Masterarbeit

Visualisierung von Wirkketten anhand von PLM Szenarien

Dipl.-Ing. (FH) Michael Nestler

Masterarbeit

Visualisierung von Wirkketten anhand von PLM Szenarien

Masterarbeit

an der

Technischen Universität Graz

Dipl.-Ing. (FH) Michael Nestler

Institut für Wissensmanagement (IWM),
Technische Universität Graz
A-8010 Graz, Österreich

Begutachter:	Assoc.Prof. Dipl.-Ing. Dr.techn. Denis Helic
Betreuer:	Dipl.-Ing. (FH) Andrea Denger

12. Dezember 2012

Kurzfassung

Diese Arbeit beschäftigt sich mit der Visualisierung von gesammelten Daten, die während eines mehrjährigen Projektes in Bezug auf zukünftiges Product Lifecycle Management (PLM) in der Automobilindustrie gesammelt wurden. Ein wichtiger Aspekt der Visualisierung liegt darin, projektaußenstehenden Personen, die Inhalte leicht und schnell vermitteln zu können. Die Arbeit ist im Wesentlichen in zwei Teile unterteilt.

Im theoretischen Teil werden im Allgemeinen verschiedene Ansichten der Informationsvisualisierung aus der Literatur behandelt. Vertiefend wird auf die Visualisierung von Textdokumenten, Netzwerken und Graphen eingegangen. Die Visualisierung von Graphen nimmt im praktischen Teil einen hohen Stellenwert ein, weswegen diese auch im theoretischen Teil genauer behandelt wird.

Aufbauend auf den theoretischen Teil befasst sich der praktische Teil mit der Implementierung einer Client/Server-Software. Die gesammelten PLM relevanten Daten in Form einer Datei, werden automatisch von der Serversoftware eingelesen und in ein zuvor definiertes Datenmodell umgewandelt. Dabei kommen zwei Datenmodelle zur Anwendung. Zum einen werden die Daten intern in der Software verwaltet, zum anderen werden die Daten in ein Triple Modell, in Form einer OWL Datei gespeichert. Je nach Datenmodell erfolgte die Abfrage entweder über interne Suchabfragen oder über SPARQL Abfragen. Das Ergebnis dieser Abfrage wird clientseitig in einem Webbrowser graphisch aufbereitet und dargestellt. Hierfür werden Technologien und Codebibliotheken verwendet, welche allgemein unter HTML5 zusammengefasst sind. Für die Datenübertragung zwischen Client und Server werden JSON (Javascript Object Notation) und AJAX (Asynchronous Javascript and XML) verwendet.

Abstract

This works deals with visualization techniques for data which were collected over several years in a project, related to the product lifecycle management (PLM) in the automotive industry. The data visualization should help people who are not familiar with the topic to learn and understand the content fast and easily. The work is divided into two main parts.

The related work part shows a variety of information visualization examples from science literature. Examples, particularly in the field of text and document visualization respectively to network and graph visualization are shown and discussed. Because the graph visualization takes an important role in the practical work, it will be discussed in detail in the theoretical part.

The practical part of the thesis deals with a client/server application. The file with the collected PLM data is read by the server software and can be converted into two different data models. One data model is held internally by the server software, the other one is stored as a triple model in an OWL (Web Ontology Language) file. According to the data models two query methods are used. For the internal data model self written query functions are used. The triple model is queried by SPARQL commands. The client side gets the results of the queries via AJAX (Asynchronous Javascript and XML) as a JSON (Javascript Object Notation) Object. With the use of HTML5 techniques and Javascript libraries the data is visualized on the client side.

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.

Ort	Datum	Unterschrift

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.

Place	Date	Signature

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Allgemeine Beschreibung des Projektes FuturePLM

Die Entwicklung eines neuen Automobiles hat sich in den letzten Jahren stark verändert. Bei abnehmender Entwicklungszeit steigt gleichzeitig die Anzahl der Produktderivate und -varianten. Stärker, denn je müssen verschiedene Disziplinen in der Produktentwicklung verteilt an mehreren Standorten dieser Welt, ihr Wissen, Erfahrung und aktuelle projekt- und produktionsspezifische Daten über den gesamten Produktentwicklungszyklus austauschen. In diesem Zusammenhang beinhaltet PLM (Product Lifecycle Management) Prozesse, Organisationsstrukturen, Methoden und unterstützende IT (engl.: Information Technology) Systeme. Aus all diesen Bereichen darf auch der Faktor Mensch nicht außer Acht gelassen werden. Ist es dennoch möglich all diese Bereiche zu motivieren, synchronisieren und zu strukturieren kann sich ein Unternehmen von seinen Konkurrenten hervorheben und gewinnbringend arbeiten. (Denger et al. [2011]).

”Das Kompetenzzentrum Das Virtuelle Fahrzeug (ViF) ermittelte im K2 - Forschungsprojekt ”FuturePLM” zukünftige Anforderungen an Product Lifecycle Management (PLM) im Zusammenhang mit der zentralen Rolle und Einbindung des Menschen. Es wurden neue Methoden, Konzepte und Lösungsansätze rund um das zukünftige „Product Lifecycle Management“, d.h. das Management eines Produktes über seinen gesamten Lebenszyklus hinweg erforscht. Der Begriff PLM wird heute sehr unterschiedlich verstanden und nicht selten fälschlicherweise auf ein IT-System reduziert. Im eigentlichen Sinn stellt Product Lifecycle Management ein strategisches Konzept zum Management eines Pro-

duktes und seines IP (Intellectual Property) über den gesamten Produktlebenszyklus dar. Die Zielsetzung ist ein perfektes Zusammenspiel aller beteiligten Elemente entlang des Produktlebenszyklus, in den unterschiedlichen Domänen und über Standorte hinweg. Das Projekt lieferte einen Beitrag, wie PLM in der Zukunft erfolgreicher gestaltet und umgesetzt werden kann.

Es wurden vier Szenarien für das Jahr 2020 entwickelt, um daraus Anforderungen an zukünftiges PLM und in weitere Folge an den Arbeitsplatz der Zukunft abzuleiten (siehe Abbildung 1.1, rechter Teil des Bildes). Die entstehenden Fragestellungen wurden in transdisziplinären Foren zur Diskussion gestellt, mit dem Ziel einen neuen Blickwinkel einzunehmen und neuartige Lösungsansätze zu generieren (siehe Abbildung 1.1, linker Teil des Bildes). Aufbauend auf die gewonnenen Erkenntnisse wurde ein Anforderungskatalog (siehe Abbildung 3.1) für zukünftiges PLM und an den Arbeitsplatz der Zukunft erstellt. (vgl. [Schmeja und Denger, 2011])" (Denger et al. [2012, Seite 6-7])

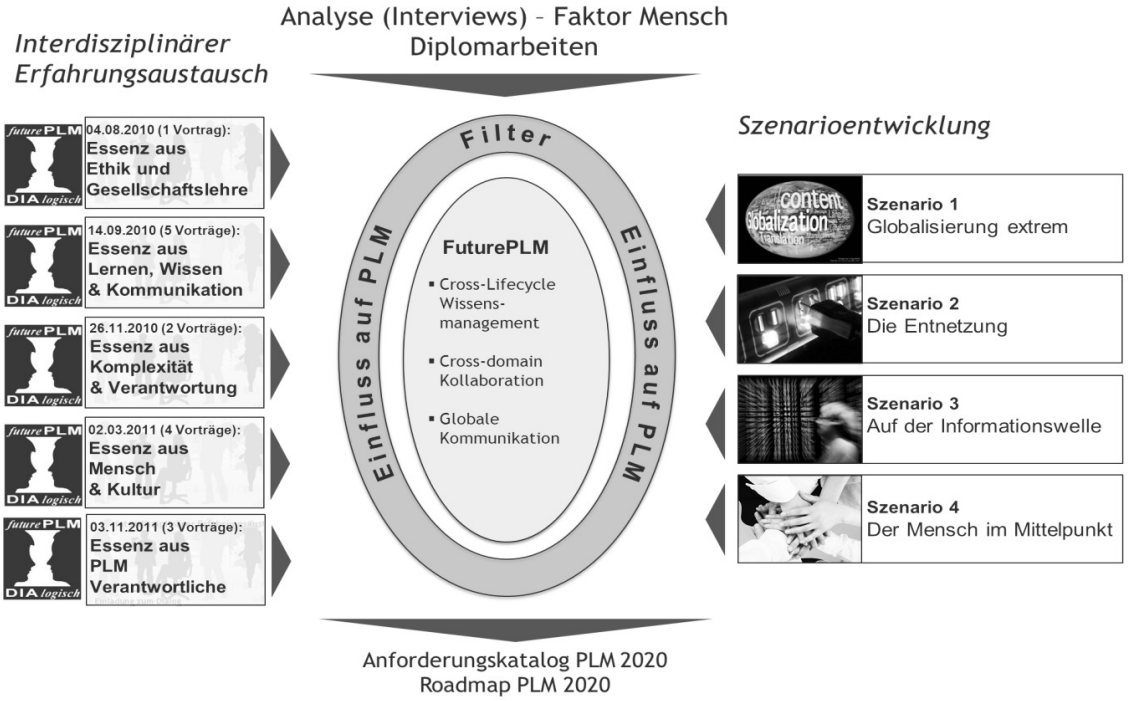


Abbildung 1.1: Einflussfaktoren des Anforderungskataloges [Denger et al. [2012, Seite 7])]

Dieser Anforderungskatalog beinhaltet im Wesentlichen Zusammenhänge der erarbeiteten Szenarien, und der zugehörigen Problemfelder sowie Annahmen, Anforderungen und möglichen Lösungsansätze. Eine solche Wirkkette kann folgendermaßen aussehen:

Wählt man beispielsweise das Szenario "Der Mensch im Mittelpunkt" so wurde ei-

nes der Probleme im Zusammenhang mit diesem Szenario als ”Arbeitsplatz der Zukunft definiert bzw. gefunden. Folglich kann angenommen werden, dass der Arbeitsplatz der Zukunft nicht lokal gebunden sein wird. Dies stellt das Unternehmen vor neuen Herausforderungen/Problemen wie z.B. dass der Mitarbeiter trotzdem vollen Zugriff auf die benötigten Daten haben muss. Der Mitarbeiter soll auch das Gefühl haben Teil des Teams zu sein, obwohl dieser nur selten persönlichen Kontakt zu anderen Teammitgliedern pflegen kann. Der Mitarbeiter soll sich an diesem Arbeitsplatz der Zukunft immer ”wie zu Hause” fühlen. Dieses einfache Beispiel zeigt, dass große Herausforderungen an die Unternehmen in der Zukunft gestellt werden. Generell zeigt Abbildung 1.2 den Aufbau einer Wirkkette. (Denger et al. [2011])

Abbildung 1.2: Allgemeiner Überlick einer Wirkkette im Anforderungskatalog

Wie oben schon erwähnt, wurden alle gefunden Beispiele in den Anforderungskatalog aufgenommen. Dieser dient nun als neue Vorgehensweise für neue Definitionen und Umsetzungen für das zukünftige PLM. Bisher wurden zur Definition und Umsetzung bestehender PLM-Systeme lediglich meist Funktionsgruppen wie CAD-Datenverwaltung, Prüfung und Freigabe von Fertigungsdokumenten, Änderungsdienst und Stücklistenverwaltung in Betracht gezogen. Die Herausforderung besteht nun darin den Inhalt des Anforderungskataloges, welcher auch bekannte Führungsanforderungen beinhaltet, in ein zukünftiges PLM System einzubinden.

1.2 Kernthemen der vorliegenden Arbeit

Wie in Punkt 1.1 beschrieben, wurde durch das Projekt ”FuturePLM” ein Anforderungskatalog erstellt, welcher eine Sammlung an Wirkketten darstellt und in Form einer Excel-Datei gespeichert wurde. Da es aufgrund der Fülle und Komplexität des Themas sehr schwierig ist die gesammelte textuelle Information außenstehenden Personen zu vermitteln, wird in dieser Arbeit nach Alternativen gesucht, wie diese Information aufbereitet und damit schnell und verständlich anderen zugänglich gemacht werden kann. Ein weiterer Schwerpunkt der Arbeit ist das Einsetzen und Austesten sogenannter neuer Technologien, welche im wesentlichen webbasierend sein sollen. Somit können die Anforderungen

im Groben folgendermaßen zusammengefasst werden:

- Aufbereiten und analysieren der vorhandenen Daten
- Visualisieren der Daten
- Verwendung von neuen webbasierenden Technologien
- Schaffung einer Diskussionsgrundlage aufgrund der visualisierten Daten

Aufgrund der geforderten Punkte ergibt sich auch die Einteilung dieser Arbeit. Im Kapitel 2 wird der theoretische Teil der Arbeit in Form einer Literaturrecherche und einer Recherche anderer wissenschaftlicher Arbeiten in Bezug auf Informationsvisualisierungen behandelt. Ab Kapitel 3 beginnt die Beschreibung des praktischen Teils. Dabei wird näher auf die Anforderungen der Software und die daraus resultierende Technologieauswahl eingegangen. Spezielle Eigenschaften der ausgewählten Technologien werden in Kapitel 3 ebenso näher behandelt. Das Kapitel über die Implementierung (siehe Kapitel 5) beschreibt die serverseitige sowie clientseitige Funktionalität der entwickelten Software. In Kapitel 6 geht es ebenfalls um die Implementierung jedoch um einem anderen Lösungsansatz (Verwendung von Ontologien und semantischen Strukturen) verglichen mit jenem welcher unter Kapitel 5 beschrieben wird. Am Ende dieser Arbeit wird ein Überblick über die Evaluierung beziehungsweise ein Ausblick auf mögliche zukünftige Arbeiten gegeben. Abschließend findet sich ein Kapitel mit einer Zusammenfassung der hier vorliegenden Arbeit.

Kapitel 2

Informationsvisualisierung

2.1 Kapitelüberblick

In diesem Kapitel wird ein allgemeiner Überblick über die Visualisierung gegeben. Zu Beginn des Kapitels werden historische Beispiele angeführt um zu zeigen warum und wie sich die Informationsvisualisierung entwickelt hat. Danach wird anhand von Literaturbeispielen eine Klassifizierung der Visualisierung vorgenommen. Die Klassifizierung dient dazu, Definitionen einzuführen und Arbeiten in bestimmte Kategorien einordnen zu können. Ausgehend vom *Visual Information Seeking Mantra* nach Shneiderman [1996] werden Literaturbeispiele für Text und Dokumenten Visualisierung näher beschrieben. Einen Großteil des Kapitels nehmen die Visualisierungsarten für Bäume, Graphen und Netzwerke ein. In diesem Abschnitt wird genauer auf verschieden Methoden aus der Literatur eingegangen. Ziel des Kapitels ist es einerseits Beispiele aus der Literatur anzuführen und zu erklären andererseits die erworbenen Kenntnisse in den praktischen Teil einfließen zu lassen.

