2 und 1 und befindet sich in dem Abschnitt, der die kürzesten Distanzen ausgehend vom Knoten mit der ID 1 beschreibt (vgl. Listing 5, Zeile 1). Aus diesen Informationen ergibt sich also, dass für das Start-Ziel-Knotenpaar mit den IDs 1 und 2 die kürzeste Distanz im Graphen eins beträgt.

Abschnitte enden mit der letzten Zeile, bevor ein neuer Abschnitt beginnt bzw. mit dem Dateiende. Der erste Abschnitt in Listing 5 beinhaltet also die ersten vier Zeilen, da in Zeile fünf bereits der nächste beginnt. Die kürzesten Distanzen, die der erste Abschnitt beschreibt, sind die Distanzen zwischen Knoten 1 und 2, 1 und 6 sowie 1 und 5.

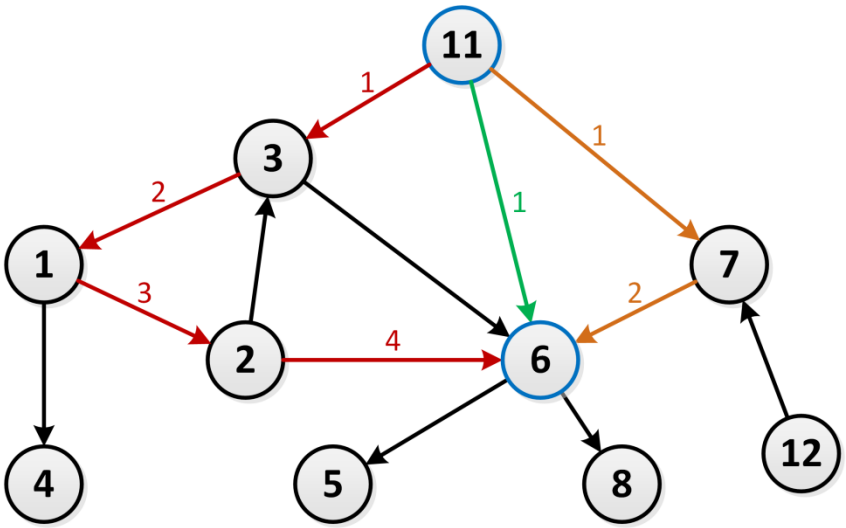

Abbildung 15: Kürzeste Distanz zwischen zwei Knoten

Abbildung 15 zeigt in grün den Pfad mit der kürzesten Distanz für das erste Suchpaar aus Listing 4. Es gibt zwischen den blau markierten Start- und Ziel-Knoten zwar weitere Pfade, wie zum Beispiel den rot markierten mit der Länge vier oder dem Pfad der Länge zwei in orange, aber, wie auch im letzten Abschnitt in Listing 5 ersichtlich, beträgt die kürzeste Distanz hier eins.

5. Das Framework

Dieses Kapitel befasst sich mit der Bibliothek SNAP und beschreibt die Struktur und Implementierung des Navigations-Frameworks. Vor allem im Programmcode wird das Framework oft als „MUN-Framework“ bezeichnet. Die Abkürzung MUN steht für „Modeling User Navigation“ und fasst den Zweck, nämlich die Modellierung von Benutzer-Navigation, des in C++ geschriebenen Frameworks, zusammen.

5.1 SNAP

Dass der effiziente Umgang mit großen Graphen der Grund dafür ist, die Bibliothek SNAP als Grundlage für das Framework zu benutzen, wurde bereits in Abschnitt 2.5 besprochen. Dieser Abschnitt geht etwas näher auf die Bibliothek und die Funktionalität, die von ihr geboten wird, ein.

Das Framework benutzt die aktuelle Version 1.11 (Stand Januar 2013) von SNAP, die auf der Downloadseite[12] zum Herunterladen angeboten wird.

5.1.1 Unterstützung

SNAP ist für mehrere Plattformen verfügbar. Mit Hilfe der Gnu Compiler Collection[13], kurz GCC, lässt sich SNAP unter Mac OS X, Linux und Windows mit Cygwin[14] kompilieren. Aber auch die Entwicklungsumgebung von Microsoft, Visual Studio, wird unter Windows unterstützt.

Zur grafischen Aufbereitung unterstützt SNAP Gnuplot[15], Graphviz[16], und NodeXL[17].

12 http://snap.stanford.edu/snap/download.html
13 http://gcc.gnu.org/
14 http://www.cygwin.com/
15 http://www.gnuplot.info/
16 http://www.graphviz.org/
17 http://nodexl.codeplex.com/

Gnuplot ist ein Werkzeug, das über die Kommandozeile und mit Scripts gesteuert wird und dazu dient, mathematische Funktionen und die verschiedensten anderweitigen Daten grafisch darzustellen.

Graphviz ist ein Softwarepaket, das die grafische Darstellung von Graphen und Strukturen ermöglicht.

NodeXL kann dazu genutzt werden, um Netzwerkdaten zu analysieren und zu visualisieren. Dabei wird Microsoft Excel als Plattform für die Repräsentation der Daten genutzt. Auch Nutzer ohne Programmierkenntnisse können so zum Beispiel eine Kantenliste in Microsoft Excel bereitstellen und erhalten die visuelle Aufbereitung des Graphen.

5.1.2 Struktur

SNAP ist, wie bereits erwähnt, in C++ geschrieben. Dieser Abschnitt gibt einen Überblick über die vorhandene Funktionalität.

Die SNAP Bibliothek besteht aus mehreren Teilen. Die Aufteilung erfolgt durch die Strukturierung in mehrere Ordner wie folgt:

- *snap-core*
  Dieser Ordner enthält die Kernfunktionalität von SNAP.

- *snap-adv*
  Die hier enthaltenen Komponenten gehören nicht zum Kern der Bibliothek, werden aber teilweise von einem der Beispiel-Programme benötigt.

- *snap-exp*
  In diesem Ordner befinden sich all die Komponenten, die sich noch in der Entwicklung befinden und nicht reif dafür sind, in den Kern von SNAP aufgenommen zu werden.

- *examples*
  Die Beispiele sind kleine Programme, die die Funktionalität von SNAP nutzen und demonstrieren.

- *tutorials*
  Ähnlich den Beispielen sind auch die Tutorials kleine Programme, die die Funktionalität von SNAP demonstrieren. Im Gegensatz zu den Beispielprogrammen sind die Tutorials jedoch darauf ausgerichtet, speziell den Umgang mit jeweils einer der von SNAP zur Verfügung gestellten Klassen und deren Methoden zu demonstrieren.

- *glib-core*
  Die hier enthaltenen Dateien stellen Implementierungen von Standard-Datentypen und Datenstrukturen wie Vektoren oder HashMaps zur Verfügung.

- *test*
  Dieser Ordner beinhaltet alle Unit-Tests, die für verschiedene Klassen vorhanden sind.

- *doxygen*
  Die Dokumentation der SNAP Bibliothek wird mit Hilfe von Doxygen[18] direkt aus den im Quellcode enthaltenen Kommentaren generiert und ist in diesem Ordner zu finden.

Ein kleiner Überblick über die Anwendungsmöglichkeiten von SNAP ist durch die Beispiel-Applikationen im „examples“-Ordner gegeben. Die Beispiele umfassen dabei etwa das Generieren von Graphen, das Errechnen von verschiedensten Knoten-Zentralitäts-Metriken bis zur Berechnung der strukturellen Eigenschaften von Graphen. [37]

Der Kern der Bibliothek befindet sich im Ordner „snap-core“. Die folgende Tabelle gibt einen auszugsweisen Überblick darüber, welche Funktionalitäten dabei an welcher Stelle implementiert sind.

[18] http://www.stack.nl/~dimitri/doxygen/

Tabelle 1: Funktionalität des SNAP-Kerns [37]

Quelldatei	Inhalt
alg.h	Algorithmen zur Graph-Manipulation und Analyse
bfsdfs.h	Algorithmen, die auf Breiten- oder Tiefensuche basieren
centr.h	Knoten-Zentralitäts-Maße
cncom.h	Konnektivität (stark / schwach zusammenhängende Komponenten,…)
gbase.h	Definiert Methoden zum Testen von Graph-Eigenschaften (gerichtet, Multigraph,…)
ggen.h	Verschiedene Generatoren zum Erzeugen von Graphen
ghash.h	Implementierung einer HashTabelle mit gerichteten Graphen als Schlüssel
gio.h	Implementierung der Speicher- und Lade-Funktionen für Graphen
graph.h	Die drei Graph-Klassen TUNGraph, TNGraph und TNEGraph
gstat.h	Struktur-Eigenschaften von Graphen
gviz.h	Die Schnittstelle zum Graphviz Software-Paket
network.h	Implementierung von verschiedenen Netzwerk-Typen
Snap.h	Die Haupt-Header-Datei von SNAP
subgraph.h	Methoden zum Extrahieren von Subgraphen sowie zur Konversion zwischen verschiedenen Graph-Typen
triad.h	Methoden zum Finden und Zählen von Knoten-Tripeln, die untereinander verbunden sind
util.h	Einfache Hilfsfunktionen zum Beispiel für die Manipulation von Zeichenketten

Aus all den Funktionalitäten, die von der Bibliothek zur Verfügung gestellt werden, wird von der Implementierung des Frameworks nur ein Teil genutzt, weil viele Dinge zu speziell und für das Framework nicht notwendig sind.

Zu den wichtigsten Bestandteilen der Bibliothek gehören die folgenden drei Haupt-Graph-Typen, die von SNAP unterstützt werden:

- Ungerichteter Graph – Wie im Kapitel 2 besprochen, besteht dieser Graph aus einer ungeordneten Liste aus Knotenpaaren, wobei zwischen jedem Paar höchstens eine Kante vorhanden sein kann.

- Gerichteter Graph – Diese Graphen-Art besitzt gerichtete Kanten.
- Gerichteter Multigraph – Zwischen zwei Knoten kann eine beliebige Anzahl an gerichteten Kanten vorhanden sein.

Diese drei Graph-Typen sind in der SNAP Bibliothek in den Klassen *TUNGraph* (ungerichtet), *TNGraph* (gerichtet) und *TNEGraph* (Multigraph) in der Datei *graph.h* (vgl. Tabelle 1) implementiert.

Vom Framework werden aktuell die Implementierungen des ungerichteten und gerichteten Graphen genutzt.

Zum Umgang mit diesen Typen sind im Namensraum „TSnap“ einige Funktionen definiert. Darunter befindet sich auch die Unterstützung von mehreren Datenformaten, in denen Graphen gespeichert und geladen werden können. Neben den in Abschnitt 4.2 besprochenen Kantenlisten im Klartext und dem Binärformat wird zum Beispiel auch ein XML-Format unterstützt.

5.1.3 Datensätze

Auf der Projekt-Webseite ist auch eine sehr große Auswahl an Datensätzen vorhanden, die aufzeigen, mit welchen Datenmengen die Bibliothek umgehen kann. Unter den Datensätzen befinden sich unter anderem Daten aus sozialen Netzwerken, Online-Communities, Kommunikationsnetzwerken, Zitierungsnetzwerken, Peer-To-Peer Internet-Netzwerken, Wikipedia und Twitter[19]. [38]

Einige der vorhandenen Datensätze sind zum Beispiel ein kleiner Ausschnitt des Netzwerks von Facebook [20]und Google+[21]. Von der Community der Videoplattform Youtube[22] existiert ein Graph mit 1,1 Millionen Knoten und 2,9 Millionen Kanten. Zum Peer-To-Peer Netzwerk Gnutella sind mehrere Datensätze von über einem Monat verteilten Zeitpunkten verfügbar. Die Datensätze zu

[19] https://twitter.com/
[20] https://www.facebook.com/
[21] https://plus.google.com/
[22] https://www.youtube.com/

Wikipedia umfassen unter anderem eine Historie der Artikelbearbeitung mit 2,3 Millionen Nutzern, 3,5 Millionen Artikeln und 250 Millionen Bearbeitungen. [38]

Diese Datensätze untermauern, dass die SNAP Bibliothek als Grundlage für die Implementierung eines Navigations-Frameworks sehr gut geeignet ist.

5.2 Überblick des Ablaufs

Der grundlegende Ablauf einer Simulation, die mit dem Framework ausgeführt wird, ist in einige Schritte eingeteilt.

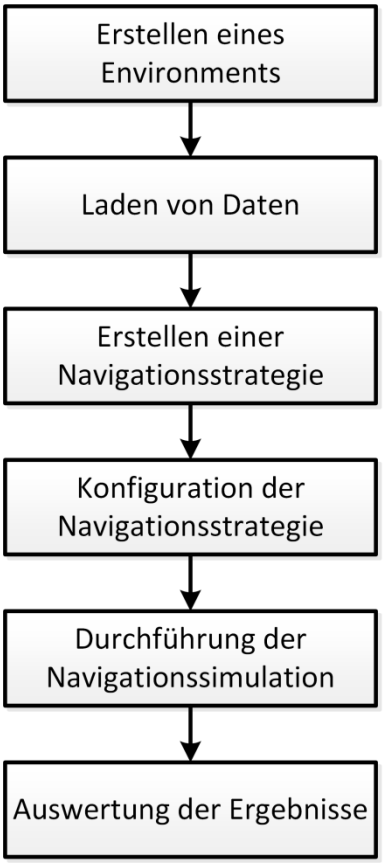

Abbildung 16: Allgemeiner Ablauf einer Navigationssimulation

Abbildung 16 zeigt, dass als erster Schritt ein „Environment“ erstellt wird. Dieses speichert und verwaltet die gesamte Konfiguration des Frameworks. Genaueres dazu wird in Abschnitt 5.4 erläutert.

Als nächstes wird bestimmt, welche Navigationsstrategie benutzt werden soll, und es wird mit dem Laden von erforderlichen Daten begonnen. Diese Daten

umfassen unter anderem den Graphen und die Hierarchie. Weiters werden die Suchpaare und kürzesten Distanzen, sofern angegeben, geladen. Ist keine Eingabedatei für die Suchpaare oder kürzesten Distanzen im Environment vorhanden, so werden die Suchpaare bzw. kürzeste Distanzen generiert.

Die Generierung der Suchpaare erfolgt dabei per Zufall, wobei aber einige Konfigurationsparameter zur Verfügung stehen, mit denen eingeschränkt gesteuert werden kann, wie die Suchpaare ausgewählt werden. Der Parameter „pairs.strat.u“ zum Beispiel bestimmt, wie viele Suchpaare einfach nach uniformer Verteilung zufällig aus allen Knoten des Graphen ausgewählt werden sollen.

Wenn alle Daten zur Verfügung stehen, von denen die jeweiligen Strategien abhängen, kann eine konkrete Navigationsstrategie erstellt werden. Dies geschieht in einer Fabrik, die in Abschnitt 5.5 näher besprochen wird. Außerdem wird die erstellte Strategie mit den notwendigen Daten versorgt und allgemein konfiguriert.

Somit ist alles für die Simulation vorbereitet und es kann mit dem Generieren von Navigationspfaden begonnen werden. Einen Überblick, wie dieser Vorgang abläuft, wird in Abschnitt 5.9 gegeben.

Ist die Simulation durchgelaufen, so sind alle Pfade generiert und die Aufgabe des Frameworks ist erfüllt. Es kann mit der Auswertung der generierten Pfade zum Beispiel durch grafische Aufbereitung der Ergebnisse begonnen werden.

5.3 Allgemeine Struktur

Das Framework ist strukturell in mehrere Teile aufgespalten. Die beiden Hauptteile sind der Framework-Kern und der Navigationsteil.

5.3.1 Framework-Kern

Der Kern des Frameworks ist im Ordner „core“ enthalten und umfasst Klassen, die Datenstrukturen implementieren sowie grundlegende Funktionalität bieten.

Die wichtigsten Klassen, die der Kern enthält, sind die folgenden:

- *Environment*
  Verwaltet alle Parameter, mit denen das Framework konfiguriert wird. Ein kleiner Auszug der gebotenen Funktionalität ist in Abbildung 17 ersichtlich.

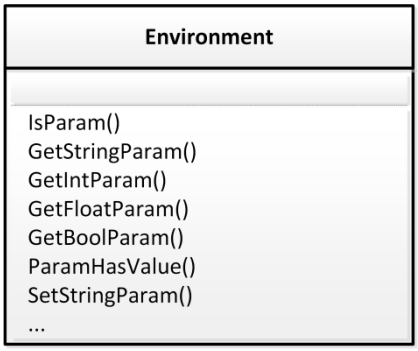

Abbildung 17: Die Environment Klasse

- *Graph*
  Definiert das Graph-Interface bzw. die abstrakte Klasse Graph. Abbildung 18 zeigt einige der zahlreichen Methoden, die diese Schnittstelle definiert.

- *GraphD*
  Repräsentiert einen gerichteten Graphen. Implementiert das von der Graph-Klasse definierte Interface. Ist gleichzeitig ein Adapter/Wrapper rund um die TNGraph bzw. PNGraph-Klasse der Bibliothek SNAP.

- *GraphU*
  Repräsentiert einen ungerichteten Graphen. Implementiert ebenfalls das von der Graph-Klasse definierte Interface. Ist ein Adapter zur TUNGraph bzw. PUNGraph-Klasse von SNAP.

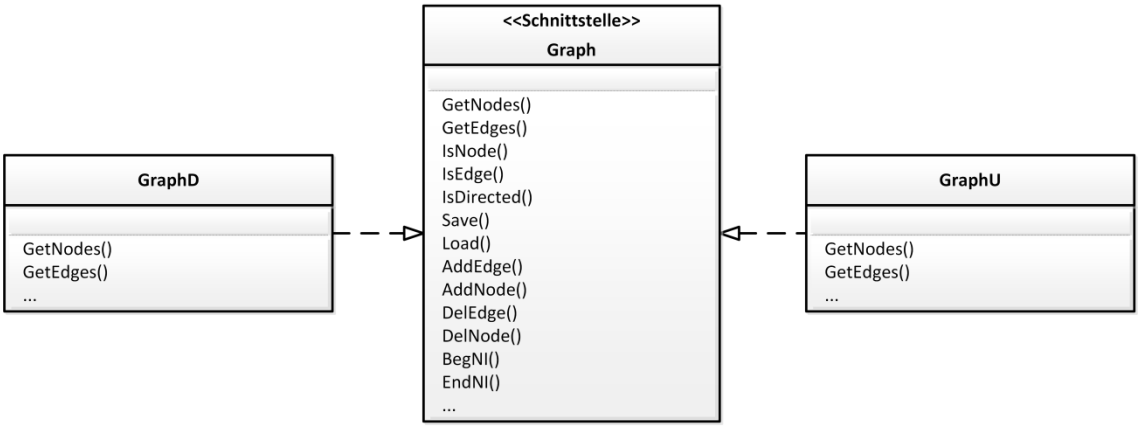

Abbildung 18: Klassen für gerichtete und ungerichtete Graphen

- *NodeI*

  Definiert das Interface für Knoten-Iteratoren. Durch diese Schnittstelle werden die Methoden festgelegt, die von den Knoten-Iteratoren unterstütz werden sollen. Einen kleinen Ausschnitt der definierten Funktionalität ist in Abbildung 19 zu sehen.

- *NodeID*

  Ist die Implementierung eines Knoten-Iterators für gerichtete Graphen.

- *NodeIU*

  Implementiert Knoten-Iteratoren für ungerichtete Graphen.

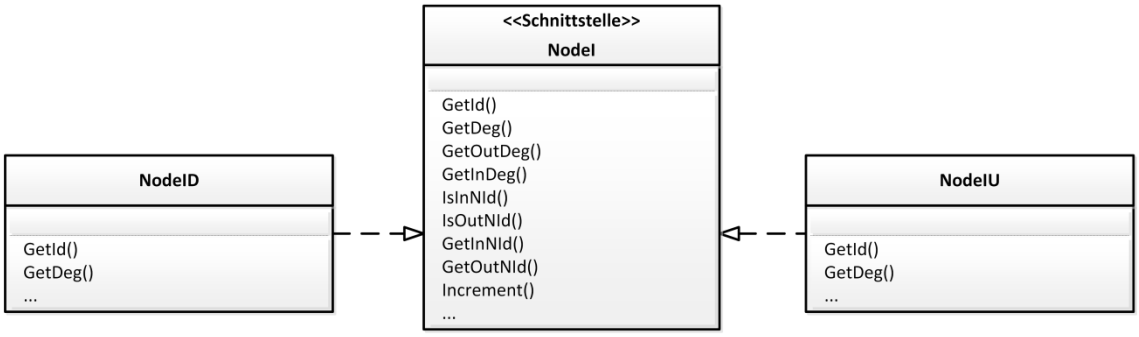

Abbildung 19: Gerichteter und ungerichteter Knoten-Iterator

- *Pairs*

  Dient zur Generierung und Verwaltung von Suchpaaren. Ein Teil der verfügbaren Methoden der Klasse sind in Abbildung 20 (links) dargestellt.

- *Path*
  Die Path-Klasse repräsentiert Navigationspfade und die damit in Verbindung stehenden Daten.

- *PathCollection*
  Wie auch Abbildung 20 zeigt hält eine „Pfad-Sammlung“ eine Menge von „Path“-Objekten und bietet Funktionen, wie das Laden und Speichern, die mit der gesamten Pfad-Menge arbeiten.

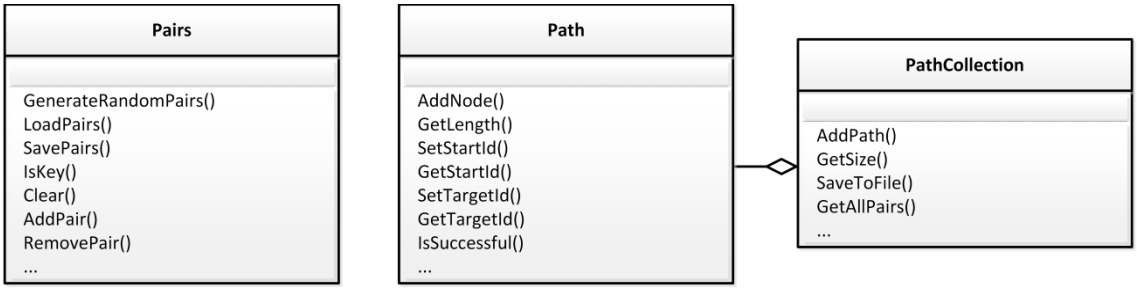

Abbildung 20: Pairs, Path und PathCollection Klasse

- *ShortestDistance*
  Repräsentiert eine Sammlung von kürzesten Distanzen und bietet Methoden wie zum Beispiel das Errechnen der kürzesten Distanzen für ein „Pairs“-Objekt.

Die Graph-Klasse definiert Funktionen um die Graph-Eigenschaften (wie die Anzahl der Knoten und Kanten) abzufragen. Außerdem werden Methoden angeboten, die es erlauben, festzustellen, ob ein Knoten mit einer bestimmten ID oder eine bestimmte Kante zwischen zwei Knoten im Graphen vorhanden ist. Weiters ist die Manipulation des Graphen durch Einfügen und Löschen von Knoten und Kanten möglich. Das Laden und Speichern des Graphen sowohl im Kantenlisten-Format als auch binär wird unterstützt. Zum leichteren Arbeiten mit dem Graphen kann auch ein Knoten-Iterator für den Knoten mit einer bestimmten ID sowie dem ersten und letzten Knoten angefordert werden.

Die beiden Klassen GraphD und GraphU sind Spezialisierungen der abstrakten Graph-Klasse.

Mit dem Knoten-Iterator, der in der Klasse NodeI definiert ist, kann sehr einfach über alle Knoten eines Graphen iteriert werden. Über die vom Iterator definierten Funktionen können die verschiedensten Eigenschaften der Knoten abgerufen werden. Ausgang-, Eingangs- und der Grad an sich können abgefragt werden. Die IDs der Nachbarknoten sowie Tests auf aus- und eingehende Kanten stehen zur Verfügung.

NodeID und NodeIU sind wiederum die konkreten Implementierungen der abstrakten Knoten-Iterator Klasse.

5.3.2 Navigation

Neben dem Framework-Kern ist der zweite Hauptteil des Frameworks durch den Navigationsteil gegeben. Dieser Teil befindet sich im Verzeichnis „navigation“ und enthält die Navigator-Klasse, die NavigationStrategyFactory sowie die vier Ordner „attrition“, „linkfilter“, „nodeselector“ und „strategy“ mit den darin enthaltenen Klassen.

Der Navigator stellt die Kern-Klasse des Navigations-Teils des Frameworks dar. Er hält das Environment-Objekt und somit die gesamte Konfiguration. Seine Aufgabe im Framework besteht darin, alles für die Navigationssimulation vorzubereiten. Zu diesem Zweck erstellt sich der Navigator während seiner Konstruktion auch eine Navigations-Strategie-Fabrik. Mit Hilfe der Konfiguration, die im Environment-Objekt gespeichert ist, und seiner NavigationStrategyFactory holt sich der Navigator eine Instanz einer Navigations-Strategie-Implementierung. Wird die Navigationssimulation später über einen Methodenaufruf gestartet, so hält der Navigator alle nötigen Daten bereit und kann mit dem Generieren von Navigationspfaden beginnen, indem die entsprechende Methode der Navigations-Strategie aufgerufen wird, die die Simulation durchführt und ein PathCollection-Objekt mit den generierten Pfaden zurückliefert.

Die NavigationStrategyFactory ist die zweite wichtige Klasse des Navigations-Teils des Frameworks. Sie ist, wie schon beschrieben, dafür zuständig eine Navigations-Strategie für den Navigator zu erstellen. Näheres dazu wird in Abschnitt 5.5 erläutert.

Der Ordner „attrition“ enthält die folgenden Klassen:

- *IAttrition*
  Interface, dessen Implementierung es ermöglicht das Abbrechen der Navigation aus verschiedenen Gründen (z.B. aus Frustration) in die Navigationssimulation mit einzubeziehen.

- *IAttritionSupport*
  Dieses Interface muss von jenen Navigations-Strategien implementiert werden, die mit Abbruchsimulationen, die das IAttrition-Interface implementieren umgehen können. Durch die Implementierung dieser Schnittstelle ist es dem Framework, im Speziellen der Navigations-Strategie-Fabrik, möglich eine Attrition-Implementierung an die Strategie zu übergeben. Dies geschieht durch die Benutzung der in Abbildung 21 ersichtlichen Methode „SetAttrition“.