2.2 Einführung in die Visualisierung

2.2.1 Allgemeines

Die Informationsvisualisierung hat seit ca. 15-20 Jahren einen hohen Anstieg zu verzeichnen. Dies ist letztlich auf den Einsatz von Computern und verwandten Technologien zurückzuführen. Mit der heute zu Verfügung stehenden Technologie ist es möglich,

schneller und detailreicher, Information zu visualisieren. Jedoch gab es schon früher Visualisierungen, welche heute noch immer in Verwendung sind und als Klassiker der Informationsvisualisierung gelten. Zu diesen Klassikern zählen die drei folgenden Beispiele, welche hier stellvertretend für die Ursprünge der Informationsvisualisierung angeführt werden. Diese Beispiele werden auch in [Spence, 2007] angeführt und kurz dokumentiert.

2.2.2 Historische Beispiele

Kartenzeichnung von Minard

Charles Joseph Minard war Kartenzeichner für Napoleon Bonarparte. Die hier angeführte Karte (siehe Abbildung 2.1) zeigt den Marsch von Polen nach Moskau der französischen Armee. Jemand der die Karte noch nie zuvor gesehen hat und auch mit der Thematik nicht vertraut ist, kann sich trotzdem schnell ein Bild von der Situation machen. Die Karte spiegelt geografische Landschaftspunkte, Truppenstärke, Marschrichtung und Temperatur wider. So wird zum Beispiel die Truppenstärke als Dicke der Linienzeichnung dargestellt. Die Farben der Linien symbolisieren den Marsch auf Moskau (braune Linie) und den Rückzug der Einheiten (schwarze Linie). Im unteren Bereich der Karte ist der Temperaturverlauf vermerkt. Gut zu erkennen ist die Korrelation zwischen niedrigen Temperaturen und der Dezimierung der Truppe auf deren Rückzug. Markante geografische Landschaftspunkte, vor allem jene die entscheidend für die Truppen waren sind ebenfalls vermerkt und stellen einen Bezug zur Truppenstärke her. So ist klar zu erkennen, dass die Überquerung des Flusses Beresina erhebliche Verluste in der französischen Armee verursachte. Die Karte von Minard verdeutlicht, dass gute Visualisierung von Information, wenig Erklärung bedarf. Anhand dieser Karte ist eine langlebige Erinnerung in unseren Köpfen gespeichert, viel besser als würde die hier visualisierten Fakten in textueller Aufzählungsform dargestellt werden.

Polar-Area Diagramm von Nightingale

Das Polar-Area-Diagramm von Nightingale ist ebenfalls selbsterklärend und hinterlässt im Kopf des Betrachters eine klare Vorstellung der Information. Dieses, für zyklisch wiederkehrende Ereignisse, geeignete Polar-Area-Diagramm wurde das erste Mal von Florence Nightingale verwendet. Florence Nightingale war Krankenschwester und zeigte mit diesem Diagramm die Anzahl der Toten pro Monat in einem Krankenhaus auf. Dabei ist

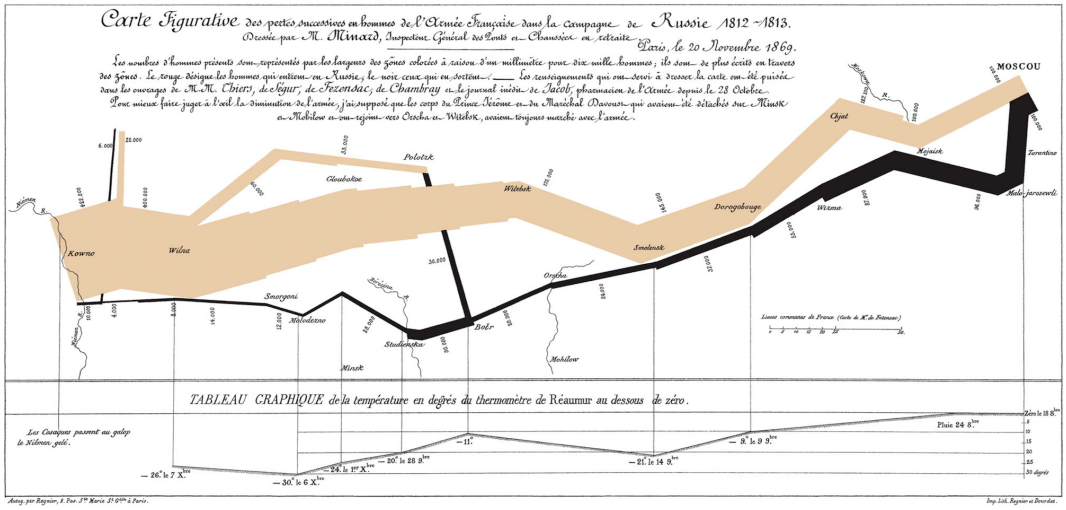


Abbildung 2.1: Karte von Minard - Napoleons Russlandfeldzug [http://www.edwardtufte.com/tufte/minard Zuletzt besucht am: 2012.08.03]

der Winkel eines Kreissegmentes proportional zu der Anzahl der Tage eines bestimmten Monats und die Fläche eines Segmentes entspricht der Anzahl der Todesfälle in diesem Monat. Durch die zyklische Darstellung kann der Unterschied graphisch gezeigt werden. Wie im Falle von Florence Nightingale, die Krankenhaus Bedingungen verbesserte und somit von Monat zu Monat die Flächen, welche der Anzahl der Todesfälle entsprechen, immer kleiner wurden. Abbildung 2.2 zeigt ein Polar-Area-Diagramm von Nightingale, welches die Todesursachen in den Krimkriegen monatsweise dokumentiert. Dabei sind alle Flächen ausgehend vom Mittelpunkt gezeichnet. Blaue Flächen zeigen die Anzahl der Toten, welche vermeidbar gewesen wären, rote Flächen entsprechen den Toten, welche durch Infektionen verstorben sind und schwarze Flächen kennzeichnen alle Toten anderer Ursachen. Durch diese übersichtliche Visualisierung ist es möglich Ursachen zu erkennen und dementsprechend Korrekturen einzuführen, um diese letztendlich wieder einer Kontrolle zu unterziehen.

Königsberger Brückenproblem

Als dritte historische Visualisierungsart, welche in der heutigen Zeit einen hohen Stellenwert in der Darstellung von Netzwerken jeglicher Art einnimmt, ist die Graphenvisualisierung. In der wissenschaftlichen Landschaft tauchte diese Art der Visualisierung erst durch Leonhard Euler mit dem Königsberger Brückenproblem auf. Die Fragestellung lautete, ob es möglich wäre alle sieben Brücken in Königsberg nur einmal zu überqueren und dabei einen Rundweg zu beschreiten, welcher einen wieder an den Startpunkt zurückführt. Während Euler noch eher eine Zeichnung (siehe Abbildung 2.3a), als einen

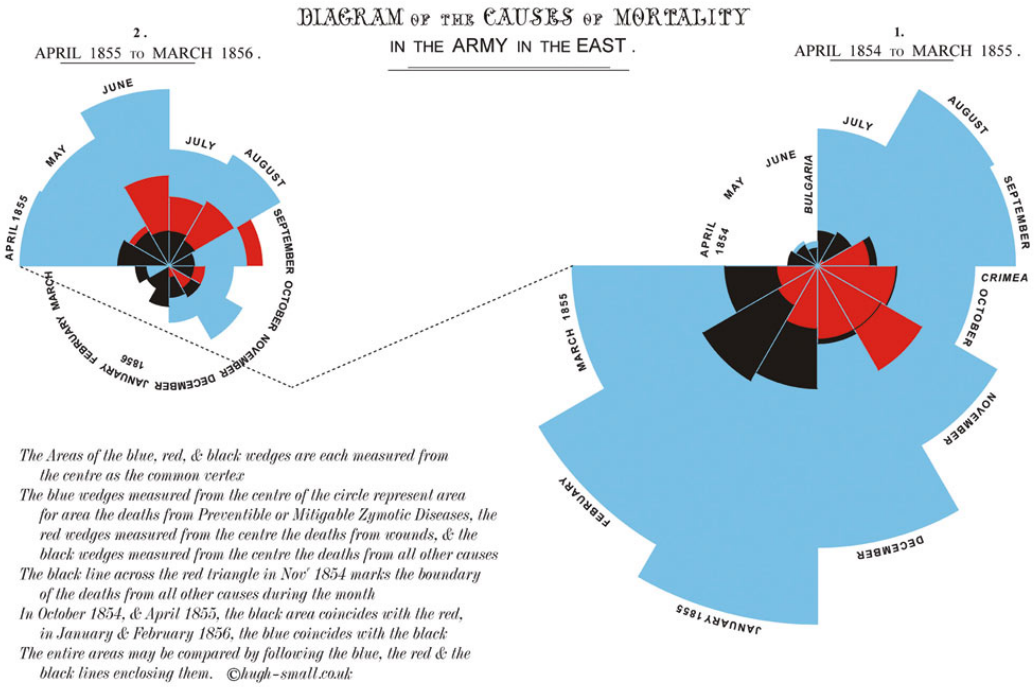


Abbildung 2.2: Polar-Area-Diagramm von Nightingale [http://www.historyandwomen.com/2011/08/florence-nightingale.html Zuletzt besucht am: 2012.08.03)]

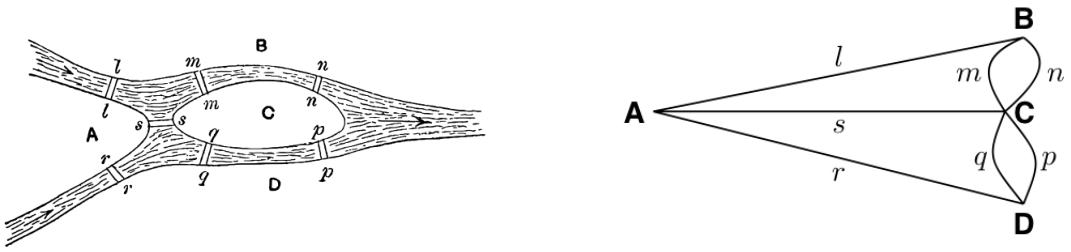
Graph anfertigte, verfeinerte Ball [1905] die Darstellung (siehe Abbildung 2.3) in ein abstraktes Knoten-Link-Diagramm.

Knoten-Link-Diagramme sind generell ein gutes Visualisierungsinstrument. (vgl mit [Battista et al., 1999]):

- Graphen können bei Daten eingesetzt werden, die Beziehungen zueinander aufweisen. Zum Beispiel werden Graphen zur Darstellung von Dateisystemen, Stammbäumen oder des World Wide Web eingesetzt. In der Softwareindustrie werden Softwarekomponenten als Graphen in Form von UML Charts dargestellt. Software Programme zum Zeichnen von elektrischen Schaltplänen oder Platinen Layouts sowie auch Navigationssystem und Projektmanagement Tools arbeiten im Hintergrund mit Graphen um so den kürzesten, schnellsten oder kritischen Pfad zu ermitteln.

- Die Graphentheorie wird seit Jahrhunderten erforscht und weiterentwickelt. Dieses Fundament ist auch Ausgangspunkt vieler verschiedener Algorithmen zu graphischen Darstellung dieser Graphen. Dabei spielt auch die Ästhetik der Darstellung, damit das Auge des Betrachters einem bestimmten Pfad leicht folgen kann, eine entscheidende Rolle.

- Die Darstellung mit Hilfe von Graphen kann Aufgabenstellungen von einer anderen Seite beleuchten und damit mehr Informationen liefern, welche direkt ohne weitere Interpretationen verwendet werden können.



(a) Darstellung nach Euler **(b)** Darstellung nach Ball

Abbildung 2.3: Graphische Darstellung des Königsberger Problems [Ball [1905, Seite 144]]

2.3 Klassifizierung der Informationsvisualisierung

2.3.1 Allgemeines

In der Literatur gibt es mehrere Klassifizierungen für die Informationsvisualisierung. In diesem Unterkapitel sollen einige verschiedene Arten erörtert werden, mit dem Ziel den Inhalt dieser Arbeit eindeutig zu klassifizieren.

2.3.2 Wissenschaftliche und nicht wissenschaftliche Einteilung

Visualisierung wird nach Card et al. [1999] folgendermaßen definiert "....the use of computer supported, interactive, visual representations of data to amplify cognition....". Diese allgemeine Definition ist grundsätzlich anerkannt, wird aber im Detail verschieden interpretiert. Eine angesehene Einteilung (vgl. mit [Spence, 2007] und [Shneiderman, 1996]) spricht von *Scientific Visualization* und abstrakter *Information Visualization*.

Wobei unter *Scientific Visualization* die Visualisierung von wissenschaftlichen Daten meist Messdaten wie z.B. Luftströme über Flugzeugflügel, Volumina oder die Wärmeableitung einer Verbrennungsmaschine verstanden wird. Einhergehend mit der *Scientific-Visualization* ist die Darstellung von geographischen Daten. Das heißt, die wissenschaftlichen Daten werden oftmals mit räumlichen Daten verbunden und dargestellt.

Information Visualization beinhaltet Daten, welche nicht aus der Wissenschaft entspringen und keinen geographischen Hintergrund besitzen. Beispiele für Daten, welche der *Information Visualization* zugeordnet werden, sind Finanzdaten oder Textinformation in Dokumenten.

Diese Einteilung ergibt leider Graubereiche für Daten, welche entweder der *Scientific Visualization* oder der *Information Visualization* zugeordnet werden können. Zum Beispiel plädiert Rhyne et al. [2003] für eine Einteilung nach *Scientific-Visualization* oder *Information-Visualization* je nachdem, ob die geographischen Daten zum Datensatz gegeben sind (*Scientific Visualization*), oder zum bestehenden Datensatz hinzugefügt werden (*Information Visualization*). Ihr Argument diesbezüglich liegt darin, dass *Scientific Visualization* sehr wohl informativ und umgekehrt *Information Visualization* auch wissenschaftlich ist.

2.3.3 Einteilung nach der Wissenspyramide

Eine andere Einteilung baut auf der sogenannten Wissenspyramide (siehe Abbildung 2.4) auf, welche zwischen Daten, Information und Wissen unterscheidet. Folgende Definitionen wurden von Burkhard [2005] übernommen.