Abbildung 21: Die Schnittstellen IAttrition und IAttritionSupport

Das Verzeichnis „linkfilter“ beinhaltet diese Klassen:

- *ILinkFilter*
  Mit Hilfe dieser Schnittstelle können die Knoten, über die der nächste Navigationsschritt erfolgen kann, gefiltert werden, bevor ein Knoten-Selektor den nächsten Knoten auswählt. Dies geschieht über die in Abbildung 22 sichtbare Methode „FilterNextNodeCandidates“.

- *ILinkFilterSupport*
  Analog zur Klasse IAttritionSupport muss dieses Interface von Navigations-Strategien implementiert werden, die einen Link-Filter benutzen möchten.

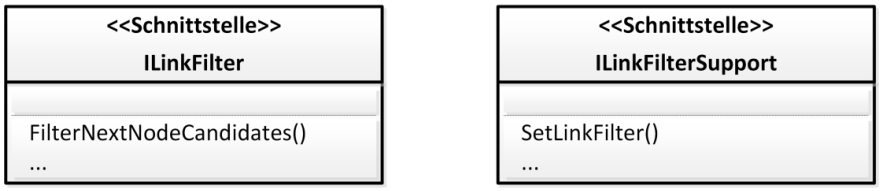

Abbildung 22: Die Schnittstellen ILinkFilter und ILinkFilterSupport

Im Ordner mit der Bezeichnung „nodeselector“ sind die folgenden Klassen zu finden:

- *INodeSelector*
  Die Hauptaufgabe von Knoten-Selektoren ist es aus einer Liste von möglichen nächsten Knoten nach einer bestimmten Strategie einen auszuwählen. Dieses Interface definiert diese Aufgabe.

- *NSBaseNodeSelector*
  Um die Implementierung von Knoten-Selektoren zu vereinfachen bietet diese abstrakte Klasse bereits einige Teile der Implementierung in einer Form wie sie von den meisten Selektoren benötigt wird.

- *NSRandom*
  Dies ist die Implementierung des NodeSelector-Interfaces, die Knoten auf der Basis des Zufalls als nächsten Navigationsschritt auswählt. Die Vererbungshierarchie zu dieser Klasse ist in Abbildung 23 dargestellt.

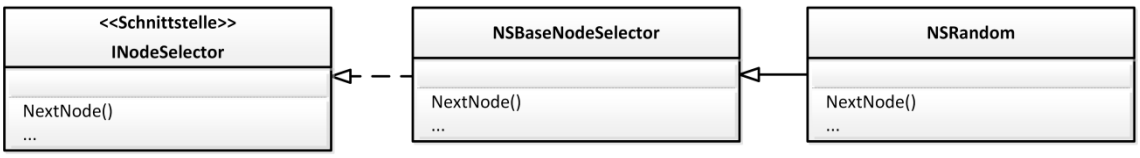

Abbildung 23: Die INodeSelector Schnittstelle und Implementierungen

Das letzte Verzeichnis „strategy“ umfasst diese Klassen:

- *INavigationStrategy*
  Diese ist das Interface, das eine Navigations-Strategie definiert. Die Navigations-Strategie ist der austauschbare Teil des Navigators, der das Verhalten des Navigators während der Simulation bestimmt. Die Navigations-Strategien werden in Abschnitt 5.5 näher besprochen.

- *SBaseStrategy*
  Ist eine abstrakte Klasse, die das NavigationStrategy-Interface teilweise implementiert und so gemeinsam nutzbare Code-Teile für speziellere Strategie-Implementierungen zur Verfügung stellt.

- *SCombiNavigator*
  Ist eine Navigations-Strategie-Implementierung, die mit mehreren Knoten-Selektoren umgehen kann. Welcher Selektor genutzt wird, wird auf Basis der Wahrscheinlichkeit und anderer Konfigurationsmöglichkeiten entschieden.

- *SRandomNavigator*
  Eine einfache Navigations-Strategie, die den Zufalls-Knoten-Selektor „NSRandom“ benutzt, um durch den Graphen zu navigieren. Die Vererbungshierarchie rund um die Navigations-Strategien ist in Abbildung 24 zu sehen.

Abbildung 24: INavigationsStrategy Schnittstelle und Implementierungen

Kompiliert wird das gesamte Framework unter Linux mit Hilfe der dafür erstellten Makefiles. Das Standard-Ziel, das make dabei kompiliert, ist das gesamte Framework, das aus dem Kern, dem Navigations-Teil und der SNAP Bibliothek, die in einem eigenen Ordner liegt, besteht.

Außerdem können noch über die make-Ziele „test“ und „runtest“ die Unittests, die das Google C++ Testframework[23] benutzen, kompiliert bzw. ausgeführt werden.

5.4 Konfiguration und Parameter

Wie in Abbildung 16 zu sehen ist, ist der erste Schritt, der bei der Ausführung des Frameworks geschieht, die Erstellung eines Environment-Objekts. Diese Umgebung beinhaltet und verwaltet alle Einstellungen und Parameter, die dem Framework beim Start zur Verfügung gestellt wurden.

Die Environment kann dabei auf drei Wegen mit Konfigurationswerten befüllt werden.

- Parameter werden aus einer Standard-Konfigurationsdatei eingelesen.
- Beim Aufruf des Frameworks kann eine Konfigurationsdatei über die Kommandozeile angegeben werden, aus dem Parameter gelesen werden.
- Parameter können direkt beim Programmaufruf auf der Kommandozeile angegeben werden.

Die Standard-Konfigurationsdatei wird dabei immer eingelesen. Die beiden anderen Möglichkeiten sind optional, es können aber auch alle drei Methoden kombiniert werden.

[23] http://code.google.com/p/googletest/

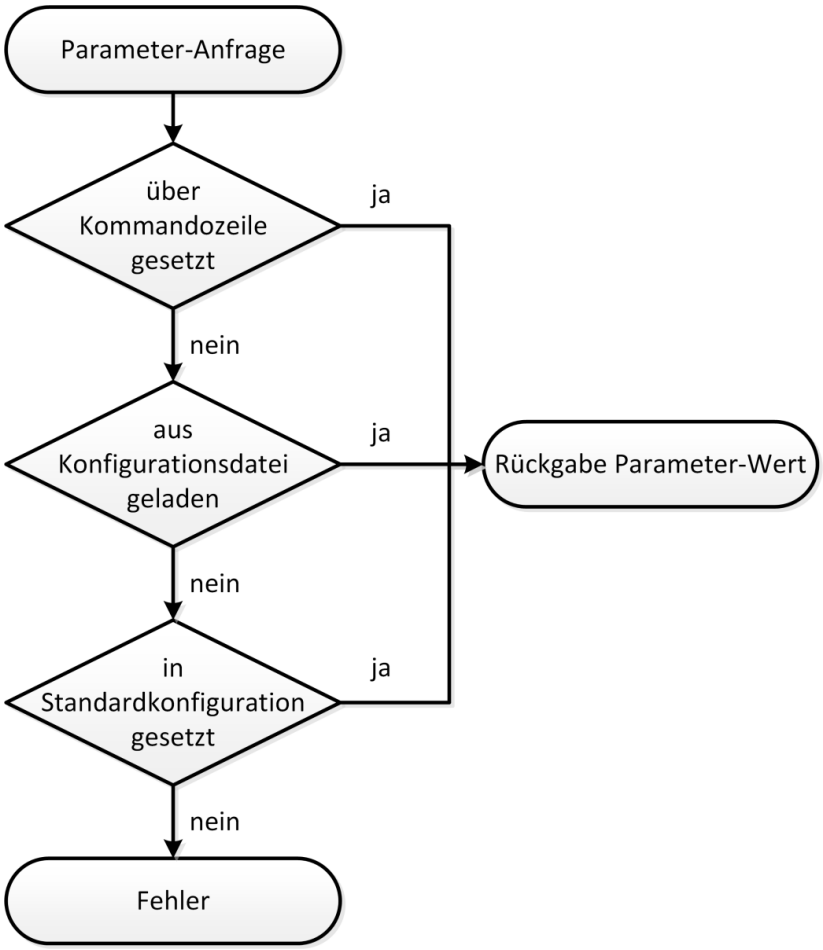

Abbildung 25: Ablauf beim Abrufen eines Parameter-Wertes

Parameter, die aus einer angegebenen Konfigurations-Datei stammen, überschreiben dabei Parameter mit demselben Namen, die aus der Standard-Konfiguration geladen wurden. Parameter-Werte, die direkt über die Kommandozeile gesetzt werden, überschreiben sowohl aus der Standard- als auch der spezifizierten Konfigurations-Datei gelesene Werte.

Wie das Holen eines Parameter-Wertes aus dem Environment abläuft, ist in Abbildung 25 zu sehen.

Grundsätzlich wird das gesamte Framework über die Konfiguration gesteuert. Der empfohlene Weg zur Nutzung des Frameworks ist der, dass die Standard-Konfigurationsdatei immer unverändert bleibt und tatsächlich die Standard-Werte enthält. Alle Parameter, die bei der Ausführung benötigt werden und von den vordefinierten Werten abweichen sollen, sollten in einer eigenen Konfiguration definiert werden und beim Aufrufen des Frameworks angegeben werden.

```
./munFramework --config:extraConfig.txt
-graph.file:g.txt -graph.type:d
```

Listing 6: Kommandozeilen-Aufruf des Frameworks

Ein Beispiel für die Nutzung der möglichen Konfigurationsformen ist in Listing 6 angegeben. Parameter aus der Datei „extraConfig.txt“ werden die gleichnamigen Werte aus der Standard-Konfiguration überschreiben. Außerdem wurden die beiden Parameter „graph.file“ und „graph.type“ direkt auf der Kommandozeile gesetzt und definieren somit diese Parameterwerte unabhängig davon, auf welche Werte diese Parameter zuvor schon in der Standard-Konfiguration oder in der Datei „extraConfig.txt“ gesetzt wurden.

Weiters ist zu beachten, dass der Parameter zum Angeben einer Konfigurationsdatei als einziger mit zwei Minuszeichen beginnt um ihn eindeutig von den anderen Parametern unterscheiden zu können.

Die wichtigsten Parameter, über die das Framework gesteuert werden kann, sind die folgenden:

- *global.amount*
  Über diesen Parameter wird die generelle Obergrenze des Navigators gesteuert. Dieser Wert begrenzt, für wie viele Suchpaare eine Navigationssimulation maximal ausgeführt wird (unabhängig davon wie viele Suchpaare tatsächlich geladen wurden). Der Wert 0 hebt diese Begrenzung auf und die Anzahl der Navigationssimulationen ist alleine dadurch bestimmt, wie viele Suchpaare vorhanden sind. Dieser Parameter ist vor allem beim schnellen Abschätzen der Ergebnisse von neu gewählten Strategieparametern praktisch, da dadurch eine kleine stichprobenartige Simulation auf großen Datensätzen ausgeführt werden kann.

- *global.output*
  Damit wird der Ausgabe-Ordner bestimmt, in dem alle während der Navigation erzeugten Dateien abgelegt werden.

- *graph.file*
  Dieser Parameter beinhaltet den Pfad und Dateinamen des Graphen, der bei der Simulation benutzt werden soll.

- *graph.type*
  Damit wird der Graph-Typ angegeben. Mögliche Werte sind „d“ (directed) für gerichtet und „u“ (undirected) für einen ungerichteten Graphen.

- *hier.file*
  Durch diesen Parameter werden Pfad und Dateiname der Hierarchie, die als Hintergrundwissen verwendet wird, angegeben.

- *hier.type*
  Analog zu „graph.type“ wird der Typ der Hierarchie festgesetzt.

- *pairs.inputfile*
  Mit diesem Wert kann wiederum ein Pfad inklusive Dateiname angegeben werden. Dieser bestimmt, aus welcher Datei Suchpaare geladen werden sollen. Wird dieser Parameter nicht angegeben, so wird durch andere „pairs.“-Parameter bestimmt, wie das Framework Suchpaare generieren soll.

- *shortestd.inputfile*
  Gleich wie beim Parameter „pairs.inputfile“ wird hier eine Datei zum Laden angegeben. Wird keine Eingabedatei mit bereits berechneten kürzesten Distanzen angegeben, so werden diese vom Framework berechnet.

- *navigation.strategy*
  Durch diesen Parameter wird bestimmt, welche Basisstrategie zur Navigation benutzt wird. Ein Spezialfall ist hier der Wert „combi“. Damit wird die Strategie ausgewählt, die es erlaubt, verschiedene grundlegende Strategien zu kombinieren. Es können für die „combi“-Strategie dann mehrere sogenannte NodeSelektoren angegeben werden die das Navigationsverhalten bestimmen.

- *navigation.maxhops*
  Dieser Wert legt fest, nach maximal wie vielen Schritten ein einzelner Navigationspfad abgebrochen wird. Wird das Ziel nicht gefunden oder bewegt sich der Navigator im Kreis, so wird nach dieser Anzahl an Schritten abgebrochen.