Daten

Unter Daten versteht man eine formale Repräsentation von Rohdaten, welche einem definierten Format und einer bestimmten Bedeutung zugeordnet werden können. Diese Daten können von Menschen oder Computern gespeichert und weitergegeben werden. Daten an sich besitzen keine Bedeutung, nur durch weitere Interpretation bzw. Verarbeitung kann diese Bedeutung zugeordnet werden. Ab diesem Punkt werden aus Daten Informationen.

In [Tory und Möller, 2004] werden ebenfalls verschiedene Typen von Daten unterschieden. Grundsätzlich werden abhängige und unabhängige Variablen unterschieden und folgendermaßen weiter unterteilt.

- Skalare, Vektoren oder kompliziertere Strukturen
- Diskrete (z.B. Anzahl der Einwohner einer Stadt) oder kontinuierlich (z.B. Einwohnerdichte einer Stadt)
- Nominale Daten, welche keine natürliche Ordnung besitzen (z.B. Banane, Birne, Ananas). Ordinale Daten, welche eine natürlich Ordnung besitzen (z.B.

klein, mittel, groß bzw. die alphabetische Ordnung von Ananas, Banane, Birne). Intervall Daten, welche einem metrischen System zugeordnet sind und somit Berechnungen durchgeführt werden können.

Information

Wie oben erwähnt resultiert die Information aus der Bearbeitung, Manipulation und Interpretation von Daten. Information hat kein fix definiertes Format und die Bedeutung ist einem bestimmten Kontext zugeordnet.

Wissen

Wissen wiederum wird aus Information generiert, welches eingeteilt, identifiziert und validiert wird. Wissen basiert auf Überzeugungen der Realität und beinhaltet Beschreibungen, Hypothesen oder formalisierte Konzepte wie Wörterbücher, Taxonomien und Ontologien.

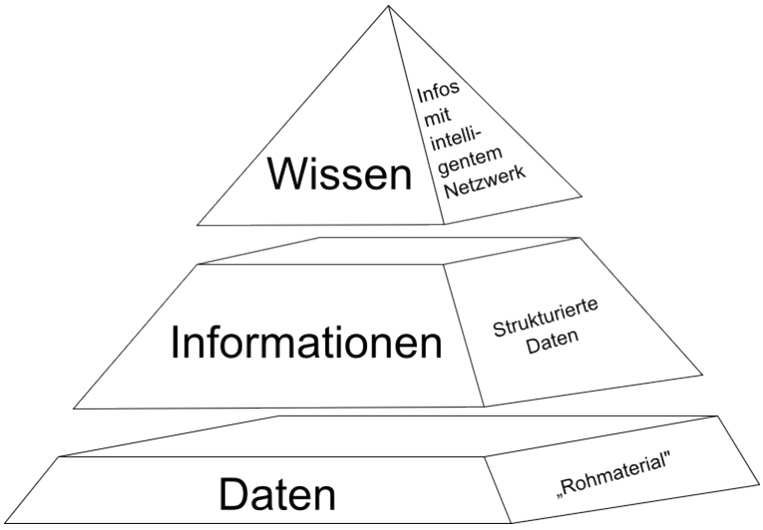


Abbildung 2.4: Wissenspyramide (Quelle: `http://commons.wikimedia.org/w/index.php?title=File:Wissenspyramide.svg&page=1` Zuletzt besucht am: 2012.09.03)

Aus diesen Definitionen ergeben sich die Klassifizierungen *Data/Scientific Visualization*, *Infomation Visualization* und *Knowledge Visualisation*. Durch diese Einteilung lässt sich auch eine Aussage über die benötigte Rechenleistung bzw. Dartstellungskomplexität treffen. Die benötigte Rechenleistung ist für die *Data/Scientific Visualization* am Höchsten und nimmt bei der *Infomation Visualization* bis hin zur *Knowledge Visualization* ab. Umgekehrt verhält es sich mit der Darstellungskomplexität. Diese nimmt ausgehend von der *Knowledge Visualization* bis hin zur *Data/Scientific Visualization* ab.

2.3.4 Alternativer Ansatz

Einen anderen Ansatz in der Taxonomy bezüglich Informationsvisualisierung wählen Tory und Möller [2004]. Für die beiden Autoren ist die grundsätzliche Einteilung in Scientific Visualization und Information Visualization zu widersprüchlich. Und in der Tat ist es oft schwer, nach der von oben beschriebenen Einteilung, praktische Anwendungen zuzuordnen. Zum Beispiel basiert die Flugverkehrsüberwachung auf physikalischen und geographischen Daten, dennoch ist diese Disziplin eher der *Information Visualization*, als der *Scientific Visualization* zuzuordnen. Als weitere Motivation zur Einführung einer neuen Taxonomie geben die beiden Autoren folgende Gründe an:

1. Benutzer, welche nicht aus dem Informationsvisualisierungsbereich kommen, sollen leicht geeignete Literatur finden können. Dafür muss diese aber sinnvoll kategorisiert sein.

2. Wissenschaftler soll es erleichtert werden, ihre Arbeit einer bestimmten Kategorie zuordnen zu können bzw. um auch Wissenschaftler ihresgleichen einfacher zu finden. Die neue Einteilung könnte den gesamten Forschungszweig schneller und fokussierter entwickeln. Letztendlich will man dazu anregen, Visualisierung von verschiedenen Blickwinkeln zu betrachten und somit neue Ideen in das Forschungsgebiet einzubringen.

Tory und Möller [2004] nehmen von der Einteilung der Visualisierung ausgehend von den Rohdaten Abstand. Stattdessen wird der Begriff des Design Modells eingeführt. Dieser Begriff bedeutet, dass zwar die Daten in einer bestimmten Form (z.B. kontinuierlich oder diskrete) vorliegen können, jedoch der Designer, welcher für die Visualisierung verantwortlich ist, entscheidet, wie das Design Modell auszusehen hat. Kontinuierliche Design Modelle können entstehen, wenn die Daten interpoliert werden können. Zum Beispiel ist es einfach metrische Intervalldaten in einem diskreten aber auch kontinuierlichen Design Model darzustellen. Nominale und ordinale Daten können meist nur als diskrete Modelle dargestellt werden, da eine Interpolierung wenig Sinn macht. Zum Beispiel gibt es keine sinnvolle Interpretation, wenn man zwischen den Daten ”Mann” und ”Frau” interpoliert.

Die zweite Einteilung, die getroffen wurde, betrifft die Visualisierung an sich. Dabei ist entscheidend, welche Anzeige Eigenschaften für das Design Modell benutzt werden (räumliche Darstellung, zeitliche Darstellung, Farben, etc.). Einige Eigenschaften sind dabei von den Daten vorgegeben, bzw. können ausgewählt werden. Nimmt man geographische Daten, welche auf einer Karte dargestellt sind, so ist klar, dass räumliche Daten

vorgegeben sind. Jedoch gibt es auch hier Freiheitsgrade für den Designer. Wird die gesamte Weltkugel dargestellt, so muss entschieden werden, an welcher Position die Weltkugel zur Darstellung aufgeschnitten wird.

	Eigenschaften der Visualisierung		
	Gegeben	**Einschränkungen**	**Ausgewählt**
Kontinuierlich	Bilder (z.B. in der Medizin) Flüsse von Gasen und Flüssigkeiten, Druckverteilungen Molekulare Strukturen (Verteilungen von Masse, Ladungen, etc) Verteilungen rund um den Globus (z.B. Höhenprofil)	Verzerrungen/Veränderungen von den gegeben kontinuierlichen Daten (z.B. 2 dimensionale Landkarten, Fischaugenobjektiv Darstellung) Einteilung/Anordnung der numerischen Werte	kontinuierliche mathematische Funktionen kontinuierliche zeitveränderliche Daten, räumliche Darstellung mit zusätzlicher zeitlicher Information Regressionsanalyse
Diskret	Zugeteilte Daten/Bilder (z.B. Teilbilder in der Medizin) Positionen im Flugverkehr Modekulare Strukturen (exakte Position von Komponenten) diskrete geographische Daten (z.B. Positionen von Städten)	Verzerrungen/Veränderungen von den gegeben kontinuierlichen Daten (z.B. 2 dimensionale Landkarten, Fischaugenobjektiv Darstellung) Einteilung/Anordnung der ordinalen bzw. numerischen Werte	diskrete zeitveränderliche Daten, räumliche Darstellung mit zusätzlicher zeitlicher Information zufälliges Beziehungsdiagramm (z.B. Dateistruktur) zufällige Multidimensionale Daten (z.B. Beschäftigungsstatistik)

Tabelle 2.1: Visualisierungstaxonomie [Tory und Möller [2004]]

Die Tabelle 2.1 hat eine gewisse Verwandtschaft mit der Einteilung von [Shneiderman, 1996]. Dabei ist der Bereich der *Scientific Visualization* tendenziell in der oberen rechten Ecke der Tabelle 2.1 anzufinden. *Information Visualization* sind meist in der unteren linken Ecke der Tabelle 2.1 zu finden. Die Mitte der Tabelle ist mehrdeutig und kann daher entweder der *Scientific Visualization* oder der *Information Visualization* zugeordnet werden, oder aber keiner von beiden. Damit schließt die Tabelle die Lücke, welche bei

der Einteilung von *Scientific Visualization* und *Information Visualization* entstanden ist.

2.3.5 Einteilung der Darstellung für die relevanten Daten dieser Arbeit

Da der vorliegende Datensatz für diese Arbeit in Form von Text vorliegt, welche teilweise hierarchisch aufgebaut ist, fällt die Zuteilung nach [Spence, 2007] und [Shneiderman, 1996] in den Bereich der *Information Visualization*. Diese Beobachtung deckt sich auch mit der Tabelle nach 2.1, in welcher die zu erstellende Visualisierung im rechten unteren Quadranten zu finden sein wird.

2.4 Visual Information Seeking Mantra

Shneiderman [1996] geht ebenfalls davon aus, dass eine abstrakte Informationsvisualisierung ein wichtiger Bestandteil in der aufkommenden Datenflut ist, um Muster, Cluster oder Ausreißer in statistischen Daten zu erkennen. Des Weiteren nimmt Shneiderman [1996] an, dass der Anwender ein viel höheres Auffassungsvermögen hat als es die Darstellungen von 1996 forderten. Obwohl mittlerweile 16 Jahre vergangen sind und sich vieles in diesem Gebiet verändert hat, hat sein von damals aufgestelltes Visualisierungsmotto ”Overview first, zoom and filter, then details-on-demand” nach wie vor Gültigkeit. Beziehungsweise ist es ein guter Ausgangspunkt, um Daten aussagekräftig für den Anwender darzustellen. Ausgehend von sieben unterschiedlichen Datentypen (1-, 2-, 3-dimensionale Daten, zeitabhängige und multi-dimensionale Daten sowie Baumstrukturen oder Netzwerkdaten) führt er sieben Aufgabenbereiche ein, welche auf die Datensätze angewandt werden können.

Overview

Es soll ein gesamter Überblick über den vorhandenen Datensatz erstellt werden. Dabei ist es auch von Vorteil, wenn man in bestimmte Bereiche der Übersicht hinein zoomen kann. Hierfür kann auch eine sogenannte Fischaugen Lupe verwendet werden.

Zoom

Nur einzelne, ausgewählte Bereiche oder Datensätze sollen angezeigt werden. Der Anwender soll die Möglichkeit haben den Zoomfokus sowie den Zoomfaktor selbst

zu bestimmen. Ein stufenloses Zoomen hat den Vorteil, dass der Anwender nicht die Position in der Visualisierung und somit den Kontext aus den Augen verliert. Auch die Steuerung des Zooms darf den Anwender nicht vom eigentlichen Kontext ablenken.

Filter

Nicht erwünschte Daten sollen nach Möglichkeit ausgeblendet werden. D.h. der Anwender soll dynamisch entscheiden können, welche Teile der Datensätze angezeigt bzw. ausgeblendet sind. Damit wird es dem Anwender ermöglicht, die Bereiche genauer anzusehen, welche für ihn von Interesse sind, ohne sich von unerwünschten Informationen abzulenken.

Details on Demand

Datensätze können zu Gruppen zusammengeführt und damit kompakter dargestellt werden. Diese Gruppen können dann vom Anwender wieder aufgelöst werden, um weitere Details zu zeigen. Dies dient ebenfalls dazu, dass der Betrachter sich aus seiner Sicht immer auf das Wesentliche konzentrieren. Weiterführende Informationen können zum Beispiel in Form von Weblinks an den Datensatz angehängt werden.

Relate

Beziehungen zwischen verwandten Datensätzen sollen angezeigt werden. Dabei können Attribute eines Datensatzes zu anderen Datensätzen verlinkt sein. Damit kann eine Navigierbarkeit zwischen Datensätzen entstehen.

History

Da normalerweise die Erforschung von Information für den Anwender ein Prozess von mehreren Iterationsschritten ist, ist es selten, dass Anwender gleich beim ersten Mal alle Schritte richtig ausführen. Damit der Anwender Schritte rückgängig machen können und nicht jedes Mal von vorne beginnen muss, ist es wichtig eine History zu führen.

Extract

Teilergebnisse einer Suche oder Abfrage sollen gespeichert werden können. Von Vorteil wäre, wenn der Datensatz in ein oder mehrere gängige Dateiformatie umgesetzt werden kann, um auch von anderen Programmen genutzt werden zu können.

Untergeordnete Datensätze sollen bei Bedarf extrahiert und damit angezeigt werden. Über Abfrageparameter kann dieser Datensatz eingeschränkt angezeigt werden.

2.5 Text und Dokumenten Visualisierung

Für die Visualisierung von Texten gibt es mittlerweile eine Vielzahl von Daten aus digitalen Bibliotheken, E-Mail Kommunikationen, Dokumentdateien und letztendlich die schier unendliche Anzahl von Webseiten (z.B. blogs, twitter feeds usw.). In Ward et al. [2010] werden drei verschiedene Text Darstellungen unterschieden.

2.5.1 Begriffsdefinitionen

Folgende Begriffsdefinitionen sind aus Ward et al. [2010] entnommen.

Lexikalisch

Hier wird eine Zeichenfolge in ihre kleinsten Teile, auch Token genannt, zerlegt. Dabei können Token einzelne Zeichen oder auch eine Sequenz aus Zeichen sein (z.B. N-Gramm, Wörter, Wortstämme, Phrasen, etc). Es können viele Regeln angewandt werden, um spezifische Tokens zu erhalten. Die bekanntesten Regeln sind über reguläre Ausdrücke definiert und werden mittels Zustandsmaschinen ausgewertet.

Syntaktisch

Auf der syntaktischen Ebene wird versucht jeden Token zu identifizieren. Das heißt, es kann die Position im Satz ermittelt werden, oder ob es sich bei dem Token um ein Hauptwort, Expletiv, Adjektiv usw. handelt. Die Tokens können singular oder plural sein bzw. weitere Eigenschaften wie z.B. Datum, Position, Organisation, Person oder Zeit besitzen. Die Methode, aus einem Satz, die oben aufgezählten Informationen zu erhalten wird im englischen *named entity recognition* (NER) genannt.

Semantisch

Die semantische Ebene beinhaltet die Bedeutung und die Beziehung zwischen den Teilen, welche in der syntaktischen Ebene gefunden wurden. Das Ziel ist eine analytische Interpretation eines vorhandenen Textes, möglicherweise sogar eine Interpretation in einem bestimmten Kontext.

2.5.2 Vektorraummodell

Das Vektorraummodel (engl.: Vector Space Model - VSM) wird als Basis für viele Texte, Dokumenten Visualisierungen und Analyse Techniken verwendet.

In einer sehr einfachen Form des Vektorraummodels werden die verschiedenen Wörter und deren Anzahl pro Absatz oder pro einem oder mehreren Dokumenten in einem Vektor eingetragen. Dabei können sogenannte Stoppwörter ignoriert werden, weil diese die Effizienz der Algorithmen mindern würden und andererseits keinen wesentlichen Beitrag zum Inhalt des Textes beitragen. Stoppwörter können zum Beispiel bestimmte oder unbestimmte Artikel sowie Präpositionen oder Konjunktionen sein also meist Wörter die der Grammatik dienen. Wenn man nun alle Wörter in einem Vektor mit der zugehörigen Anzahl an Vorkommnissen in dem Text ermittelt hat, kann man nach Wörtern, Termen oder zum Beispiel nach n-Grammen gewichten. [Ward et al., 2010]

2.5.3 Visualisierungstypen für einzelne Dokumente

Schlagwortwolke - Tag Cloud

Ein Anwendungsbeispiel des Vektorraummodells ist die sogenannte Schlagwortwolke (engl.: tag cloud). Dabei ist die Häufigkeit, mit welcher die Wörter im Text vorkommen durch z.B. Schriftgröße und Helligkeit kodiert. D.h. Wörter, welche häufig vorkommen, werden größer und heller in der Schlagwortwolke visualisiert als Wörter, welche selten vorkommen. Abbildung 2.5 zeigt eine mögliche Darstellung einer Schlagwortwolke.[Ward et al., 2010]

Wortbäume - Word Tree

Wortbäume (engl.: word tree) Darstellungen werden für unstrukturierten Text verwendet. Dabei kann der Benutzer ein Wort oder eine Phrase als Suchoption eingeben. Kommt dieses Wort oder die Phrase im gegebenen Text vor werden alle nachfolgenden Wörter oder Phrasen der Suchabfrage in einer Baumstruktur dargestellt. Um dieser Baumstruktur zu erhalten, liegt die Methode der Suffix Bäume zu Grunde, welche es schon seit einigen Jahrzehnten gibt. Neu in Wattenberg und Viegas [2008] ist, dass diese Suffix Bäume für unsortierte Texte visualisiert werden. Des Weiteren werden Visualisierungen, wie bei der Tag Cloud verwendet. Das heißt, je öfter ein Wort oder eine Phrase im Text vorkommt, desto größer ist deren Schrift (die Schriftgröße ist proportional zur Wurzel aus der Anzahl

Abbildung 2.5: Beispiel einer Schlagwortwolke - Tag Cloud [http://commons.wikimedia.org/wiki/File:Web_2.0_Map.svg Zuletzt besucht am: 2012.12.03]

der Wörter oder Phrasen). Die Verzweigungen im Word Tree laut Wattenberg und Viegas [2008] werden solange fortgeführt bis eine Phrase vorkommt, welche einzigartig ist. Nach dieser einzigartigen Phrase wird zusätzlich eine fix definierte Anzahl an Zeichen im Zweig des Baumes hinzugefügt. Um grafisch anzuzeigen, welcher Teil des Textes einmalig ist und welcher Teil als zusätzliche Information noch daran gehängt wurde, werden diese beiden Textteile farblich anders dargestellt (ersterer in dunkler Schrift, letzterer in grauer Schrift).

Für die Benutzbarkeit und Interaktivität des Word Trees gibt es eine Webseite IBM Research [2012], welche die beschriebenen Möglichkeiten zeigt. Die Beispielimplementierung wurde mit Java Applets realisiert. Im oberen Bereich kann das Suchwort oder die Suchphrase eingegeben werden. Zusätzlich ist es möglich den Baum nach der Häufigkeit und dem Alphabet zu sortieren. Der Wurzelknoten des Baumes kann links oder rechts ausgerichtet werden. Interessant an der Beispielimplementierung ist die Darstellung bei übermäßig vielen Treffern. Um diese auf einem eingeschränkten Display darzustellen, wie es zum Beispiel bei einem Monitor der Fall ist, wurden zwei Methoden angewandt. Zum Ersten werden Zweige welche sehr viele Blätter besitzen, ab einer gewissen Häufigkeit (z.B. Pixelhöhe von weniger als 3) nicht mehr angezeigt. Statt dessen wurde nur der längste Zweig gesucht und schematisch dargestellt. Damit kann zumindest noch die Information der Struktur des Baumes erhalten werden. Zum Zweiten kann der Baum sehr stark in die Breite wachsen und damit ebenfalls nicht mehr dargestellt werden. Dieser Fall tritt

zum Beispiel ein, wenn ein Artikel als Suchwort eingegeben wird. Auch hier wird wieder über die Häufigkeit und damit letztendlich über die Pixelhöhe selektiert. Oder man definiert eine gewisse Prozentzahl an Zweigen welche angezeigt werden soll (z.B. 1% der am Häufigsten vorkommenden Zweige). Abbildung 2.6 zeigt einen Word Tree für das Wort ”holy, ” in der englisch übersetzten Bibel von King James.

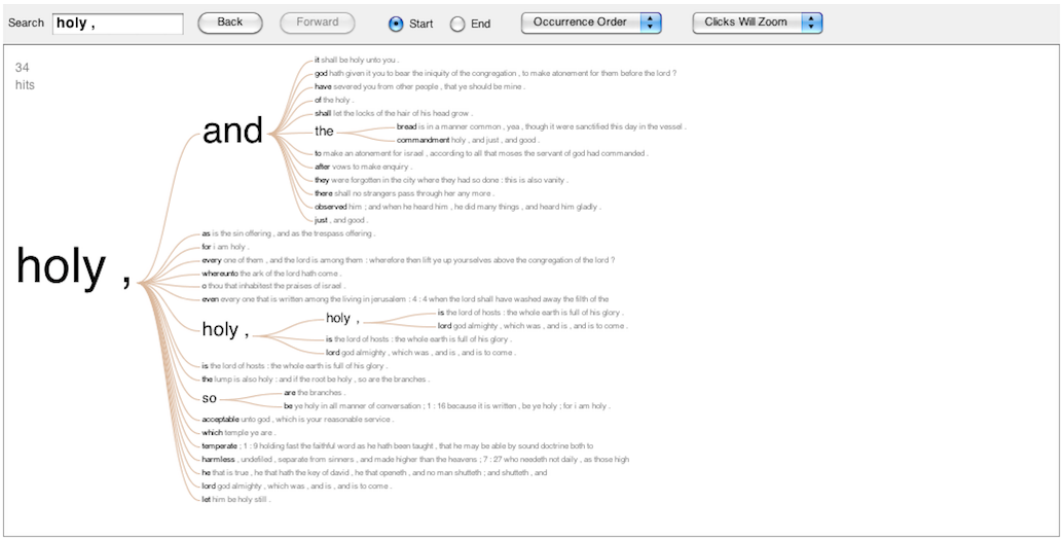


Abbildung 2.6: Beispiel eines Word Trees anhand des Wortes ”holy”, gesucht in der Bibel [http://www-958.ibm.com/software/data/cognos/manyeyes/visualizations/new/word-tree/bible-11/1 Zuletzt besucht am: 2012.10.02)]

TextArc

Mit Hilfe der Visualisierung von TextArc ([Paley, 2002]) ist es möglich mittelgroße, unstrukturierte Texte auf einer Seite darzustellen. Ziel der Darstellung ist es dem Betrachter einen Überblick über den vorhandenen Text zu geben. Dabei wird die lineare Anordnung des Textes nicht verändert. Das heißt im Wesentlichen werden alle Wörter so wie sie im Text vorkommen entlang einer Ellipse im Uhrzeigersinn dargestellt. Stoppwörter werden bei der Darstellung grundsätzlich weggelassen und es werden auch nur die Wortstämme der Wörter angezeigt. Wörter, welche nur einmal im Text vorkommen werden exakt auf der Ellipsenbahn dargestellt, hingegen rücken Wörter, welche zwei- oder mehrmals vorkommen in die Mitte der Ellipse. D.h. das Wort wird nicht mehrmals in der Ellipsenbahn dargestellt, sondern nur einmal im Schwerpunkt aller Vorkommnisse. Damit rücken Wörter die im Text gleichmäßig verteilt sind in die Mitte der Ellipse. Auch in dieser Darstellungsart werden Wörter größer und heller geschrieben, je häufiger diese im Text auf-

treten. Dabei geht man davon aus, dass die Häufigkeit direkt proportional zur Wichtigkeit ist insbesondere da Stoppwörter normalerweise nicht dargestellt werden.

In Abbildung 2.7 wird der analysierte und visualisierte Text ”Alice in Wunderland” von Lewis Carroll dargestellt. Der äußere elliptische Ring zeigt die einzelnen eingelesenen Zeilen in der richtigen Reihenfolge des Werkes. Der innere elliptische Ring zeigt alle Wörter (ebenfalls in der Reihenfolge wie in der Geschichte) die nur einmal vorkommen. Im Inneren werden alle Wörter dargestellt die öfter als einmal vorkommen. Hell geschriebene Wörter kommen öfter in der Geschichte vor. Wenig verwunderlich, ist vielleicht die Tatsache, dass der Name Alice ziemlich genau in der Mitte der Ellipse steht und mit heller Farbe geschrieben ist.

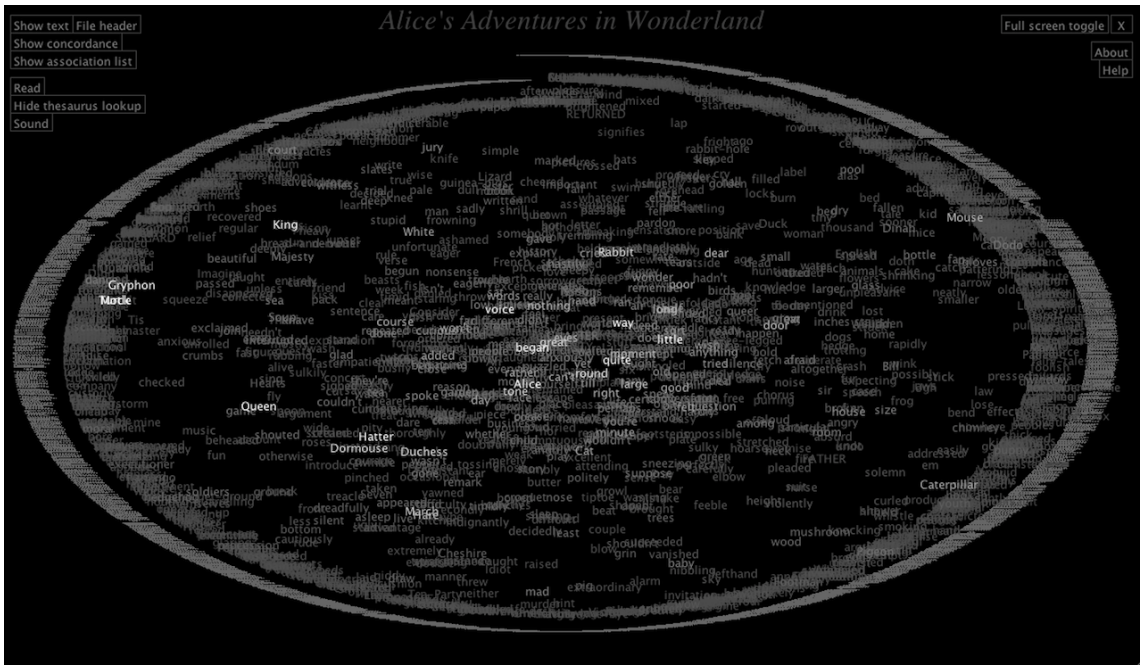


Abbildung 2.7: Beispiel eines TextArc anhand vom Text ”Alice im Wunderland” [http://www.textarc.org Zuletzt besucht am: 2012.09.25]

2.6 Visualisierungsarten für Bäume, Graphen und Netzwerke

2.6.1 Allgemein

In diesem Abschnitt geht es um Entitäten, die miteinander verknüpft sind bzw. eine Verbindung oder Verwandtschaft miteinander aufweisen. Dabei kann es sich um Verbin-

dungen von Netzwerken, oder um hierarchische Verbindungen (Eltern-Kind-Beziehung bzw. Mengen und deren Untermengen) handeln. Auch Ähnlichkeiten der Werte oder andere Eigenschaften in den Datensätzen können Beziehungen darstellen. Die Verbindungen können gerichtet oder ungerichtet, gewichtet oder ungewichtet sein. Laut Battista et al. [1999] sind diese Entitäten und deren Beziehungen in den Computerwissenschaften, zum Beispiel als UML-Diagramme oder zur Darstellung von Computernetzwerken, allgegenwärtig. Aber nicht nur in der Informatik sondern auch in vielen anderen Anwendungsbereichen lassen sich solche Abhängigkeiten der Entitäten finden. So werden Schaltpläne und deren zugehörigen Platinen Layouts als Graphen interpretiert. Herman et al. [2000] führt Graphenbeispiele in biochemischen Reaktionswegen, Evolutionsbäumen sowie genetischen und molekularen Darstellungen an. Graphendarstellungen lassen sich aber auch in Projektmanagementtools, sowie mittlerweile weit verbreitet, in Navigationssytemen finden.

Wie in Battista et al. [1999] aber auch in vielen anderen Fachliteraturen angeführt wird, ist die Visualisierung von Beziehungen und deren Strukturen nur dann sinnvoll, wenn damit Information an den Betrachter übermittelt werden kann. Das heißt ein gutes Diagramm hilft dem Betrachter das System zu verstehen, ein schlechtes Diagramm wird den Betrachter eher verwirren und möglicherweise dazu verleiten falsche Schlüsse aus der Visualisierung zu ziehen.