- *navigation.revisits*
  Gibt die maximale Anzahl an, wie oft ein bestimmter Knoten durch den Navigator wieder besucht werden kann. Ähnlich zum Parameter „navigation.maxhops“ ist dies eine Beschränkung der erzeugten Navigationspfade. Bewegt sich der Navigator so, dass für einen Pfad ein bestimmter Knoten immer wieder besucht wird, so ist die Anzahl dieser Besuche beschränkt und bei Überschreitung wird die aktuelle Navigation abgebrochen.

Diese Liste stellt nur einen Auszug der wichtigsten Parameter dar. Einige weitere Parameter, wie zum Beispiel die Konfiguration bzw. Angabe von Nodeselektoren, werden in anderen Abschnitten erklärt.

Parameter werden auf der Kommandozeile, wie zum Beispiel in Listing 6 zu sehen, gesetzt, indem der Parametername und Wert durch einen Doppelpunkt getrennt angegeben werden, wobei alle Parameternamen mit einem Minus beginnen. Ein Beispiel für das Format von Konfigurationsdateien ist in Listing 7 zu sehen.

```
# example config file
global.amount:1000
global.output:results/
graph.file:graph.txt
graph.type:d
hier.file:graph.hier
navigation.maxhops:50
navigation.revisits:10
```

Listing 7: Beispiel für eine Konfigurationsdatei des Frameworks

Jeder Parameter wird in einer eigenen Zeile angegeben, wobei Parametername und Wert wieder durch einen Doppelpunkt getrennt sind. Zeilen, die mit einer Raute beginnen, sind Kommentare.

5.5 Navigationsstrategien

Die in Abschnitt 5.3 aufgelistete Navigator-Klasse beinhaltet nicht direkt die Implementierung eines Navigationsalgorithmus. Stattdessen besitzt der Navigator, im Stile des Strategie-Entwurfsmusters, eine INavigationStrategy Member-Variable. Diese Variable speichert eine Referenz einer konkreten Navigations-Strategie.

Wird der Navigator gestartet indem seine Run()-Methode aufgerufen wird, so leitet der Navigator diesen Aufruf einfach an die von ihm gespeicherte Navigations-Strategie weiter, in der der tatsächliche Navigationsalgorithmus implementiert ist.

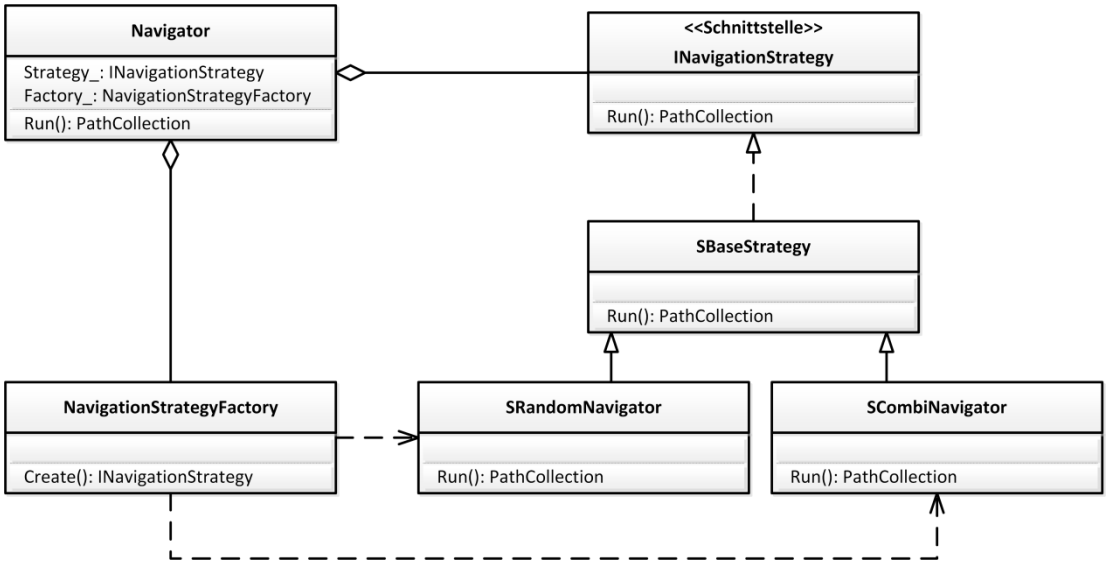

Abbildung 26: Eine Fabrik erzeugt die Navigationsstrategie des Navigators

Wie in Abbildung 26 zu sehen ist, besitzt der Navigator zusätzlich eine Navigations-Strategie-Fabrik. Diese wird schon ganz zu Beginn vom Navigator dazu benutzt, um sich entsprechend der Konfiguration eine Navigations-Strategie bauen zu lassen. Die Erstellung der Strategie über einen Methodenaufruf anstatt durch die direkte Benutzung eines Konstruktors entspricht dem Fabrik-Methoden-Entwurfsmuster. Der große Vorteil dieser Art der Konstruktion ist, dass der Navigator die konkreten Klassen, die das INavigationStrategy-Interface implementieren, nicht kennen muss.

Normalerweise bauen sich Objekte die Daten und weiteren Objekte, von denen sie abhängig sind bzw. die sie brauchen, selbst. Bei den Navigations-Strategien ist dies aber nicht der Fall. Die benötigten Daten werden von der NavigationStrategyFactory geladen bzw. die entsprechenden Objekte erstellt.

Die Abhängigkeiten werden dann im Stile des „Dependency Injection“-Musters in die Strategien „injiziert“.

Die Objekte, die nicht von den Navigations-Strategien erstellt, sondern von außen bereitgestellt werden, umfassen unter anderem den Graphen, die Hierarchie, Suchpaare, die kürzesten Distanzen, Link-Filter und Knoten-Selektoren.

Über die Navigationsstrategien bietet das Framework die Möglichkeit der Erweiterung. Eine neue Navigations-Strategie kann zum Framework hinzugefügt werden, indem entweder das INavigationStrategy-Interface direkt implementiert wird oder von der SBaseStrategy-Klasse abgeleitet wird. Die neue Navigationsstrategie muss lediglich mit einem Aufruf der Registrier-Methode in der Navigations-Strategie-Fabrik registriert werden und kann dann wie die bereits vorhandenen Strategien verwendet werden.

5.6 NodeSelector-Klasse

Die austauschbaren Navigations-Strategien sind sehr flexibel. Durch Implementierung der INavigationStrategy-Schnittstelle kann eine beliebig komplexe Navigations-Strategie erstellt werden.

Durch Ableiten der abstrakten SBaseStrategy-Klasse sind auch schon einige Teile der Implementierung vorhanden. Der grundlegende Navigations-Ablauf ist mit einer Schablonenmethode bereitgestellt, wobei aber einzelne Teile des Algorithmus durch konkrete Implementierungen ausgetauscht werden können.

Sind diese bereitgestellten Codeteile nicht für die spezielle Implementierung der Navigations-Strategie nutzbar, so kann, wie schon oben beschrieben, einfach die INavigationStrategy-Schnittstelle komplett implementiert werden ohne von SBaseStrategy abzuleiten.

Strategien wie der SRandomNavigator lassen sich aber durch Ableiten der Basis-Strategie recht einfach implementieren.

Ein zentrales Konzept der Navigations-Strategien ist es dabei, die konkrete Entscheidung, welcher Knoten als nächster Navigations-Schritt ausgewählt wird, in einen sogenannten NodeSelector auszulagern.

Die Navigations-Strategie ist also für den allgemeinen Ablauf der Navigation verantwortlich. Zum Beispiel werden Überprüfungen, ob das Ziel erreicht wurde, ob der aktuelle Pfad im Kreis verläuft, ob die maximale Anzahl der Navigationsschritte erreicht wurde, etc. von der Navigations-Strategie vorgenommen. Die konkrete Auswahl des nächsten Navigationsschrittes ist aber wiederum austauschbar nach dem Strategie-Entwurfsmuster in die NodeSelector-Implementierungen ausgelagert.

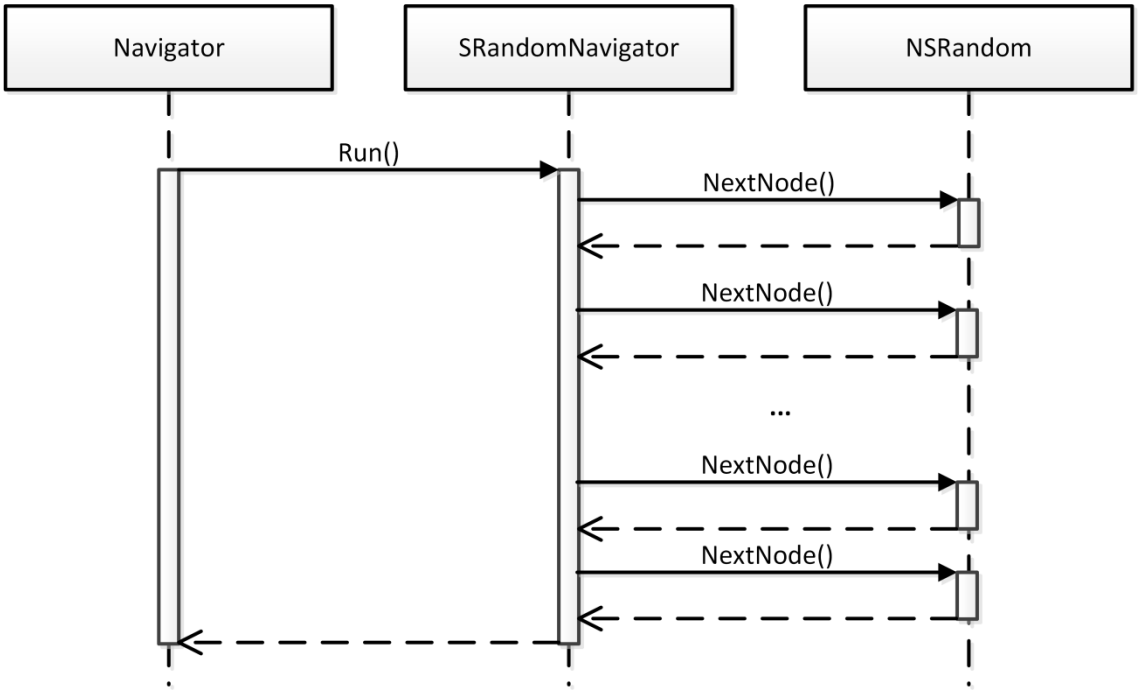

Abbildung 27: Der nächste Knoten wird durch NodeSelectoren ausgewählt

Dieser Umstand macht es auch möglich die NodeSelectoren in verschiedenen Navigations-Strategien wiederverwendbar zu machen. Abbildung 27 zeigt, dass der Navigator für den Navigationsvorgang eine Strategie nutzt, die wiederum die Knoten für die einzelnen Navigationsschritte von einer NodeSelector-Implementierung auswählen lässt.

So können natürlich beliebige Strategien implementiert werden, die die vorhandenen NodeSelector-Implementierungen ausnutzen und diese zum Beispiel kombinieren.

Um die Anzahl der Navigations-Strategien, die zum Zweck der Kombination von verschiedenen Knoten-Auswahl-Verfahren implementiert werden müssen, gering zu halten, bietet das Framework die SCombiNavigator-Strategie.

Mit dieser Strategie versucht das Framework etwas zur Verfügung zu stellen, das schon so flexibel ist, dass es in vielen Fällen nicht nötig ist neue Strategien zu implementieren. Es können mehrere vorhandene NodeSelectoren kombiniert werden, wobei die Steuerung, welche kombiniert werden, über die Konfiguration des Frameworks geschieht.

```
navigation.strategy:combi
combi.ns0:random
combi.ns1:somenodeselector
combi.ns1.aparam:10
combi.ns1.probability:20
combi.ns2:othernodesel
combi.ns2.useamount:10
combi.ns2.probability:40
```

Listing 8: Auszug aus einer Konfigurationsdatei

Der Wert für den Framework-Parameter „navigation.strategy“ muss zum Auswählen der SCombiNavigator-Strategie auf „combi“ gesetzt sein. Die einzelnen NodeSelectoren müssen in der Konfiguration, wie man in Listing 8 sieht, über den Parameter „combi.ns#“ definiert werden, wobei „#“ eine fortlaufende Ganzzahl ist. Das Beispiel definiert drei Node-Selektoren, „random“, „somenodeselector“ und „othernodesel“, wobei nur NSRandom tatsächlich implementiert ist. Die beiden anderen Selektoren dienen hier nur als Beispiele.

Parameter der einzelnen Selektoren werden über „combi.ns#.parametername“ festgelegt. Die vierte Zeile in Listing 8 legt zum Beispiel den Parameter „aparam“ von „somenodeselector“ auf den Wert 10 fest.

Der SCombiNavigator besitzt außerdem zwei Parameter, „combi.ns.probability“ und „combi.ns.useamount“, die die Standardwerte für die Wahrscheinlichkeit,

dass ein bestimmter Selektor eingesetzt wird, sowie die Anzahl, wie oft ein Selektor pro Suchpaar eingesetzt werden kann, festlegen.

Diese Parameter können, wie in Listing 8 zu sehen ist, für jeden Selektor einzeln überschrieben werden.