2.6.2 Allgemeine Graphen Definitionen

Folgende Definitionen in Bezug auf Graphen wurden aus Battista et al. [1999] und [Ward et al., 2010] entnommen. Ein *Graph* G = (V, E) besteht aus einer endlichen Anzahl an Knoten V (engl.: vertices) und einer endlichen Sammlung an Kanten E (engl.: edges). Kanten haben immer den Ursprung und das Ende in einem Knoten. Die Enden einer Kante e werden mit u und v benannt, daraus ergibt sich, dass u und v immer angrenzend oder benachbart (engl.: adjacent) zueinander sind, sowie die Kante e immer einfallend (engl.: incident) zu u und v bezeichnet wird. Die Nachbarn von v sind die angrenzenden Knoten. Zwei Kanten heißen ebenfalls benachbart, wenn diese über einen gemeinsamen Knoten verbunden sind.

Der *Grad* von v ist die Anzahl der Nachbarn.

Eine Kante (u, v) bei der $u = v$ ist, wird als Schleife (engl.: self-loop) bezeichnet. Das heißt, der Ausgangsknoten und Endknoten ist derselbe. Eine Kante die mehrmals in E vorkommt wird im Englischen auch ”multiple edge” bezeichnet.

Ein *gerichteter Graph* (engl.: directed graph) ist ein Graph mit *gerichteten Kanten*. Unter einer gerichteten Kante (engl.: directed edge) versteht man eine Kante die von einem Knoten ausgeht und in einem anderen Knoten mündet. Bei gerichteten Kanten wird ein Pfeil in den einmündeten Knoten eingezeichnet.

Knoten, welche nur eingehende Kanten besitzen werden als *Senken* (engl.: sinks) bezeichnet bzw. Knoten, welchen nur ausgehende Kanten besitzen werden als *Quellen* (engl.: sources) bezeichnet. Es gibt auch Knoten die eingehende und ausgehende Verbindungen besitzen. Diese Knoten können mit dem Grad, der Anzahl an eingehenden (engl.: indegree) bzw. ausgehenden (engl.: outdegree) Kanten, bestimmt werden.

Vollständiger Graph (engl.: connected graph) ist ein Graph in dem jeder Knoten mit jedem anderen Knoten über eine Kante miteinander verbunden ist. Als Beispiel siehe Abbildung 2.15b.

Schnittknoten (engl.: cutvertex) ist ein Knoten, in einem zusammenhängenden Graphen, welcher nicht mehr zusammenhängend ist, wenn der Knoten entfernt wird.

Zweizusammenhängender Graph (engl.: biconnected graph) ist ein Graph, indem es keine Schnittknoten gibt.

Ein *Block (engl.: block)* ist ein maximaler zweizusammenhängender Teilgraph eines Graphen.

Ein *Seperationspaar (engl.: separating pair)* ist ein Paar aus zwei Knoten, welche einen zweizusammenhängenden Graphen trennen, falls dieses entfernt wird.

Dreizusammenhängender Graph (engl.: triconnected graph) ist ein Graph in dem es kein Seperationspaar gibt.

Ein *gerichteter Weg* in einem gerichtetem Graphen ist eine Sequenz aus bestimmten Knoten $(v_1, v_2, v_3, ..., v_h)$ von G. Für das Paar $(v_i, v_{i+1}) \in E$ muss gelten $1 \leq i \leq h-1$.

Bei einem *ungerichteten Weg* in einem ebenfalls ungerichteten Graphen muss für die Menge $\{v_i, v_{i+1}\} \in E$ ebenfalls $1 \leq i \leq h-1$ gelten. Damit stellt ein Weg eine endliche Menge durch Kanten verbundener Knoten dar.

Ein (gerichteter) Weg ist ein *(gerichteter) Zyklus*, wenn $(v_h, v_1) \in E$.

Ein gerichteter Graph ist *azyklisch*, wenn er keine gerichteten Zyklen besitzt.

Ein unverzweigter Weg wird *Pfad* genannt.

Eine *transitive Kante* (u, v) in einem gerichteten Graphen, liegt vor wenn ein direkter Weg von u nach v existiert, welcher nicht die Kante (u, v) beinhaltet.

Ein Graph, welcher für jeden Weg von u nach v eine Kante (u, v) besitzt wird auch

transitiver Graph bezeichnet.

Eine *transitive Hülle* (engl.: transitive closure) eines gerichteten Graphen hat eine Kante (u, v) für jeden Weg von u nach v im Graphen.

In vielen Anwendungen ergibt ein gerichteter Graph die gleiche Information wie die transitive Hülle des gleichen Graphen. Als Beispiel können hierfür Klassenhierarchien (siehe Abbildung 2.8) in objektorientierten Computerprogrammen angegeben werden. Oder aber auch die Hierarchie in einer Organisation. Wenn Herr ”X” der Vorgesetzte von Herrn ”Y” ist und Herr ”Y” der Vorgesetzte von Herrn ”Z” ist, ergibt sich indirekt dass Herr ”X” auch der Vorgesetzte von Herrn ”Z” ist. Wie in Abbildung 2.8 ersichtlich ist, gibt es ein Durcheinander von Pfeilen, welche nicht unbedingt zur Lesbarkeit des Graphen dienen. In Abbildung 2.9 wird der gerichtete Graph ohne transitive Hülle dargestellt. Diese Darstellung ist viel leichter lesbar und trägt zum Verständnis des Inhaltes bei.

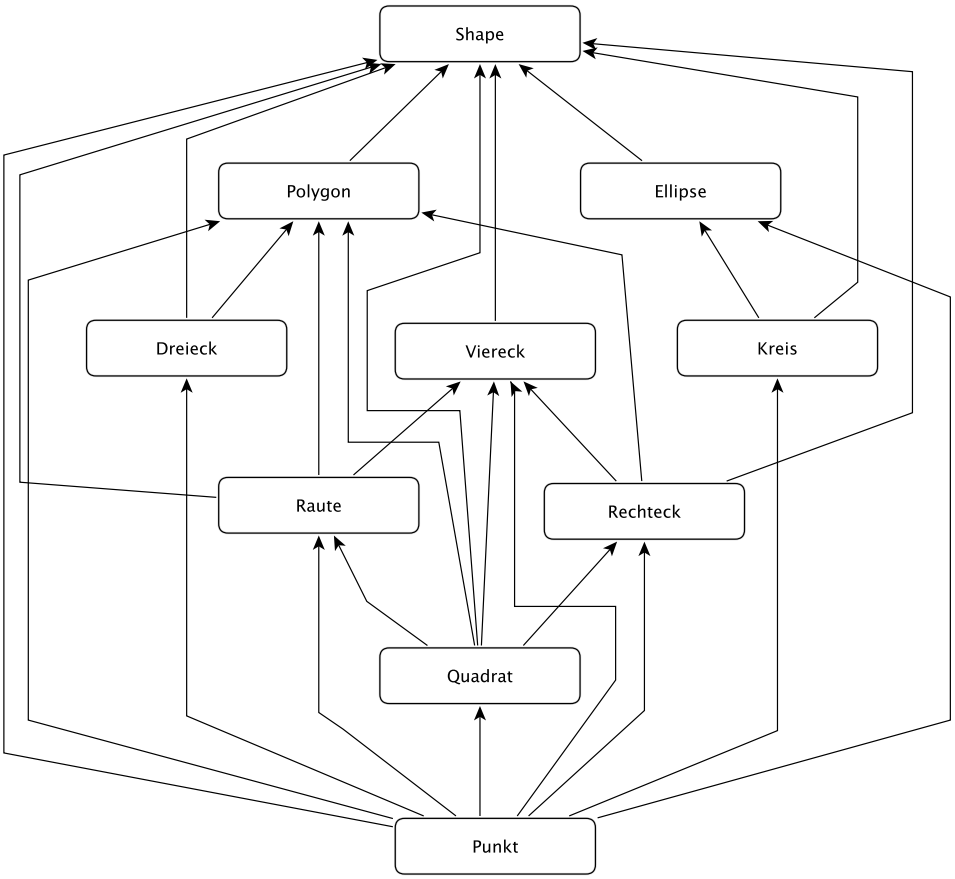


Abbildung 2.8: Transitive Hülle [Battista et al. [1999, Seite 5]]

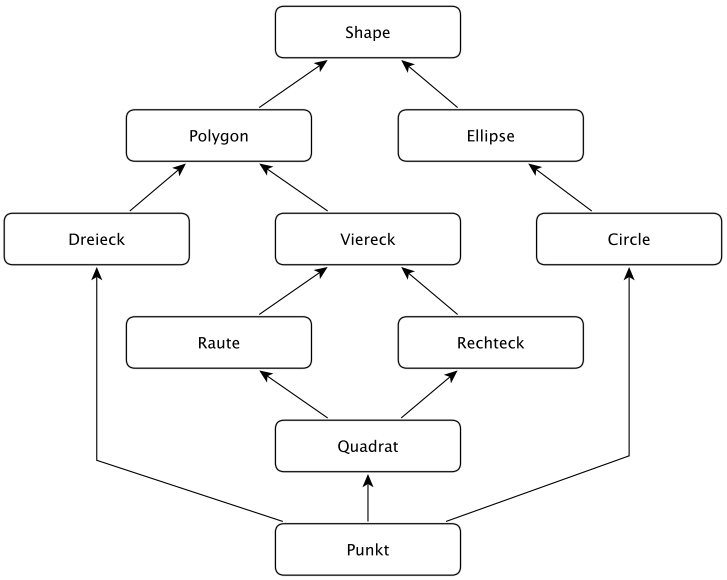


Abbildung 2.9: Gerichteter Graph [Battista et al. [1999, Seite 5]]

2.6.3 Adjazenzmatrix und Adjazenzliste

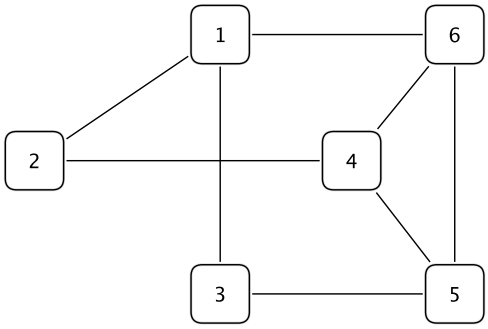
Graphen mit n Knoten können auch als $(n \times n)$ Adjazenzmatrizen A (auch als Nachbarschaftsmatrizen bekannt) dargestellt werden. Dabei werden in den Reihen und Zeilen der Matrix die Knoten von 1 bis n eingetragen.

In die Matrix wird eine 1 eingetragen, wenn die beiden Knoten miteinander benachbart sind bzw. eine 0 wenn keine direkte Verbindung zwischen den Knoten besteht. Allgemein kann geschrieben werden, dass $A_{uv} = 1$ wenn $(u, v) \in E$ ansonsten $A_{uv} = 0$. Bei gerichteten Graphen wird eine 1 in die Matrix eingetragen, wenn der Knoten u, eingetragen in der Zeile der Matrix, Vorgänger des Knotens v, eingetragen in der Spalte der Matrix, ist. Abbildung 2.11 zeigt eine Adjazenzmatrix und deren zugehörigen ungerichteten Graphen.

Als Alternative zur Adjazenzmatrix kann auch eine Adjazenzliste L verwendet werden. Dabei wird für jeden Knoten eine Liste all seiner Nachbarn angelegt. Bei gerichteten Graphen beinhaltet die Liste nur die Knoten der Nachfolger. Dabei kann die Liste auch weitere Eigenschaften wie zum Beispiel die Kanten- oder Knotengewichte beinhalten. Abbildung 2.12 zeigt eine Adjazenzliste und deren zugehörigen gerichteten Graphen. ([Cormen et al., 2009])

	1	*2*	*3*	*4*	*5*	*6*
1	0	1	1	0	0	1
2	1	0	0	1	0	0
3	1	0	0	0	1	0
4	0	1	0	0	1	1
5	0	0	1	1	0	1
6	1	0	0	1	1	0

(a) Adjazenzmatrix

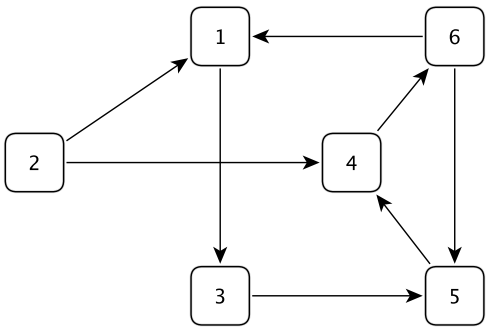


(b) Graph korrespondierend zur Adjazenzmatrix 2.10a

Abbildung 2.10: Adjazenzmatrix und korrespondierendem ungerichtetem Graph

	1	*2*	*3*	*4*	*5*	*6*
1	0	0	1	0	0	0
2	1	0	0	1	0	0
3	0	0	0	0	1	0
4	0	0	0	0	0	1
5	0	0	0	1	0	0
6	1	0	0	0	1	0

(a) Adjazenzmatrix



(b) Graph korrespondierend zur Adjazenzmatrix 2.11a

Abbildung 2.11: Adjazenzmatrix und korrespondierendem gerichtetem Graph

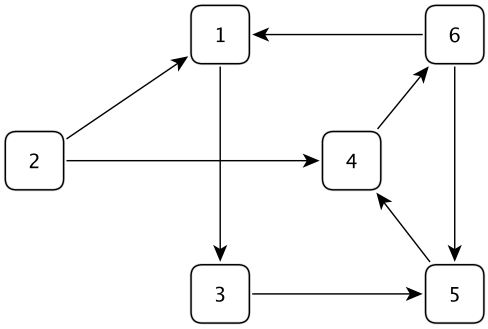
2.6.4 Planare Graphen

Planare Graphen sind Darstellungen von Graphen, welche alle Knoten und Kanten auf einer Ebene darstellen, ohne dass sich die Kanten auf dieser Ebene kreuzen. Für die Darstellung von Graphen sind planare Graphen von Wichtigkeit, da sie leichter zu verstehen bzw. zu lesen sind, als Graphen deren Kanten sich kreuzen.