NodeSelector mit hohen Werten für „probability“ werden demnach von der Kombinations-Navigations-Strategie häufiger eingesetzt als solche mit niedrigen Werten. Über „useamount“ kann zum Beispiel festgelegt werden, dass ein bestimmtes Knoten-Auswahlverfahren nur ein einziges Mal pro Navigations-Pfad verwendet werden darf.

5.7 Interface zur Link-Filterung

Eine Schnittstelle, die es erlaubt die möglichen Knoten, die als nächster Navigationsschritt in Frage kommen, zu filtern, ist mit dem ILinkFilter-Interface festgesetzt.

Ein Filter wird ausgeführt bevor der nächsten Knoten durch einen NodeSelector ausgewählt wird. Er bekommt eine Liste der möglichen nächsten Knoten und die Information wie viele Navigationsschritte schon ausgeführt wurden. Durch diese Zusatzinformation kann das Filterverhalten zeitlich angepasst werden. Das Ziel ist, dass der Filter die Liste der Kandidatenknoten je nach Implementierung verändert und zum Beispiel Knoten mit hohen Ausgangsgraden oder anhand irgendwelcher anderen Kriterien aus der Liste der möglichen nächsten Knoten entfernt.

Diese Schnittstelle wird von Geigl in [3] dazu genutzt um einen speziellen Filter für das Framework zu implementieren um ein menschenähnliches Link-Auswahlverfahren zu unterstützen.

5.8 Attrition-Interface

Das Interface zur Simulation von Abbruchverhalten wird von Horwath in [4] dazu benutzt um Frustration, die während Navigationsvorgängen entstehen und durchaus zum Abbruch dieser führen kann, zu simulieren.

Ähnlich zur Link-Filter-Schnittstelle bekommen die Implementierungen dieses Interface die Information in welchem Navigationsschritt sich die Simulation gerade befindet. Darauf basierend wird dann eine Entscheidung getroffen, ob der Navigationsvorgang abgebrochen werden soll oder nicht.

5.9 Simulationsvorgang

Wie der Simulationsvorgang, der von den unterschiedlichen Simulations-Strategien implementiert ist, abläuft, ist in Abbildung 28 zusammengefasst. Das Diagramm in der Abbildung zeigt dabei im Groben die Schablonen-Methode zum Ablauf, wie er in der Klasse SBaseStrategy implementiert ist.

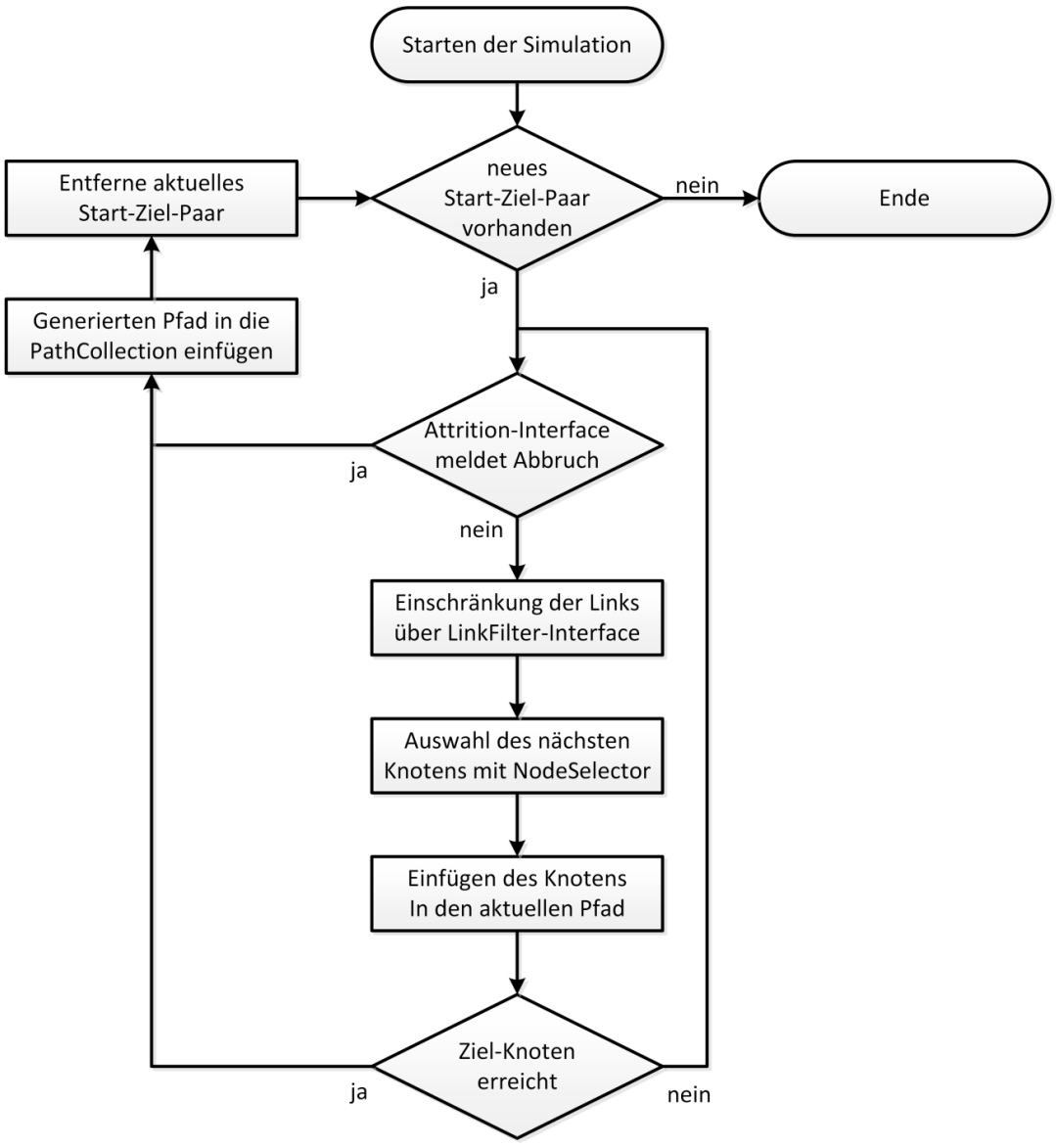

Abbildung 28: Ablauf einer Navigations-Strategie

Nach dem Starten der Simulation wird für jedes Suchpaar die folgende Abfolge von Vorgängen durchgeführt.

Als erstes wird für jeden Navigationsschritt mittels der Attrition-Schnittstelle überprüft, ob der Navigationsvorgang für das aktuelle Start-Ziel-Paar abgebrochen werden soll.

Ist dies nicht der Fall, so werden ausgehend vom aktuellen Knoten im Graphen, bei dem sich der Navigator gerade befindet, die möglichen nächsten Knoten eingeholt. Es wird also eine Liste mit jenen Knoten erstellt, die Ziele von allen ausgehenden Kanten des aktuellen Knotens sind.

Diese Liste wird über die Nutzung der LinkFilter-Schnittstelle je nach Implementierung des LinkFilters entsprechend eingeschränkt.

Die neue Liste mit möglichen Kandidaten für den nächsten Navigations-Schritt wird nun an einen NodeSelector weitergegeben, der daraus den Knoten auswählt, der das Ziel des nächsten Navigations-Schrittes ist.

Der ausgewählte Knoten wird in den aktuellen Navigations-Pfad eingefügt.

Die Überprüfung, ob der Zielknoten erreicht wurde, entscheidet, ob der aktuelle Navigationsvorgang abgeschlossen ist und der erzeugte Pfad in die Pfad-Sammlung eingefügt werden kann, oder ob mit dem nächsten Navigations-Schritt fortgefahren wird.

Aus Gründen der Übersichtlichkeit wurde neben anderen Details in der Abbildung auch die Überprüfung weggelassen, die sicherstellt, dass nach einer bestimmten maximalen Anzahl an Schritten die Navigation ebenfalls abgebrochen wird, um keine endlosen Pfade zu erzeugen.

5.10 Tests und Dokumentation

Wie in Abschnitt 5.3 bereits erwähnt, ist das Framework in mehrere Teile gegliedert. Neben dem Ordnern „core“, „navigation“ und „snap“ gibt es auch die beiden Verzeichnisse „test“ und „doxygen“.

Im Ordner „test“ befinden sich die Unit-Tests zum Framework. Diese können durch das Ausführen von make im Ordner „test“ kompiliert werden. Weiters können die Tests mit einem Aufruf von make im Hauptverzeichnis des Frameworks mit dem make-Target „test“ übersetzt werden.

Die Tests verwenden das Google C++ Test-Framework in der Version 1.6.0. Die Ausführung der Tests funktioniert im Hauptverzeichnis des MUN Frameworks wiederum über den make-Befehl. Mittels „make runtest“ werden alle Unit-Tests ausgeführt. Alternativ kann nach dem Kompilieren auch die ausführbare Datei „runAllTests“ aufgerufen werden.

Die Dokumentation des gesamten Frameworks wird mit Hilfe des freien Softwaredokumentationswerkzeugs Doxygen automatisch erstellt. Dazu werden alle Kommentare direkt aus dem Quellcode verwendet um eine Dokumentation im HTML-Format zu erstellen.

Die Dokumentation wird mit dem Aufruf „make docu“ im Hauptverzeichnis des Frameworks erzeugt und kann mittels „make cleandocu“ wieder entfernt werden.

Nach dem Erzeugen der Dokumentation liegt diese im Ordner „doxygen“ und kann durch das Öffnen der Datei „index.html“ im Browser angezeigt werden.

Die Dokumentation umfasst alle Klassen und Methoden, sowie Parameter, Rückgabewerte, Konstanten und Klassen-Member-Variablen.

6. Anwendung des Frameworks

Dieses Kapitel beschreibt einen konkreten Anwendungsfall, bei dem das Framework bereits eingesetzt wird. Geigl [3] und Horwath [4] benutzen das Framework dazu, um die Navigation von Benutzern durch das Wikipedia-Netzwerk zu modellieren.

6.1 Grundlagen zum Fallbeispiel

Um den Umfang und die Bedingungen für diesen Anwendungsfall des Frameworks zu verstehen, sind in diesem Abschnitt die Grundlagen des Informationssystems, in dem Benutzernavigationssimulationen durchgeführt werden, nämlich Wikipedia, sowie das Wikigame, der Ursprung der Benutzerdaten, mit denen die Ergebnisse der Experimente verglichen werden, erklärt.

6.1.1 Wikipedia

Wikipedia[24] ist ein Projekt, das darauf abzielt durch freie Inhalte, die jeder erstellen kann, ein webbasiertes Nachschlagewerk für jedermann bereitzustellen. Das Projekt ist unglaublich erfolgreich und Wikipedia ist das bekannteste und größte Online-Lexikon, das es gibt. Alle Einträge, die bei Wikipedia Artikel genannt werden, sind über Hyperlinks miteinander verknüpft. Das so entstehende Netzwerk aus Artikeln ist für jeden frei einsehbar und bearbeitbar.

Es gibt keine zusätzliche Hürde, wie zum Beispiel eine zwingende Benutzerregistrierung, um Wikipedia-Artikel bearbeiten zu können. Mit einem Browser und einer Internetverbindung kann jedermann etwas zur Wikipedia beisteuern. Außerdem wird für jeden Artikel eine komplette Historie der Bearbeitungen gespeichert. Auf diese Weise kann die inkrementelle Weiterentwicklung der Artikel beobachtet, optimiert und gegebenfalls rückgängig gemacht werden. Zur Formatierung der Einträge gibt es eine spezielle Syntax, die von den Artikel-Autoren benutzt werden muss. Der Vorteil dieser leicht erlernbaren Auszeichnungssprache liegt darin, dass man kein Programmierer oder Webentwickler

[24] http://www.wikipedia.org/

sein muss, um seinen Beitrag zum Wikipedia-Projekt leisten zu können. Neben den kollaborativ erstellten Artikeln werden auch multimediale Inhalte, wie zum Beispiel Audio-Aufnahmen, unterstützt und können in Wikipedia-Einträge eingebunden werden. Außerdem gibt es zu jedem Artikel eine Diskussionsseite, auf der sich Autoren austauschen und weitere Bearbeitungsschritte und Inhalte besprechen können. [39]

Gegründet wurde Wikipedia von Jimmy Wales gemeinsam mit Lawrence Sanger. Am 15. Jänner 2001 war der offizielle Start. Von diesem Datum an war die Wikipedia im Internet abrufbar. Bis März 2005 konnte der Meilenstein einer halben Million englischsprachiger Artikel erreicht werden. Ab diesem Zeitpunkt stieg die Wachstumsrate der Plattform so stark an, dass bereits 2006 und 2007 eine Million bzw. zwei Millionen Artikel verzeichnet werden konnten. [39]

Die englische Wikipedia umfasst zum Stand von November 2012 rund 4,2 Millionen Artikel und ist somit die größte Wikipedia. Von den mehr als 280 anderen Sprachen, in denen Wikipedia verfügbar ist, folgen der englischen Wikipedia geordnet nach der Größe die deutschsprachige mit rund 1,5 Millionen, die französische mit 1,3 Millionen und die niederländische mit 1,1 Millionen Artikeln. [40]

Dabei ist zu beachten, dass die sprachlich verschiedenen Versionen von Wikipedia nicht eine einfache Übersetzung der englischen Wikipedia sind, sondern von verschiedenen Autoren und Helfern bearbeitet werden und sich unabhängig entwickeln.

Auf Grund der Eigenschaft der ständigen Veränderung, der Offenheit von Wikipedia, der riesigen Masse an Daten und der Unüberprüfbarkeit der Herkunft aller Informationen ist Wikipedia als wissenschaftliche Quelle nicht geeignet. Doch auch wenn Wikipedia nicht zum Zitieren in wissenschaftlichen Arbeiten geeignet ist, so bieten die meisten Artikel trotzdem einen sehr guten Überblick über das besprochene Thema. In ihrer Gesamtheit bildet Wikipedia das menschliche Wissen zu sehr großen Teilen und in hoher Qualität ab. Aus diesem Grund gibt es auch das Bestreben Wikipedia noch besser zugänglich zu

machen und zum Beispiel eine Version in simplem Englisch[25] anzubieten, die auf die Verwendung von einfachen Wörtern und einfacher Grammatik setzt und so auch für Kinder geeignet ist.

Technologisch basiert Wikipedia auf der MediaWiki-Software[26]. Die in PHP[27] geschriebene Wiki-Software wurde ursprünglich speziell für Wikipedia entwickelt, kommt nun aber in vielen Projekten zum Einsatz. Das Konzept der Wiki-Technologie bestand schon vor Wikipedia und MediaWiki und umfasst unter anderem die Möglichkeit alle Inhalte durch wenige Klicks direkt bearbeiten zu können. [41]

Der Aufbau eines typischen Wikipedia-Artikels ist in Abbildung 29 ersichtlich. Das gesamte Netzwerk aus Einträgen ist sehr stark über Links (siehe Textstellen mit blauer Schrift in Abbildung 29) verknüpft. Über diese Links kann von Artikel zu Artikel navigiert werden.

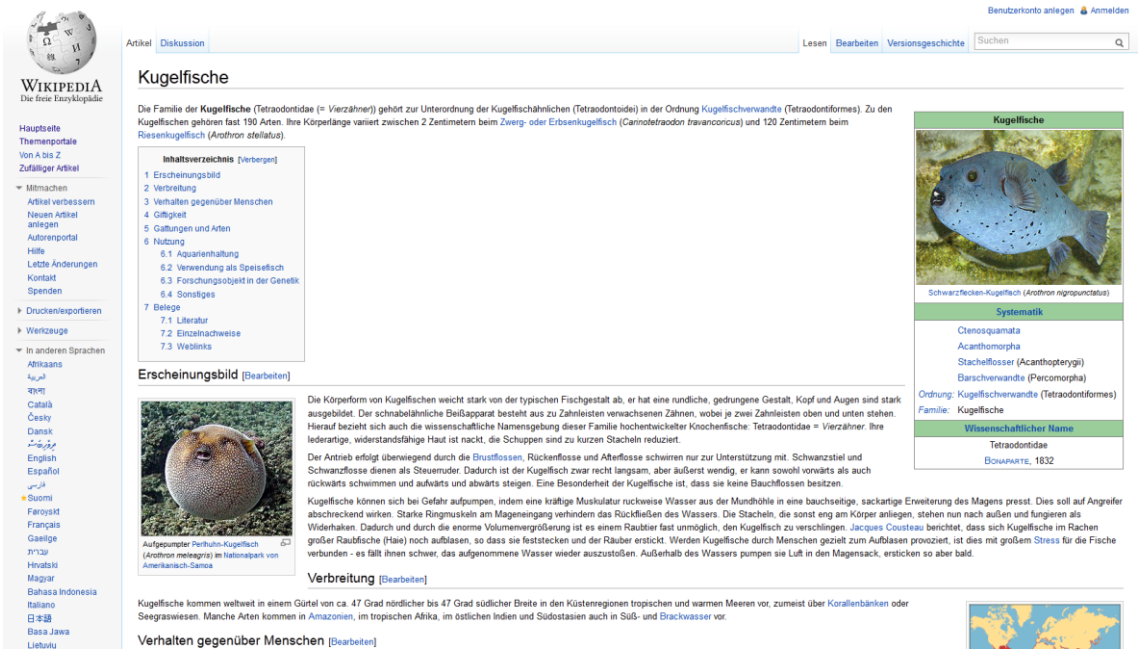

Abbildung 29: Ein typischer Wikipedia-Artikel am Beispiel "Kugelfische"

25 http://simple.wikipedia.org/

26 http://www.mediawiki.org/

27 Programmiersprache - http://php.net/

Grundsätzlich können gewünschte Inhalte über reine Navigation oder durch eine Suchfunktion erreicht werden.

Betrachtet man alle Seiten der Wikipedia als Knoten und Verlinkungen zwischen den Seiten als gerichtete Kanten, so entsteht ein riesiges Netzwerk. Dank der Grundphilosophie von Wikipedia ist die Seite nicht nur frei zugänglich, sondern es stehen auch alle Daten für Wissenschaftler und Interessierte als sogenannte „Dumps“[28] zum freien Download zur Verfügung.

Da alle Daten der Wikipedia bereitgestellt sind, ist das Wikipedia-Netzwerk ausgezeichnet zur Erforschung von Navigationsverhalten geeignet.

Die Struktur der Wikipedia-Daten bzw. welche Daten relevant für das Framework sind, wird in Abschnitt 6.2.1 genauer betrachtet, aber vorweg sei die Einteilung in die Namensräume, die sogenannten „Namespaces“ kurz angeführt. Alle Seiten des Wikipedia-Netzwerks sind einem Namensraum zugeteilt. Diese umfassen unter anderem:

- Main – alle Artikel und Weiterleitungen
- User – Seiten für den persönlichen Gebrauch von Benutzern
- Wikipedia – Informationen zum Wikipedia-Projekt (Regeln für Autoren,…)
- File – Beschreibungsseiten für multimediale Inhalte
- Template – Vorlagen (z. B. zum Erstellen neuer Seiten)
- Help – Hilfeseiten für Wikipedia-Anwender und Autoren
- Portal – Portalseiten für die verschiedenen Wissensgebiete

Der für die Untersuchung von Navigationsverhalten wichtigste Namensraum ist der „Namespace Main“ bzw. Hauptnamensraum. Mit einem Netzwerk, das nur aus Knoten aus dem Hauptnamensraum besteht, kann die Navigation durch die Benutzung von Links zwischen Artikeln betrachtet werden.

[28] http://dumps.wikimedia.org/

6.1.2 The Wiki Game

Rund um den Erfolg von Wikipedia entstanden unzählige Projekte, die das Konzept oder die Daten von Wikipedia benutzen, um die verschiedensten Dinge zu ermöglichen. Unter all diesen Projekten finden sich auch Spiele, die auf Wikipedia basieren.

Beispiele für Spiele, die auf Daten der Wikipedia basieren, sind die folgenden:

- Wikigame[29] – Quiz-Spiel
- Wikispeedia[30] – Navigations-Spiel
- The Wiki Game[31] – Navigations-Spiel

Das Quiz-Spiel Wikigame ist ein Frage-und-Antwort-Spiel, bei dem der Benutzer die richtige Antwort aus vier Möglichkeiten finden muss. Das Spiel ist online verfügbar und kann mit jedem internetfähigen Gerät und einem Browser gespielt werden. Alle Fragen und die dazugehörigen Antworten basieren auf Informationen, die in der Wikipedia aufgefunden werden können. Ähnlich zur Philosophie von Wikipedia können Benutzer selbst Fragen und die dazugehörigen Antworten erstellen und diese dann anderen Benutzern zugänglich machen.

Wikispeedia ist ein Spiel, das zu Forschungszwecken entwickelt wurde. Die Aufgabe der Spieler in dem webbasierten Spiel ist es, von einem Startartikel im Wikipedia-Netzwerk nur durch Verfolgen von Links auf den Artikelseiten zu einem Zielartikel zu navigieren. Die gesammelten Spieldaten werden zum Beispiel dazu genutzt, semantische Distanzen zwischen alltäglichen Konzepten abzuschätzen. [42]

The Wiki Game ist ebenfalls ein navigationsbasiertes Spiel. Das Grundprinzip besteht wieder darin, dass Spieler ausgehend von einem Startartikel nur durch

---

29 http://www.wikigame.org/

30 http://www.cs.mcgill.ca/~rwest/wikispeedia/

31 http://thewikigame.com/

Verfolgung von Links, die in den Artikeln, auf die Benutzer stoßen, vorkommen, zu einer Zielseite gelangen.

Auf Grund der ähnlichen Namen des Quiz-Spiels Wikigame und dem Navigationsspiel The Wiki Game sei an dieser Stelle festgehalten, dass an allen folgenden Stellen im Text dieser Arbeit mit den Bezeichnungen Wiki Game und Wikigame immer das Navigationsspiel The Wiki Game gemeint ist.

6.1.3 Allgemeines zum Spiel

Das Wikigame ist, wie bereits beschrieben ein Spiel, bei dem Nutzer durch das Wikipedia-Netzwerk navigieren um einen vorgegebenen Zielartikel zu finden. Der Artikel, bei dem eine Spielrunde startet und der Artikel, der von den Spielern erreicht werden muss, wird durch das Spiel vorgegeben und kann nicht, wie es zum Beispiel bei Wikispeedia möglich ist, selbst gewählt werden.

Das Spiel ist ein Projekt von Alex Clemesha, ist webbasiert und kann direkt im Browser gespielt werden. Die Technologien, auf denen das Spiel basiert, umfassen unter anderem Phyton[32], Django[33], Redis[34] und XMPP[35] in Verbindung mit Strophe[36].

Eine Registrierung ist nicht nötig. Beim Besuch der Webseite wird jedem Spieler, sofern kein Benutzer-Account registriert wurde, eine Gast-ID zugewiesen, mit der direkt an Spielrunden teilgenommen werden kann.

Der Vorteil eines Benutzerkontos liegt darin, dass die Seite dann Statistiken über alle Spiele sammelt und anzeigen kann, sowie eine Auflistung des Benutzers in Ranglisten möglich ist. Die tägliche und wöchentliche Rangliste für jeden Spieltyp wird aus den Ergebnissen der Spielrunden erstellt. Je nach Spieltyp erhalten die schnellsten Spieler, die das Ziel erreichen, Punkte. Für den ersten bis zum zehnten Platz werden dem Spieler zwischen 1000 und 100 Punkten für

---

[32] Programmiersprache - http://python.org/

[33] Phyton Web-Framework - http://djangoproject.com/

[34] In-Memory Datenbank - http://redis.io/

[35] Nachrichten-Protokoll - http://xmpp.org/

[36] XMPP Bibliothek - http://strophe.im/

die Ranglistenplatzierung angerechnet. Alle weiteren, die in einer Spielrunde das Ziel erreichen, erhalten 50 Punkte.

Grundsätzlich gilt für jede Spielrunde ein Zeitlimit. Nach dessen Ablauf beginnt eine neue Runde mit neu gewählten Start- und Ziel-Artikeln. Während einer Spielrunde gelten für jeden Spieler, der sich im Spiel befindet, derselbe Start- und Ziel-Artikel. Wie in Abbildung 30 ersichtlich, werden alle aktiven Spieler und deren aktuelle Anzahl an Navigationsschritten für diese Spielrunde am linken Bildschirmrand unter der Ziel-Artikel-Anzeige aufgelistet.

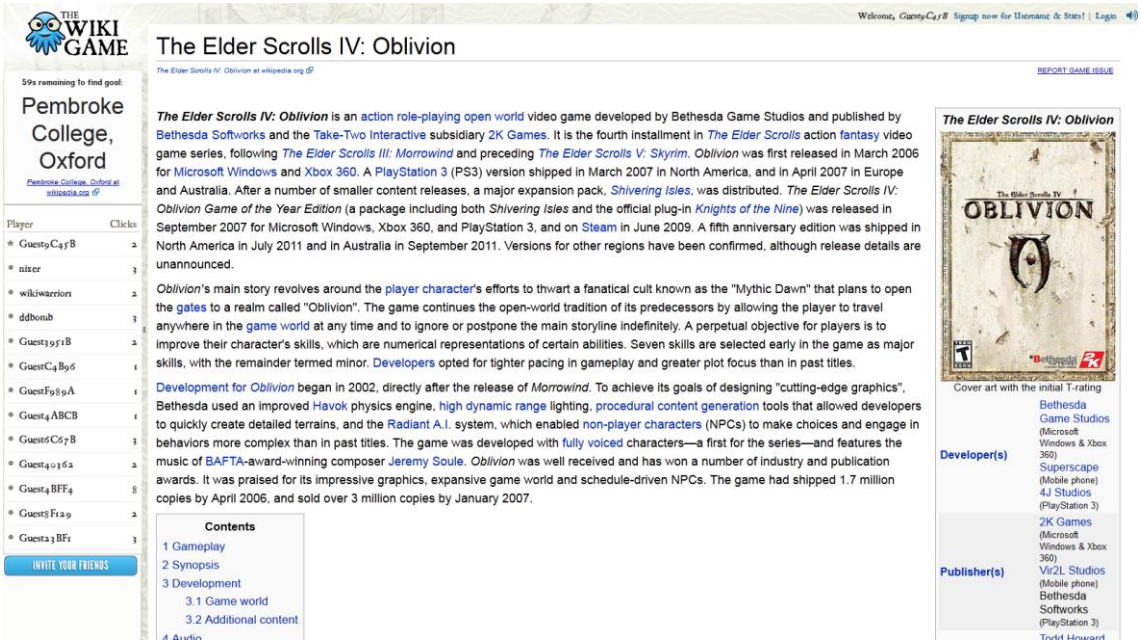

Abbildung 30: Die Wikigame-Benutzeroberfläche während einer Spielrunde

Für den Spieler ist eine Spielrunde zu Ende, wenn das Zeitlimit abgelaufen ist, das Ziel erreicht wurde oder wenn der Spieler aufgibt und die Seite verlässt.