Die planare Zeichnung eines Graphen unterteilt die Ebene in topologisch verbundenen Regionen, welche Flächen (engl. faces) genannt werden. Diese Flächen werden durch die Reihenfolge ihrer adjazenten Kanten beschrieben. Mit welcher Kante die Reihenfolge beginnt ist für die Definition der Fläche nicht wesentlich. Die Fläche, welche durch keine Kanten nach außen begrenzt wird, wird als Außenfläche (engl. external face) bezeich-

L_1	(1,3)
L_2	(2,1) , (2,4)
L_3	(3,5)
L_4	(4,6)
L_5	(5,4)
L_6	(6,1) , (6,5)

(a) Adjazenzliste



(b) Graph korrespondierend zur Adjazenzliste 2.12a

Abbildung 2.12: Adjazenzliste und korrespondierendem gerichtetem Graph

net. Eine Einbettung eines Graphen in der Ebene führt zu einer $zyklischen\ Ordnung$ des Graphen. Das heißt, für jeden Knoten wird eine Reihenfolge (z.B. im Uhrzeigersinn) der einfallenden (engl. incident) Kanten festgelegt. Zwei planare Zeichnungen des gleichen Graphen werden als äquivalent bezeichnet, wenn deren Reihenfolge und damit die zyklische Ordnung gleich ist. Eine $Einbettung$ (engl. embedding) ist eine Äquivalenzklasse von planaren Zeichnungen, welche durch die $zyklischen\ Ordnung$ festgelegt ist. Ein $eingebetteterGraph$ (engl.: embedded Graph) ist ein Graph mit einer bestimmten Einbettung. Anhand dieser Definitionen kann ein Graph exponentiell viele verschiedene Einbettungen besitzen.

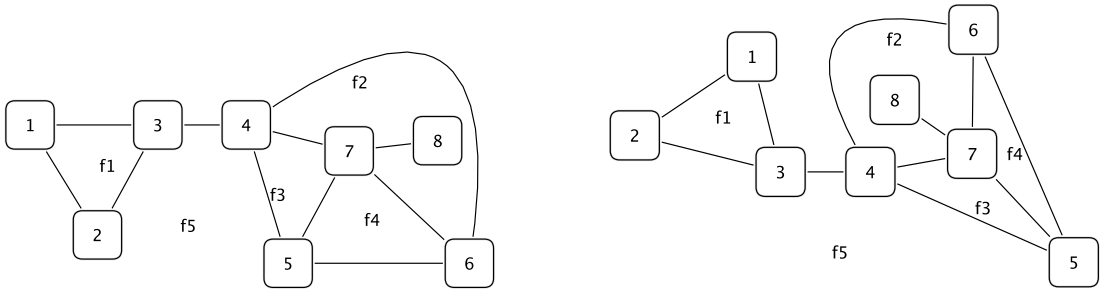


Abbildung 2.13: Eingebetteter Graph 1 [Battista et al. [1999, Seite 7]]

Die Abbildungen 2.13a und 2.13b zeigen den gleichen eingebetteten Graphen, da die zyklische Ordnung beider Darstellungen ident ist. Die Flächen, welche sich durch die Kanten ergeben, sind mit f1 bis f5 bezeichnet. Die Fläche f5 ist die Außenfläche der Darstellung. Anders verhält es bei Darstellung desselben Graphen in Abbildung 2.14.

Hier wird die Ebene ebenfalls in fünf Flächen unterteilt, jedoch begrenzen die einzelnen Flächen nicht zwangsweise dieselben Kanten wie in Abbildung 2.13. Das heißt die zyklische Ordnung in 2.13 entspricht nicht der zyklischen Ordnung von 2.14. *Damit legt die Einbettung, welche wie oben erwähnt exponentiell oft vorkommen kann, die Gestalt der Zeichnung fest.* ([Battista et al., 1999])

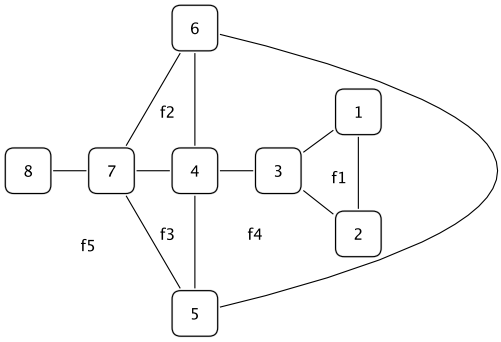


Abbildung 2.14: Eingebetteter Graph 2 [Battista et al. [1999, Seite 7])]

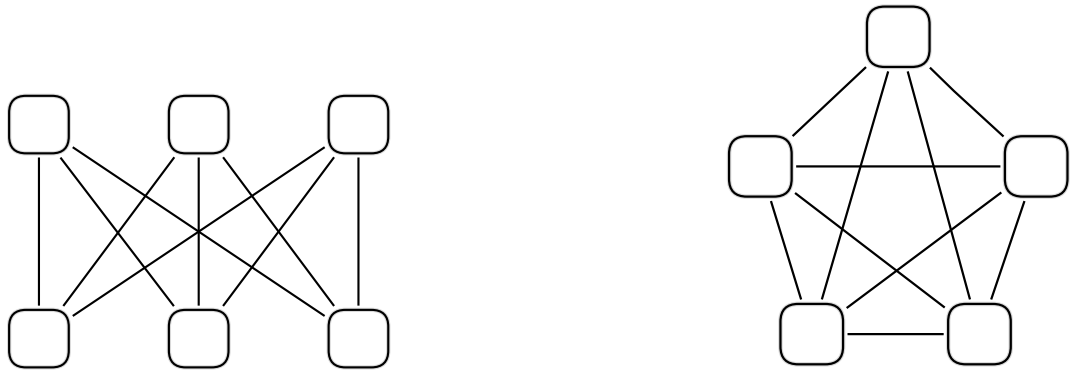
2.6.5 Eulersche Polyederformel

In diesem Zusammenhang sei auch die Eulersche Polyederformel erwähnt. Die Formel sagt aus, dass ein kreuzungsfrei gezeichneter zusammenhängender schlichter Graph mit n Knoten und m Kanten die Ebene in $m - n + 2$ Gebiete zerlegt. Als Beweis für die Eulersche Polyederformel kann man von einem Graphen mit nur einem Knoten und keiner Kante ausgehen. Dieser Graph erfüllt die Eulersche Polyederformel. Ausgehend von diesem Graphen können zwei Operationen ausgeführt werden, die den Graphen erweitern, aber nicht die Polyederformel verletzen. Zum einen kann eine Kante zusammen mit einem Knoten hinzugefügt werden ohne die Polyederformel zu verletzen, da sich die Gesamtanzahl der Knoten sowie die Gesamtanzahl der Kanten um eins erhöht, bleibt die Differenz der beiden gleich. Es kann sich durch hinzufügen der Kante und des Knotens kein neues Gebiet bilden. Zum anderen kann eine Kante hinzugefügt werden um zwei bestehenden Knoten zu verbinden. Damit ändert sich die Gesamtzahl der Kanten, nicht aber die Gesamtzahl der Knoten. Jedoch ändert sich durch das Verbinden zweier bestehender Knoten die Gesamtanzahl der Gebiete um eins und damit stimmt die Eulersche Polyederformel wieder. Die Gültigkeit der Formel kann auch anhand der Abbildungen 2.13 und 2.14 beobachtet werden. [Eppstein, 2005]

2.6.6 Kuratowski Graphen

Der Vollständigkeit in Bezug auf planare Graphen sei hier der Satz von Kuratowski angeführt. Dieser besagt, dass ein Graph planar ist, wenn er keinen Teilgraphen enthält, der durch Unterteilung von K_5 oder $K_{3,3}$ entstanden ist.

Der Graph $K_{3,3}$ ist ein vollständiger bipartiter Graph. Ein bipartiter Graph besitzt zwei disjunkte Teilmengen. Innerhalb der Teilmengen gibt es keine Verbindungen untereinander. Der Graph K_5 ist ein vollständiger Graph mit fünf Knoten. Abbildung 2.15 zeigt die beiden Graphen, welche die kleinsten nicht planaren Graphen sind. Unter Unterteilung eines Graphen G versteht man, dass man bei gewissen Kanten neue Ecken einfügt, indem man Knoten mit dem Grad von 2 einfügt. Durch Unterteilung ändert sich die Eigenschaft des Graphen planar oder nicht planar zu sein, nicht. [Kuratowski, 1930]



(a) $K_{3,3}$ **(b)** K_5

Abbildung 2.15: Kuratowski Graphen

2.6.7 Parameter für Graphenvisualisierungsmethoden

In Battista et al. [1999] wird versucht mit oben genannten Definitionen die Graphen bestimmten Klassen zuzuteilen. Vor allem auch deshalb, wenn es darum geht, einen geeigneten Algorithmus für das Zeichnen eines Graphen zu verwenden. Wie oben erläutert können Graphen azyklisch, oder planar sein oder aber auch eine Baumstruktur aufweisen. Wenn man die Graphen in derartige Klassen einteilt, ist es leichter einen geeigneten Algorithmus zur Darstellung zu finden. Oder man möchte mit der Darstellung die kombinatorischen Eigenschaften des Graphen hervorheben. Hierzu ist es auch nötig zu wissen, ob zum Beispiel ein Graph azyklisch und gerichtet ist, damit auch die Darstellung diese

Eigenschaften hervorheben kann. So wäre im Falle eines gerichteten azyklischen Graphen eine Darstellung möglich, in der alle gerichteten Kanten in einer Richtung gezeichnet werden, um so dem Betrachter schnellstmöglich zu zeigen, dass der Graph keine Schleifen beinhaltet.

Andererseits gibt es keine perfekte Darstellung, da auch die Wahrnehmung des Menschen mitentscheidend ist. Und selbst die Wahrnehmung kann von Individuum zu Individuum variieren. Ein weiterer Faktor ist, in welchem Kontext die Darstellung erfolgen soll. Auch hier kann die Darstellung je nach Anwendungsfall variieren. Um diesen Anforderungen gerecht zu werden gibt es lt. Battista et al. [1999] drei Konzepte. Im Folgenden werden die *Zeichenkonventionen*, die *ästhetischen Aspekte* und das *Konzept der Einschränkung* näher erläutert. Alle drei Konzepte müssen bei der Darstellung in Einklang gebracht werden, obwohl sich einige dieser wiedersprechen. Zum Beispiel führt Purchase [1997] in ihrer Studie an, dass die Minimierung der Kantenkreuzungen meist auch die Symmetrie der Darstellung verschlechtert. Ein weiterer Punkt für die Darstellung ist die Effizienz mit der diese Graphen berechnet werden können. Die führt letztlich dazu, wie schnell eine Graphenvisualisierung berechnet und für den Anwender dargestellt werden kann.

Zeichenkonventionen

Primär geht es bei den Zeichenkonventionen darum, wie die Kanten zwischen den Knoten gezeichnet werden. Diese können geradlinig, rechtwinkelig oder mit beliebig vielen Winkel gezeichnet werden. Bei rechtwinkelig gezeichneten Kanten, werden diese meist senkrecht bzw. waagrecht angeordnet. Bei Linienzügen mit beliebigen Winkeln muss man darauf achten, nicht zu viele Winkel darzustellen, da ansonsten der Betrachter dem Linienzug nicht mehr folgen kann. Der Graph kann auch auf einem Netzgitter (engl.: grid) liegen. Dazu sind die Koordinaten der einzelnen Komponenten des Graphen als Integer abgespeichert. Bei einem azyklischen gerichteten Graphen kann man versuchen alle Kanten nach oben oder nach unten zu zeichnen (engl.: upward drawing, downward drawing).

Ästhetische Aspekte

Ästhetische Aspekte sind sehr wichtig in Bezug auf die Interpretier- und Lesbarkeit der Darstellung. Die ästhetischen Aspekte beinhalten die maximale Anzahl von Kantenkreuzungen. Da es sich nicht immer um planare Graphen handelt, ist man zumindest versucht nur eine minimale Anzahl an Kantenkreuzungen darstellen zu müssen. Ebenso kann die

Zeichenfläche zur Ästhetik gehören. Nämlich dann, wenn der Platz begrenzt ist und der Graph kleiner skaliert werden muss. Die zur Verfügung stehende Zeichenfläche kann formal als kleinste konvexe Hülle, oder als kleinstes Rechteck, welches den Graphen beinhaltet definiert werden.

Die Gesamtlänge der Kanten sowie die maximale Länge der Kanten oder eine einheitliche Länge für alle gezeichneten Kanten trägt zur Ästhetik bei. Ebenso kann man die Anzahl der Knicke bei den Kanten vereinheitlichen oder versuchen die Gesamtanzahl auf ein Minimum zu reduzieren. Des Weiteren können das Bildformat sowie die Symmetrie zur Lesbar- und Verständlichkeit beitragen.

Konzept der Einschränkungen

Ganz allgemein sei hier erwähnt, dass sich Einschränkungen in Bezug auf die Positionierung der Knoten ergeben können. Zum Beispiel will man einen bestimmten Knoten in die Mitte der Darstellung setzen, oder an den Rand. Es ist auch möglich Knoten, welche über bestimmte Eigenschaften Ähnlichkeiten zueinander aufweisen nebeneinander darzustellen (engl.: clustering). Um gewisse Sequenzen hervorzuheben soll ein bestimmter Pfad horizontal bzw. vertikal dargestellt werden.

Untersuchung der Parameter

Purchase [1997] untersucht 5 ästhetische Eigenschaften in Bezug auf die Lesbarkeit des Graphen. In ihrer empirischen Studie untersucht sie, wie effektiv und mit welcher Genauigkeit der Betrachter verschiedene Darstellungen desselben Graphen lesen kann und stellt dabei folgende Aufgaben an den Beobachter:

- Bestimme den kürzesten Pfad zwischen zwei gegebenen Knoten.
- Bestimme das Minimum an Knoten, welche entfernt werden müssen um den Pfad zwischen zwei bestimmten Knoten zu entfernen.
- Bestimme das Minimum an Kanten, welche entfernt werden müssen um den Pfad zwischen zwei bestimmten Knoten zu entfernen.

Die Graphendarstellung variiert in 5 verschiedenen ästhetischen Darstellungen. Für jeden dieser ästhetischen Darstellungen stellt Purchase [1997] eine Hypothese auf, welche sich aufgrund ihrer Auswertungen nicht alle bewahrheiten. Nachfolgend sind ihre ästhetischen Darstellungen, die aufgestellten Hypothesen und deren Auswertungen beschrieben. Die

Auswertungen beziehen sich auf die Richtigkeit der Antworten und auch auf die Schnelligkeit mit der diese gegeben wurden.