Die für das Spiel eingesetzten Artikel sind leicht veränderte Wikipedia-Artikel. Einige Teile der Artikel werden ausgeblendet. Dies betrifft vor allem Navigationselemente wie Links zu Kategorien, etc., die von der Wikipedia zusätzlich zu den in den Texten enthaltenen Links bereitgestellt werden. Des Weiteren werden Links zur Diskussionsseite des Artikels sowie die Artikel-Historie ausgeblendet und die Möglichkeit den Artikel zu editieren unterbunden. Außerdem gibt es leichte Unterschiede in der Formatierung der Artikel gegenüber den

originalen Wikipedia-Einträgen. Außerdem wird natürlich keine Suche unterstützt, damit wirklich Navigationspfade von der Startseite ausgehend entstehen.

6.1.4 Wikigame-Spieltypen

Aufbauend auf den grundlegenden Regeln bietet das Wikigame fünf verschiedene Spielmodi an.

Der erste Spielmodus, der quasi auch der Hauptspieltyp ist, ist das „Speed Race“. Am Beginn jeder Runde bekommen alle Spieler gleichzeitig dieselben zufälligen Start- und Ziel-Artikel. Ausgehend vom Start-Eintrag gewinnt der Spieler, der über die Links in den Artikeln am schnellsten zum Ziel-Artikel navigiert. Die Anzahl der Klicks bzw. Anzahl der benutzten Links spielt bei diesem Spieltyp keine Rolle.

„Least Clicks“ – also „die wenigsten Klicks“ – ist der zweite Spieltyp, den das Wikigame anzubieten hat. Analog zum „Speed Race“ werden am Anfang einer Spielrunde zwei Artikel als Start und Ziel ausgewählt. Jedoch gewinnt nicht der Spieler, der das Ziel als erstes erreicht, sondern der Benutzer, der die wenigsten Klicks dafür benötigt hat. Der Spieler mit dem kürzesten Navigationspfad erspielt sich hier also die meisten Punkte für die tägliche und wöchentliche Rangliste.

Der Name des Modus „Six Degrees of Wikipedia“ ist an das in Abschnitt 3.3 besprochene Kleine-Welt-Phänomen angelehnt. Aufgabe der Spieler ist es, die ausgewählten Start- und Ziel-Einträge über einen Pfad zu verbinden, der genau sechs Klicks lang ist.

Der vierte Spieltyp nennt sich „Five Clicks to Jesus“, zu Deutsch „Fünf Klicks zu Jesus“. Wieder ist wie im vorhergehenden Modus die Anzahl der verfügbaren Klicks eingeschränkt. In diesem Fall muss das Ziel in genau fünf Schritten erreicht werden, wobei das Ziel immer der Wikipedia-Eintrag zu Jesus ist. Genau wie zuvor stellt die exakte Vorgabe, wie viele Klicks zum Ziel benötigt werden, eine besondere Anforderung an das Navigationsverhalten.

Die letzte Spielvariante trägt den Namen „No United States“. Dabei gilt die Regel, dass der Wikipedia-Artikel zu den Vereinigten Staaten von Amerika nicht in den Navigationspfaden vorkommen darf. Diese Einschränkung ist aus dem Grund interessant, da der Eintrag zu den USA eine sehr beliebte Seite ist und extrem häufig in den Navigationspfaden von Nutzern zum Beispiel in den anderen Spielmodi vorkommt. Der Grund dafür liegt darin, dass der Artikel zu den USA von besonders vielen Seiten verlinkt ist. Der Eingangsgrad dieses Eintrags ist also im Vergleich zu anderen Wikipedia-Artikeln extrem hoch.

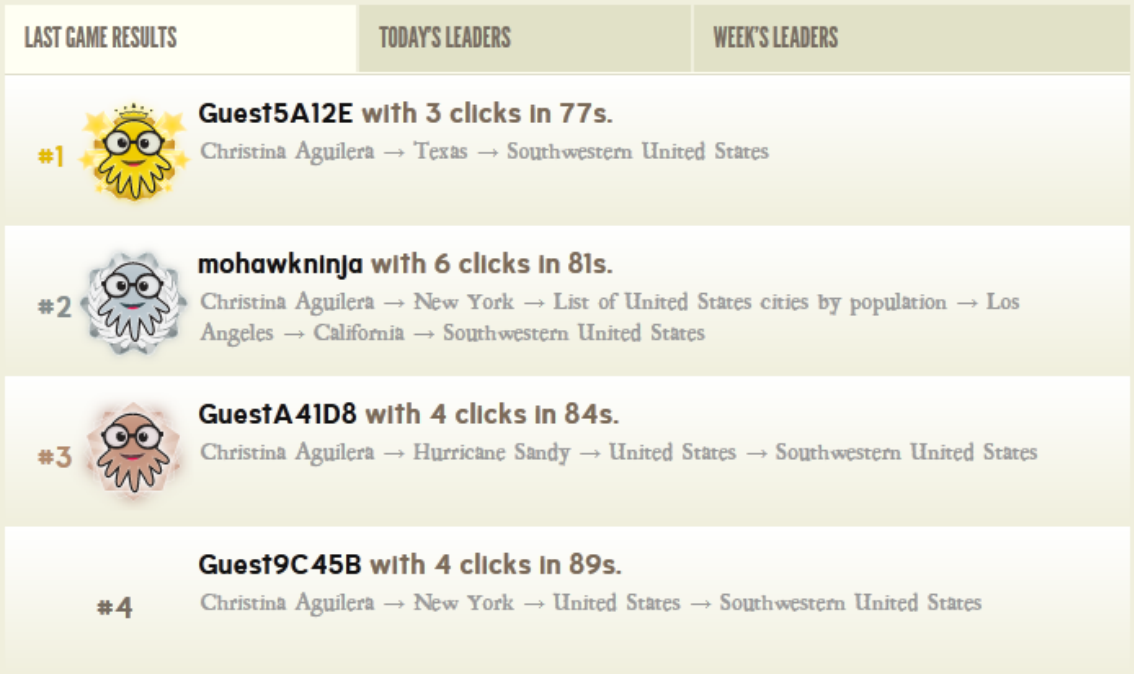

Abbildung 31: Ergebnis einer Wikigame-Spielrunde inkl. Navigationspfade

Neben den täglichen und wöchentlichen Ranglisten können für jeden Spieltypen auch die Ergebnisse der letzten Spielrunde abgerufen werden. Wie in Abbildung 31 ersichtlich, werden dort auch die erfolgreichen Navigationspfade, die von den entsprechenden Spielern erzeugt wurden, angezeigt.

6.2 Simulationseingangsdaten

Für den Einsatz des Frameworks waren alle Eingangsdaten, in den in Kapitel 4 beschriebenen Formaten, notwendig. Welche Daten als Grundlage der Simulationen dienten, ist im Abschnitt 6.1 beschrieben. Dieser Abschnitt bespricht, wie

die Daten aus der Wikipedia bzw. dem Wikigame als Eingangsdaten für Simulationen mit dem Framework benutzt werden können.

6.2.1 Graph

Der für die Navigation benutzte Graph war der Wikipedia-Graph. Der Graph wird gebildet, indem alle Artikel als Knoten und alle Hyperlinks zwischen Artikeln als gerichtete Kanten abgebildet werden.

Als Ausgangsdatensatz diente der Wikipedia-Datenbank-Dump vom 15. November 2011. Dieser Datensatz wurde deshalb ausgewählt, weil dies der älteste Datensatz war, der noch vollständig zum Download verfügbar war und er zeitlich am besten mit den anderen Daten (siehe folgende Abschnitte) zusammen passte. Die Datenbank-Dumps beinhalten alle Inhalte der Wikipedia als SQL[37]- und XML[38]-Dateien. Darunter befinden sich unter anderem alle Artikel mit der gesamten Chronologie der Bearbeitungen, Listen aller Artikeltitel, Listen aller Weiterleitungen und Seiten-zu-Seiten Linklisten.

Aus dieser Menge von Daten wurde von Horwath, wie in [4] genau beschrieben, der Wikipedia-Graph, kurz Wikigraph, erstellt. Da jeder Artikel eine eindeutige Identifikationsnummer, kurz ID, besitzt, werden zur Identifizierung von Knoten im Wikigraphen diese IDs benutzt. Alle Seiten, die als Knoten für den Graphen herangezogen wurden, stammen aus dem Wikipedia Hauptnamensraum „Main“. Horwath hat in den Graphen also keine Diskussionsseiten, Artikelvorlagen, Hilfeseiten oder ähnliches aufgenommen.

Was der Hauptnamensraum jedoch enthält, sind neben den Artikeln auch Weiterleitungen. Weiterleitungen sind Seiten, die nur dazu dienen auf einen anderen Wikipedia-Artikel zu verweisen. Um das Beispiel aus Abbildung 29 in Abschnitt 6.1.1 noch einmal aufzugreifen, ist der Wikipedia-Eintrag „Kugelfisch“ zum Beispiel eine Weiterleitung auf den Artikel „Kugelfische“.

---

[37] Structured Query Language - Datenbanksprache

[38] Auszeichnungssprache zur textuellen Darstellung von hierarchischen Daten

Weiterleitungen hat Horwath so behandelt, dass sie im Graphen aufgelöst wurden. Zum Beispiel wurden alle Links auf die Seite „Kugelfisch“ im Graph durch Kanten zum Artikel „Kugelfische“ ersetzt.

Der Wikigraph, der schlussendlich entstand, besitzt rund 232 Millionen Kanten und 3,8 Millionen Knoten, was mit den offiziellen Angaben der Artikelanzahl [40] im November 2011 übereinstimmt.

Die genaue Beschreibung, aus welchen Teilen des Wikipedia-Datensatzes und vor allem durch welche Methoden der Wikigraph extrahiert wurde, sowie alle weiteren, Schritte wie zum Beispiel das Auflösen der Weiterleitungen, sind in [4] beschrieben.

Die für das Framework aufbereitete Kantenliste (siehe Abschnitt 4.2) beansprucht ca. 3,6 Gigabyte Speicherplatz. Der in das binäre Format konvertierte Graph hat nur mehr 1,9 Gigabyte.

6.2.2 Hierarchie

Als Hintergrundwissen für die Navigation wird im Framework, wie schon erwähnt, eine Hierarchie eingesetzt. Die zum Wikigraph passenden Hierarchien wurden von Horwath erstellt. Die Vorgehensweise sowie die dazu verwendeten Algorithmen und Werkzeuge werden in [4] vorgestellt.

6.2.3 Navigationspfad-Daten

Die Navigationspfad-Daten, die als Vergleich für die vom Framework erzeugten Pfade benutzt werden, stammen vom in Abschnitt 6.1.2 besprochenen Wikigame.

Wie die vorhandenen Daten bearbeitet und gefiltert wurden, um auf das in Abschnitt 4.4 beschriebene Datenformat zu kommen, ist in [4] ausgeführt. Erwähnenswert dabei ist, dass alle Knoten-IDs denen des Wikigraphen entsprechen. Es liegen also fertige Navigationspfade über die Kanten des Wikigraphen vor.

Ähnlich zum Wikigraphen sind auch in den Navigationspfaden Weiterleitungen ein Sonderfall. Weiterleitungen sind in den Navigationspfaden grundsätzlich als nur ein Knoten enthalten. Klickt ein Nutzer also zum Beispiel auf einer Seite einen Link an, der auf die Wikipedia-Seite „Kugelfisch“ verweist, welche eine Weiterleitung auf „Kugelfische“ ist, so wird nur einer der beiden Knoten, „Kugelfisch“ oder „Kugelfische“, in den Navigationspfad aufgenommen.

Des Weiteren sind nur „echte“ Klicks in den Navigationspfaden enthalten. Das Benutzen der „Zurück“- und „Vorwärts“-Schaltfläche des Browsers durch den Nutzer wird nicht in den Navigationspfaden berücksichtigt.

6.2.4 Suchpaare

Für die Simulationen, die mit dem Framework ausgeführt wurden, machte es keinen Sinn, die Suchpaare vom Framework zufällig generieren zu lassen. Aus diesem Grund wurden die Suchpaare aus den Navigationspfad-Daten des Wikigame-Datensatzes extrahiert. Auf Basis dieser Suchpaare konnte das Framework dazu benutzt werden Navigationspfade zu erzeugen, die dieselben Start- und Ziel-Knoten besitzen, wie sie auch für die Spieler des Wikigames festgelegt waren.

6.3 Erweiterung des Frameworks

Im Zuge der Nutzung des Frameworks für die Arbeiten [3] und [4] wurde das Framework auch erweitert.