Ästhetik	**Hypothese**	**Ergebnis**
Kantenwinkel	Erhöhung der Kantenwinkel führt zu einer Minderung der Lesbarkeit der Darstellung	Der empirische Test zeigt wenig Übereinstimmung mit der Behauptung
Kantenkreuzungen	Erhöhung der Kantenkreuzungen führt zu einer Minderung der Lesbarkeit der Darstellung	Der empirische Test zeigt große Übereinstimmung mit der Behauptung
Kantenaustrittswinkel	Minimierung des Austrittswinkels aus den Knoten für Kanten erhöht die Lesbarkeit der Darstellung	Der empirische Test zeigt wenig Übereinstimmung mit der Behauptung
Orthogonalität	Kanten und Knoten werden auf ein orthogonales Netzgitter gelegt und erhöhen so die Lesbarkeit der Darstellung	Der empirische Test zeigt nur sehr wenig Übereinstimmung mit der Behauptung
Symmetrie	Erhöhung der Symmetrie in der Darstellung erhöht die Lesbarkeit der Darstellung	Der empirische Test zeigt nur sehr wenig Übereinstimmung mit der Behauptung

Tabelle 2.2: Empirische Graphenästhetik und Analyse nach [Purchase [1997]]

Das Ergebnis macht deutlich, dass die Ästhetik bei der Graphenvisualisierung nicht vernachlässigbar ist, zum anderen große Unterschiede in der Lesbarkeit verursachen.

2.6.8 Übersicht über Graphenvisualisierungsmethoden

Die folgenden Methoden ([Battista et al., 1999]) geben einen Auszug an Möglichkeiten wieder, welche Methoden und Algorithmen auf Graphen und Graphenklassen zur Visualisierung angewandt werden können. Eine Graphenklasse ist eine bestimmte Art von Graph wie zum Beispiel ein zweizusammenhängender Graph. Kann man bestimmte Methoden auf diese Graphenklasse anwenden so kann man die gleichen Methoden auch auf die Unterklassen dieser Klasse anwenden. Im Falle der zweizusammenhängenden Graphen wären die dreizusammenhängenden Graphen eine Unterklasse. Damit können alle Methoden welche für zweizusammenhängende Graphen geeignet sind, auch für dreizusammenhängende Graphen angewendet werden.

Topology-Shape-Metric Ansatz

Diese Methode beinhaltet drei hierarchische Schritte. Im ersten Schritt (engl.: planarization step) wird versucht Kreuzungen in der Zeichnung des Graphen zu vermeiden. Ist es nicht möglich einen planaren Graphen zu zeichnen, wird ein maximaler planarer Subgraph extrahiert. Danach werden sukzessive diejenigen Kanten hinzugefügt, welche eine Kreuzung verursachen. Am Kreuzungspunkt wird ein Hilfsknoten eingefügt, um den wachsenden Subgraphen und letztendlich den fertigen Graphen planar zu gestalten. Als zweiter Schritt (engl.:orthogonalization step) werden die Kanten rechtwinkelig gezeichnet. In diesem Schritt haben die Knoten und Kanten keine zugeordneten Koordinaten. Erst im dritten Schritt (engl. compaction step) werden den Komponenten des Graphen Koordinaten zugeordnet. Hier liegt die Schwierigkeit meist im Minimalisieren der Darstellung. Die hierarchische Dreistufen-Methode legt auch implizit die Prioritäten der Ästhetik fest. Weil zuerst versucht wird den Graphen planar darzustellen wird auch dem Entflechten von Kreuzungen mehr Gewicht gegeben als zum Beispiel die Anzahl der Knicke in den Kanten. Ästhetische Aspekte im dritten Schritt (Minimierung der Kantenlängen) sind ebenfalls durch die beiden vorhergehenden Schritte stärker eingeschränkt. Auch bei den Einschränkungen ergibt sich durch die Ausführungshierarchie der Schritte eine Prioritätenhierarchie. Im ersten Schritt kann beispielhaft festgelegt werden, dass bestimmte Kanten keine Kreuzungen haben sollen. Im zweiten Schritt kann dann für bestimmte Kanten gefordert werden, dass diese nicht mit Winkeln dargestellt werden sollen. Und im letzten Schritt können bestimmte Kanten länger oder kürzer gezeichnet werden. (Tamassia et al. [1988])

Hierarchischer Ansatz

Der hierarchische Ansatz eignet sich besonders für gerichtete azyklische Graphen. Eine Hierarchie nach Warfield [1977] und Sugiyama et al. [1981] besteht aus Knoten und gerichteten Kanten, welche folgende Definitionen unterliegen.

Die *Weite* (engl.: width) der Hierarchie ist die höchste Anzahl an Knoten in einem Level und das *Maß* (engl.: measure) der Hierarchie gibt die Anzahl der Kanten an.

Die Anzahl der Levels in einer Hierarchie wird als *Länge* (engl.: length) bezeichnet.

Die *Spannweite* (engl.: span) einer Kante wird als die Differenz der Level, in dem die Kante startet bis zum Ende der Kante definiert.

Die Knoten werden in Levels eingeteilt. Unter einem Level versteht man eine Teilmenge der gesamten Knoten. $V = V_1 \cup V_2 \cup \ldots \cup V_n$ mit $(V_i \cap V_j = 0, i \neq j)$. Die Indizes der einzelnen Knotenteilmengen beziehen sich auf die verschiedenen Levels in der Hierarchie (zum Beispiel: V_i ist die Knotenteilmenge vom i-ten Level).

Die Kanten sind immer von der höheren Hierarchie zur niedrigeren gerichtet (von der niedrigern zur höheren in [Sugiyama et al., 1981]). D.h. aufgrund der Knotenanordnung werden die Kanten immer vertikal von einem Level zum nächsten gezeichnet. Dabei gilt für jede Kante. $e = (v_i, v_j) \in E$ wobei $v_i \in V_i$ und $v_j \in V_j$ und $i > j$ und jede Kante in E einzigartig ist.

Die Kanten sind ebenfalls in Teilmengen abhängig von der Anzahl der Levels unterteilt ($n - 1$ Teilmengen). Dabei sind die Teilmengen mit $E = E_1 \cup E_2 \cup \ldots \cup E_{n-1}$ mit $(E_i \cap E_j = 0, i \neq j)$ und $E_i \subset V_i \times V_{i+1}, i = 1, \ldots, n-1$ definiert.

In [Warfield, 1977] werden die Knoten in einer Ebene nach dem Vorkommen angeordnet. Nach [Sugiyama et al., 1981] kann explixit eine Ordnung angegeben werden. Wobei sich die Ordnung $\sigma_i = v_1 v_2 \ldots v_{|V_i|}$ ($\mid V_i \mid$ entspricht der Anzahl der Knoten in einem Level) immer auf einen Level bezieht.

Die oben angeführten Definitionen inkludieren eine ordnungsgemäße (engl.: proper) Hierarchie. Einer ordnungsgemäße Hierarchie liegt dann vor, wenn die Spannweite aller Kanten gleich 1 ist. Um diese aus einem gerichteten azyklischen Graphen zu erreichen, kann es vorkommen, dass Hilfsknoten (engl.: dummy vertices) eingeführt werden müssen. Abb 2.16 zeigt eine ordnungsgemäße Hierarchie mit Hilfsknoten, welche mit schwarzer Füllfarbe gekennzeichnet sind.

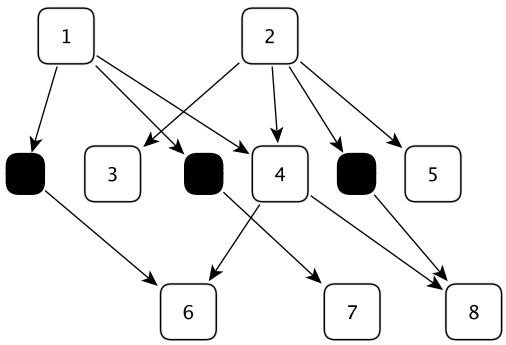


Abbildung 2.16: Hierarchischer Ansatz Teil 1[Battista et al. [1999, Seite 23]]

Liegt eine ordnungsgemäße Hierarchie vor kann diese auch als ordnungsgemäße Matrize dargestellt werden. Mit Hilfe von Algorithmen von [Sugiyama et al., 1981] und

[Warfield, 1977] kann aus der ordnungsgemäßen Matrize die Kantenüberschneidungen ausgerechnet werden. Durch die hierarchische Anordnung der Knoten und der Kantenspannweite von 1 müssen nur die benachbarten Zeilen auf Überschneidungen untersucht werden. Kantenüberschneidungen können durch vertauschen der Knoten in einer Zeile der Matrix minimiert werden. Abbildung 2.17 zeigt die gleiche Hierarchie wie in Abbildung 2.17, jedoch ohne Kantenüberschneidungen. Dies wurde durch horizontales Verschieben von Knoten einer bestimmten Ebene erreicht.

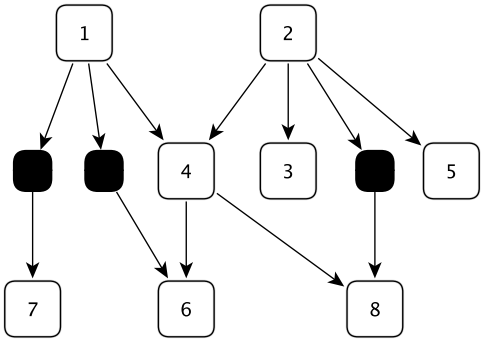


Abbildung 2.17: Hierarchischer Ansatz Teil 2 [Battista et al. [1999, Seite 23]]

Wurde ein Minimum an Überschneidungen gefunden können noch weitere Aspekte der Lesbarkeit berücksichtigt werden. So können alle Kanten geradlinig von einem Knoten zum nächsten gezeichnet werden. Werden danach, die anfangs eingefügten Hilfsknoten entfernt, kann sich für diese längeren Kanten ein polygoner Linienzug ergeben. Abbildung 2.18 zeigt nach dem Entfernen der Hilfsknoten, die geradlinigen Kantenverbindungen von Knoten 1 zu den Knoten 6 und 7. Die Kantenverbindung von Knoten 2 auf Knoten 8 wurde mit einem polygonen Linienzug gezeichnet. Weitere ästhetische Aspekte können erreicht werden, wenn die Knoten entlang der x-Achse verschoben werden, um zum Beispiel gewisse Symmetrien hervorzuheben.

Force-Directed Ansatz

Beim Force-Directed Ansatz geht es darum, dass ein physikalisches Modell als Vorbild für die Darstellung des Graphen dient. In der Literatur sind mehrere Beispiele mit unterschiedlichen physikalischen Modellen zu finden und einige werden hier angeführt. Generell kann man nach [Battista et al., 1999] den Ansatz in zwei Teile zerlegen.

Zum einen besteht der Ansatz aus einem Modell in welchem die Knoten und Kanten, einem Kraft- oder Energiesystem ausgesetzt sind.

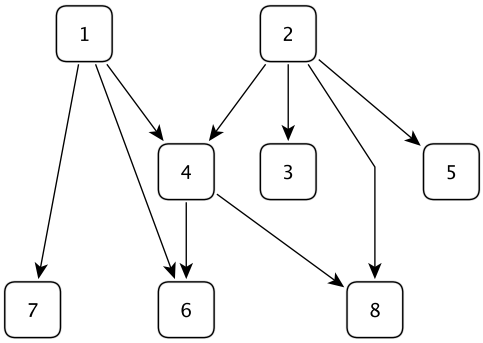


Abbildung 2.18: Hierarchischer Ansatz Teil 3 [Battista et al. [1999, Seite 23]]

Zum anderen braucht es einen Algorithmus, der ein Gleichgewicht in diesem System findet. Das heißt, es muss für jeden Knoten eine Position gefunden werden, in dem keine Kraft auf diesen wirkt.

Der Force-Directed Ansatz führt meist zu guten graphischen Resultaten und er ist auch aufgrund eines dahinterliegenden physikalischen Modells leicht zu verstehen und zu implementieren. Einer der einfachsten Force-Directed Ansätze ist laut [Battista et al., 1999] das Feder und elektrische Kräfte System (engl.: Springs and Electrical Forces). Dieser Ansatz entspricht grundlegend dem Force-Directed Ansatz von Fruchterman und Reingold [1991], auf den weiter unten noch genauer eingegangen wird. Hierbei werden die Kanten als Federn angenommen und die Knoten als elektrisch gleichgeladene Teilchen. Damit liegt eine Summenkraft aus Federkraft und Abstoßungskraft an den Knoten an.

$$F(v) = \sum_{(u,v)\in E} f_{uv} + \sum_{(u,v)\in V\times V} g_{uv} \tag{2.1}$$

In Gleichung 2.1 steht f_{uv} für die Federkraft nach dem hookeschen Gesetz zwischen den Knoten u und v und der Term g_{uv} für die elektrische Abstoßung, welche mit der Entfernung quadratisch abnimmt. Geht man nun davon aus, dass sich der Knoten v auf der Position $p_v = (x_v, y_v)$ befindet und der euklidische Abstand zweier Punkte p und q mit $d(p, q)$ angegeben wird, ergibt sich folgende Gleichung für die x-Komponente des Knotens v. Die Gleichung für die y-Komponente ist der Gleichung 2.2 ähnlich.

$$F_x(v) = \sum_{(u,v)\in E} k_{uv}(d(p_u, q_v) - l_{uv})\frac{x_v - x_u}{d(p_u, p_v)} + \sum_{(u,v)\in V\times V} \frac{e_{uv}}{(d(p_u, p_v))^2}\frac{x_v - x_u}{d(p_u, q_v)} \tag{2.2}$$

In 2.2 sind die Parameter $k_{(uv)}$, $l_{(uv)}$ und e_{uv} unabhängig von der Position der Knoten. Durch Gleichung 2.2 ergeben sich folgende Eigenschaften für das Modell.

l_{uv} ist die natürliche Länge der Feder. Das heißt, wenn die Knoten u und v sich in der Distanz l_{uv} befinden wird keine Federkraft auf beide Knoten ausgeübt.

Die Steifigkeit der Feder wird mit k_{uv} angegeben. Je größer die Steifigkeit ist, desto eher werde sich die Knoten u und v auf den Abstand l_{uv} einpendeln.

Die Abstoßung hängt von der Größe e_{uv} ab.