Geigl hat sich mit der Implementierung von NodeSelectoren, die er für seine Arbeit benötigte, als auch der LinkFilter Schnittstelle beschäftigt. Mit einer Implementierung des ILinkFilter-Interface, die Geigl Link-Restrictor nennt, versucht er das in Abschnitt 3.5.1 kurz angesprochene Zoom-In- und Zoom-Out-Verhalten von Nutzern nachzubilden.

Um dies gut zu bewerkstelligen, benutzt er eine große Zahl an Einstellungsmöglichkeiten, die über die Framework-Konfigurationsdatei gesteuert werden können und in seiner Arbeit in [3] im Detail erklärt sind.

Horwath hat das Interface IAttrition gleich mehrfach implementiert. In [4] stellt er eine Abbruchsimulation vor, die die Attrition-Schnittstelle für die Implementierung eines konstanten, linearen, quadratischen und exponentiellen Modells benutzt.

Die Modelle sind über einige Parameter konfigurierbar und sind, wie aus der Arbeit hervorgeht, sehr gut zur Nachbildung des Abbruchverhaltens von Nutzern geeignet.

6.4 Einsetzbarkeit des Frameworks

Wie aus diesem Fallbeispiel hervorgeht, kann das Framework mit großen Datenmengen wie es der Wikigraph darstellt gut umgehen. Die 3,8 Millionen Knoten und 232 Millionen Kanten stellten kein Problem für das Framework dar.

Außerdem hat sich auch gezeigt, dass die Erweiterbarkeit sinnvoll ist und die Schnittstellen gut verwendbar sind. Geigl und Horwath konnten einige Erweiterungen durchführen und das Framework ohne Einschränkungen für ihre Zwecke nutzen.

Des Weiteren ermöglichte die SCombiNavigator-Implementierung die einfache Kombination von verschiedenen Knoten-Auswahl-Verfahren gemeinsam mit dem Link-Restrictor von Geigl und den Navigations-Abbruch-Modellen von Horwath.

Im Großen und Ganzen kann also gesagt werden, dass das Framework durchaus gut für Einsatzgebiete, wie es etwa dieses Fallbeispiel aufzeigt, geeignet ist.

7. Zusammenfassung

Im Zuge dieser Arbeit ist ein Framework entstanden, mit dem Navigationspfade erstellte werden können, die auf dezentraler Suche mit hierarchischem Hintergrundwissen basieren.

An einigen Stellen konnten klassische Architektur- bzw. Software-Entwurfsmuster eingesetzt werden. An anderen Stellen mussten Lösungen ausgearbeitet werden, die speziell von der Thematik der Navigationssimulation abhängig waren.

Durch die Einschätzungen aus wissenschaftlichen Arbeiten, die sich mit dem Thema der Navigation im Allgemeinen und der Benutzernavigation im Speziellen befassten, konnten wichtige Hinweise für die Entwicklung des Frameworks abgeleitet werden. Vor allem ergab sich aber der Großteil der Anforderungen an das Framework aus der Analyse dieser Forschungsergebnisse.

Ein wichtiger Punkt war es jedenfalls, dass die Erweiterbarkeit gegeben ist, die es erlaubt, Versuche mit den verschiedensten Navigationsstrategien zu ermöglichen, da bisher keine universelle Lösung gefunden wurde die optimale Ergebnisse erzielt.

Durch das vorgestellte Fallbeispiel wurde zugleich die Nutzbarkeit des Frameworks überprüft. Der Umgang mit großen Datenmengen, wie sie zum Beispiel in Form des Wikipedia-Netzwerks auftreten, war kein Problem, solange die Daten in die richtigen Formate gebracht werden können, sodass sie vom Framework verarbeitet werden können.

Außerdem konnte dadurch, dass das Framework bereits eingesetzt wurde, wichtiges Feedback gesammelt werden, dass eine zukünftige Weiterentwicklung des Frameworks erlaubt.

8. Weiterentwicklungspotential

Software steht unter ständiger Veränderung. Es gibt keine Softwaresysteme, bei denen die Architektur und das Design zu Beginn festgelegt werden und sich dann in keiner Weise mehr ändern.

Vor allem die Erweiterung mit neuen Features führt immer wieder kleine Änderungen ein, die sich mit der Zeit aber recht stark auf das Gesamtsystem auswirken. Aus diesem Grund sollte öfters eine Restrukturierung vorgenommen und der Code überarbeitet werden.

Auch bei der Weiterentwicklung des Frameworks sollte die ständige Verbesserung der Codequalität im Vordergrund stehen, damit zumindest das Gegenteil, nämlich die Verschlechterung der Qualität der Codebasis verhindert werden kann.

Wie das Fallbeispiel gezeigt hat, ist der Kern des Frameworks im Moment recht gut einsetzbar und für die Aufgaben geeignet, für die das Framework bisher eingesetzt wurde.

Das Weiterentwicklungspotential besteht vor allem darin weitere Navigations-Strategien, NodeSelectoren, LinkFilter und Attrition-Implementierungen anzubieten.

Was für den Kern des Frameworks eventuell interessant wäre, ist eine Implementierung eines Graph-Typen, der die Speicherung von beliebigen Daten auf den Knoten bzw. Kanten erlaubt. Damit könnten Informationen gespeichert werden, die zusätzlichen Einfluss auf Navigations-Strategien nehmen.

Außerdem wäre es überlegenswert, den gesamten Output des Frameworks in ein Logging-System überzuführen um noch mehr Kontrolle über den Output zu erhalten.

Literaturverzeichnis

[1] T. Berners-Lee, R. Cailliau, J.-F. Groff und B. Pollermann, „World-Wide Web: The Information Universe“, *Internet Research (Vol. 2, No. 1),* pp. 52-58, 1992.

[2] T. Alby, Web 2.0: Konzepte, Anwendungen, Technologien (3. Auflage), München: Carl Hanser Verlag, 2008.

[3] F. Geigl, *Analyse und Modellierung des menschlichen Navigationsverhaltens in einem Wikipedia-Netzwerk,* Graz: Technische Universität Graz, noch nicht veröffentlicht.

[4] M. Horwath, *Datenaufbereitung und Entwicklung einer Abbruchsimulation zur Modellierung von Benutzernavigation,* Graz: Technische Universität Graz, noch nicht veröffentlicht.

[5] R. Diestel, Graphentheorie, Berlin: Springer Verlag, 2010.

[6] D. Easley und J. Kleinberg, Networks, Crowds, and Markets: Reasoning About a Highly Connected World, Cambridge University Press, 2010.

[7] M. Newman, Networks: An Introduction, Oxford University Press, 2009.

[8] D. Applegate, R. Bixby, V. Chvátal und W. Cook, The Traveling Salesman Problem: A Computational Study, Princeton: Princeton University Press, 2007.

[9] J. Reekie und R. McAdam, A Software Architecture Primer, Angophora Press, 2006.

[10] B. Meyer, Object-oriented software construction (2nd Edition), Upper Saddle River: Prentice-Hall, 1997.

[11] G. Booch, R. Maksimchuk, M. Engel, B. Young, J. Conallen und K. Houston, Object-Oriented Analysis and Design with Applications (3rd Edition), Amsterdam: Addison-Wesley Longman, 2007.

[12] D. Armstrong, „The quarks of object-oriented development“, *Communications of the ACM (Volume 49 Issue 2),* pp. 123-128, 2006.

[13] P. Petrov und U. Buy, „A Systemic Methodology for Software Architecture Analysis and Design“, in *8th International Conference on Information*

Technology: New Generations (ITNG), 2011.

[14] R. Krikhaar, A. Postma, A. Sellink, M. Stroucken und C. Verhoef, „A Two-phase Process for Software Architecture Improvement“, in *International Conference on Software Maintenance (ICSM)*, Oxford, England, 1999.

[15] J. Guo, „An Approach for Modeling and Designing Software Architecture“, in *Proceedings of the 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'03)*, 2003.

[16] E. Gamma, R. Helm, R. Johnson und J. Vlissides, Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software, Addison-Wesley Verlag, 2004.

[17] M. Fowler, „Inversion Of Control“, [Online]. Available: http://martinfowler.com/bliki/InversionOfControl.html. [Zugriff am 21 Januar 2013].

[18] M. Fowler, „Inversion of Control Containers and the Dependency Injection“, [Online]. Available: http://martinfowler.com/articles/injection.html. [Zugriff am 21 Januar 2013].

[19] J. Leskovec und E. Horvitz, „Planetary-Scale Views on a Large Instant-Messaging Network“, in *Proceedings of the 17th international conference on World Wide Web*, Beijing, China, 2008.

[20] R. White und S. Drucker, „Investigating Behavioral Variability in Web Search“, in *Proceedings of the 16th international conference on World Wide Web*, Banff, 2007.

[21] B. Huberman, P. Pirolli, J. Pitkow und R. Lukose, „Strong Regularities in World Wide Web Surfing“, in *Proceedings of the seventh international conference on World Wide Web*, Amsterdam, 1998.

[22] C. Olston und E. H. Chi, „ScentTrails: Integrating browsing and searching on the Web“, *ACM Transactions on Computer-Human Interaction (TOCHI) (Volume 10, Issue 3),* pp. 177-197, 2003.

[23] J. Teevan, C. Alvarado, M. Ackerman und D. Karger, „The Perfect Search Engine Is Not Enough: A Study of Orienteering Behavior in Directed Search“, in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Wien, 2004.

[24] D. Gleich, P. Constantine, A. Flaxman und A. Gunawardana, „Tracking the Random Surfer: Empirically Measured Teleportation Parameters in PageRank“, in *Proceedings of the 19th international conference on World wide web*, Raleigh, 2010.

[25] R. West und J. Leskovec, „Automatic versus Human Navigation in Information Networks“, in *Proceedings of the 23rd ACM conference on Hypertext and social media*, Milwaukee, 2012.

[26] M. Boguna, D. Krioukov und K. C. Claffy, „Navigability of Complex Networks“, *Nature Physics,* pp. 74-80, 2009.

[27] J. Kleinberg, „Complex Networks and Decentralized Search Algorithms“, in *Proceedings of the International Congress of Mathematicians*, Madrid, 2006.

[28] S. Milgram, „The Small-World Problem“, *Psychology Today,* pp. 60-67, 1967.

[29] P. Dodds, R. Muhamad und D. Watts, „An Experimental Study of Search in Global Social Networks“, *Science (Vol. 301),* pp. 827-829, 2003.

[30] L. Adamic und E. Adar, „How to search a social network“, in *Social Networks (Volume 27, Issue 3)*, 2005, pp. 187-203.

[31] D. Watts und S. Strogatz, „Collective dynamics of 'small-world' networks“, *Nature,* pp. 440-442, 1998.

[32] J. Voss, „Measuring Wikipedia“, in *Proceedings of the 10th International Conference of the International Society for Scientometrics and Informetrics*, Stockholm, 2005.

[33] L. Buriol, C. Castillo, D. Donato, S. Leonardi und S. Millozzi, „Temporal Analysis of the Wikigraph“, in *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, 2006.

[34] G. De La Calzada und A. Dekhtyar, „On Measuring the Quality of Wikipedia Articles“, in *Proceedings of the 4th workshop on Information credibility*, Raleigh, USA, 2010.

[35] R. West und J. Leskovec, „Human Wayfinding in Information Networks“, in *Proceedings of the 21st international conference on World Wide Web*, Lyon, 2012.

[36] D. Helic, „Analyzing User Click Paths in a Wikipedia Navigation Game“, in *Proceedings of the 35th International Convention of Information Communication Technology, Electronics and Microelectronics*, Opatija, 2012.

[37] „SNAP: Package Description“, [Online]. Available: http://snap.stanford.edu/snap/description.html. [Zugriff am 16 Januar 2013].

[38] „Stanford Large Network Dataset Collection“, [Online]. Available: http://snap.stanford.edu/data/index.html. [Zugriff am 16 Januar 2013].

[39] P. Ayers, C. Matthews und B. Yates, How Wikipedia Works: And How You Can Be a Part of It, San Francisco: No Starch Press, 2008.

[40] „Wikipedia Statistics - Tables - Article count (official)“, [Online]. Available: http://stats.wikimedia.org/EN/TablesArticlesTotal.htm. [Zugriff am 11 Januar 2013].

[41] M. Völkel, M. Krötzsch, D. Vrandecic, H. Haller und R. Studer, „Semantic Wikipedia“, in *Proceedings of the 2006 international symposium on Wikis*, Odense, 2006.

[42] R. West, J. Pineau und D. Precup, „Wikispeedia: An Online Game for Inferring Semantic Distances between Concepts“, in *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*, Pasadena, Calif., 2009.

Abbildungsverzeichnis

Listingverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

bzw.	beziehungsweise
ca.	circa
d. h.	das heißt
et al.	et alii
etc.	et cetera
GB	Gigabyte
GCC	GNU compiler collection
HTML	Hypertext Markup Language
ID	Identifikationsnummer
inkl.	inklusive
MUN	Modeling User Navigation
SNAP	Stanford Network Analysis Platform
u. a.	unter anderem
UML	Unified Modeling Language
usw.	und so weiter
vgl.	vergleiche
z.B.	zum Beispiel