Dadurch ergeben sich implizit zwei ästhetische Eigenschaften. Aufgrund der Federkraft sind alle Kantenlängen nahezu gleich, nämlich l_{uv}. Zum anderen sind die Knoten nicht zu nahe aneinander angeordnet. Dafür sorgt die elektrische Abstoßung e_{uv}

Auf diesen grundsätzlichen Überlegungen bauten Fruchterman und Reingold [1991] auf und führten mehrere Einschränkungen ein, um den Algorithmus einfach zu halten und die Berechnungsgeschwindigkeit und damit die Anzeigegeschwindigkeit zu erhöhen. Im ersten Versuch vereinfachten sie das System soweit, dass nur benachbarte Knoten angezogen werden, und nur die Abstoßungskraft auf alle Knoten wirkt. Dies hat den Vorteil, dass bei jeder Iteration die Zeitkomplexität für die Berechnung der Anziehungskraft auf $\Theta(\mid E \mid)$ abnimmt. Die Zeitkomplexität zur Berechnung der Abstoßungskraft bleibt jedoch bei $\Theta(\mid V^2 \mid)$. Um auch hier Vereinfachungen vorzunehmen führten sie einen neuen Gitternetzalgorithmus (engl.: grid-variant algorithm) ein. Bei diesem Algorithmus werden Gitternetze in der graphischen Ebene eingezogen. Bei jeder Iteration werden die Knotenpositionen einem Quadranten des Gitternetzes zugeteilt und nur Knoten, welche sich in einem Nachbarquadranten und zusätzlich in einem bestimmten Radius des Knotens befinden, können angezogen werden. Die Einführung eines zusätzlichen Radius war nötig, um etwaige Verzerrungen in der Darstellung zu vermeiden. Abbildung 2.19 zeigt mehrere Knoten, welche in ein Gitternetz unterteilt sind. In Abbildung 2.19 wird der Knoten q vom Knoten v angezogen, da sich q innerhalb des definierten Radius von v und in einem Nachbarquadranten befindet. Im Gegensatz zu Knoten r, welcher sich zwar in einem Nachbarquadranten zu v befindet, aber ausserhalb des Radius von v befindet. Der Knoten s wird ebenfalls nicht angezogen, schon aufgrund der Tatsache, dass sich dieser nicht in einem Nachbarquadranten befindet. Durch die Vereinfachung wird die Berechnungszeit auf $\Theta(\mid V \mid)$ minimiert ohne jedoch große Abstriche in der Darstellungsqualität hinnehmen zu müssen. Damit rechnet der Gitternetzalgorithmus in $\Theta(\mid V \mid + \mid E \mid)$. Des Weiteren widerspricht diese Vereinbarung nicht den zwei Grundprinzipien die sich Fruchterman und Reingold [1991] vorgenommen hatten.

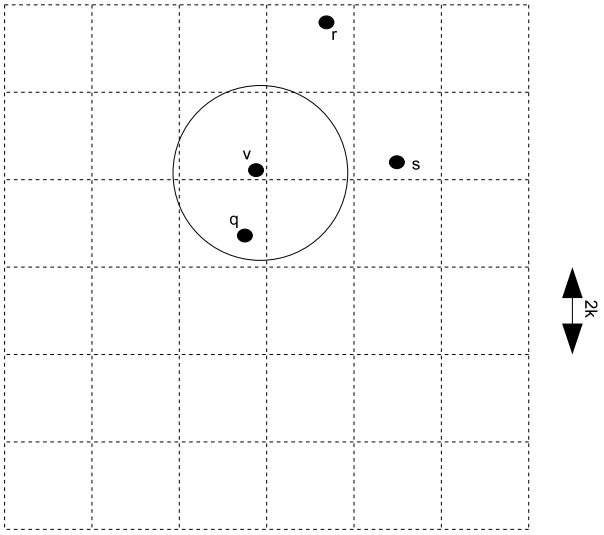


Abbildung 2.19: Grid-Variant Algorithmus [Fruchterman und Reingold [1991, Seite 1136]]

- Knoten, welche über eine Kante verbunden sind, sollen nahe aneinander gezeichnet werden.

- Knoten sollen nicht zu nahe nebeneinander gezeichnet werden.

Diese Vereinfachung in Bezug auf das Einführen von unrealistischen Kräften wurde auch schon von [Eades, 1984] angewandt. Der Algorithmus von Fruchterman und Reingold [1991] besteht im wesentlichen aus drei Schritten, indem zuerst die Anziehungskräfte, danach die Abstoßungskräfte und zuletzt ein Abkühlungstemperatur errechnet wird. Die Abkühlungstemperatur ist dafür gedacht, dass ein Knoten am Anfang der Simulation große Koordinaten Sprünge erfahren kann und über die Zeit diese Sprünge immer kleiner werden. Die Anziehungs- bzw. Abstoßungskräfte nach Fruchterman und Reingold [1991] werden folgendermaßen definiert.

$$f_a(d) = \frac{d^2}{k} \cdots Anziehungskraft \tag{2.3}$$

$$f_r(d) = \frac{-k^2}{d} \cdots Abstoßungskraft \tag{2.4}$$

Die Knoten werden gleichmäßig verteilt und die Kantenlänge ist einheitlich. Die Kantenlänge errechnet sich aus der Summer der beiden Kräfte f_a und f_r. Wenn diese Summe null ist befindet sich der Knoten im statischen Gleichgewicht und dies ergibt aus den beiden Formeln 2.3 und 2.4 immer eine Kantenlänge von k. Laut Fruchterman und Reingold

[1991] wird k folgendermaßen definiert.

$$k = C\sqrt{\frac{Fläche}{Knotenanzahl}} \tag{2.5}$$

Die Konstante C wurde experimentell gefunden, sodass k einen Radius einer leeren Fläche rund um den Knoten ergibt.

Allgemein ist der Force-Directed Ansatz für ungerichtete Graphen geeignet. Es gibt aber auch Algorithmen zum Zeichnen von gerichteten Graphen. Sugiyama und Misue [1995] benutzten zusätzlich zu dem Modell mit Federn und elektrischen Kräften ein Magnetfeld. Dieses Magnetfeld kann radial, parallel, konzentrisch oder aber auch eine Kombination aus den dreien sein. Die Federn in dem Modell können entweder nicht magnetisch, oder magnetisch (uni- bzw. bidirectional) sein. Unidirectionale Federn richten sich immer in Richtung des Magnetfeldes aus, bidirektionale können sich mit oder aber auch gegen die Richtung des Magnetfeldes ausrichten. Durch dieses Modell ist es auch möglich gemischte (bestehend aus gerichteten und ungerichteten Kanten) Graphen automatisiert und ästhetisch zu zeichnen. Auch orthogonale Kantendarstellungen sind damit möglich.

2.6.9 Platzfüllende Visualisierungen

Alle bisher besprochenen Visualisierungsarten von Graphen oder hierarchischen Strukturen wurden nicht unter dem Aspekt der maximalen Ausnutzung der Zeichenfläche betrachtet. Platzfüllende Visualisierungen zielen darauf ab, den gesamten vorhandenen Zeichenbereich zu nutzen, um so entweder mehr Information unterzubringen, oder Information größer darstellen zu können. Beides kann die Lesbarkeit erhöhen.

Unter den platzfüllenden Visualisierungen für hierarchische Strukturen sind wahrscheinlich die rechteckigen (engl.: Treemap) und radialen (engl.: Sunburst) Darstellungen am bekanntesten. Beide Darstellungen können mit Farben erweitert werden, um verschieden Attribute hervorzuheben. Abbildung 2.21 zeigt zum Beispiel ähnliche Farben zwischen Eltern- und Kindknoten.

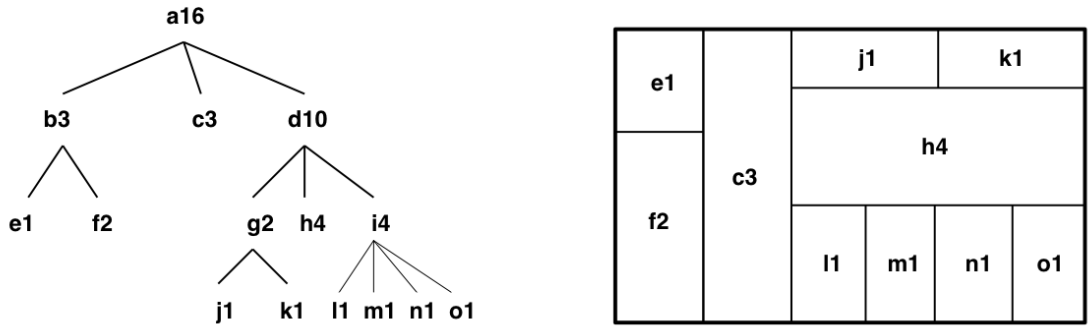
Treemap

Die Treemap wurde von Johnson und Shneiderman [1991] ursprünglich entwickelt, um die Festplattenbelegung visuell darzustellen. Dabei wird allen Knoten im Baum ein Gewicht zugeordnet. In jeder Knotenebene wird der vorhandene Platz untern den Knoten bezüglich der Gewichtung aufgeteilt. Ist der Knoten im Baum ein Blatt (besitzt keine

Kindknoten) wird der Platz nicht weiter zerlegt. Anderenfalls muss der vorhandene Platz mit den Kindknoten geteilt werden. Die Kindknoten können wiederum Kindknoten beinhalten und somit muss der Algorithmus rekursiv wieder von vorne beginnen, um den restlichen vorhanden Platz wieder aufzuteilen. Die Abbildung 2.20 zeigt den Zusammenhang zwischen dem Tree Diagramm (Abbildung 2.20a) und der resultierenden Tree Map (Abbildung 2.20b). In Abbildung 2.20b kann man die Gewichtung aufgrund der Größe des Rechtecks erkennen, dies ist nicht aus der Abbildung 2.20a ersichtlich. Von der Treemap gibt es mittlerweile viele Varianten, um zum Beispiel Schwächen der ursprünglichen Methode von Johnson und Shneiderman [1991] auszugleichen. Erwähnenswert in diesem Zusammenhand ist die quadratisch ausgerichtete Treemap (engl.: Squarified Treemap) [Bruls et al., 1999], welche eine der Schwächen des ursprünglichen Algorithmus der Treemap minimiert. Die Squarified Treemap versucht jeden Quadranten innerhalb der Treemap annähernd mit einem Seiten-Längen Verhältnis von 1:1 darzustellen. Damit können lange dünne Rechtecke, wie sie mit der Methode nach Johnson und Shneiderman [1991] entstehen, in der Darstellung meist verhindert werden. Nach Bruls et al. [1999] wird die verfügbare Fläche für die Kinder eines Knotens wie folgt aufgeteilt. Entweder werden die nächsten Flächen in vertikalen oder horizontalen Reihen ausgelegt. Dies ist abhängig davon, ob die Breite oder die Höhe des Rechteckes größer ist. Wenn nun ein neues Rechteck eines Kindes zur Reihe hinzugefügt werden soll, hängt dies davon ab, ob danach das Seitenverhältnis der gerade aktiven Reihe verbessert oder verschlechtert wird. Wird das Seitenverhältnis verbessert, also nähert es sich an 1:1 an, wird das Rechteck in die Reihe aufgenommen. Verschlechtert es hingegen das Seitenverhältnis, dann wird die aktive Reihe fixiert und die restlichen Kinder auf die übrige Fläche mit neuen vertikalen und horizontalen Reihen aufgeteilt.

Sunburst

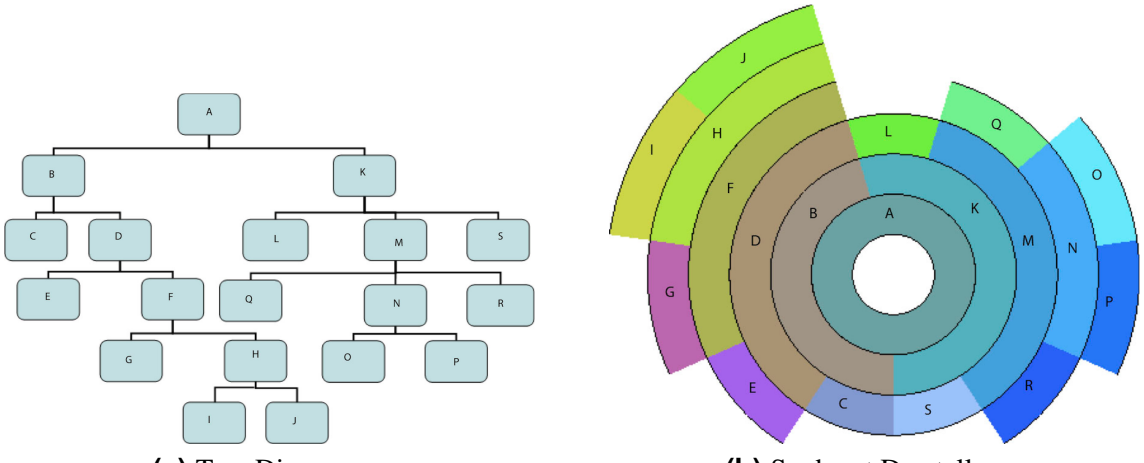
Sunburst Darstellungen sind radiale Display füllende hierarchische Visualisierungen (siehe Abbildung 2.21), welche den Wurzelknoten in der Mitte darstellen und nach außen ringförmig die verschiedenen Schichten der Hierarchie anordnen. Wobei jeder Ring in genauso viele Teile zerlegt wird, wie es Knoten in dieser Hierarchie gibt. Wird zum Beispiel ein Filesystem als Sunburst Darstellung visualisiert, so könnte der Winkel die Datei- oder Verzeichnisgröße und die Farbe den Dateityp darstellen. In [Stasko und Zhang, 2000] werden die Darstellungen von Treemaps und Sunburst miteinander verglichen. Dafür wurden zwei Filebrowser Programme geschrieben, eines der beiden Programme stellte die Hierarchie mit Sunburst, das andere mittels Treemap dar. 60 Studenten mussten mit Hilfe



(a) Tree Diagramm **(b)** Tree Map

Abbildung 2.20: Darstellung einer Tree Map in Abhängigkeit des Tree Diagrammes [Bruls et al. [1999]]

dieser Programme bestimmte Aufgaben erfüllen. Die Kriterien zur Erfüllung der Aufgaben waren die Richtigkeit und die Geschwindigkeit. Dabei stellte sich heraus, dass die Studenten die Sunburst Darstellung gegenüber der Treemap bevorzugten, weil Sunburst eine eindeutigere Darstellung der Hierarchie bietet. Dies wirkte sich vor allem dann aus, wenn es sich um eine große Hierarchie handelte (z.B. 3000 Dateien). Ausserdem konnte festgestellt werden, wenn Studenten zuerst mit dem Sunburst Programm arbeiteten, sich ein gewisser Lerneffekt einstellte und sie danach besser mit dem Treemap Programm arbeiten konnten, als Studenten welche zuerst mit dem Treemap Programm arbeiteten. Eine genaue Beschreibung der Studie ist unter [Stasko et al., 2000] nachzulesen.



(a) Tree Diagramm **(b)** Sunburst Darstellung

Abbildung 2.21: Darstellung einer Sunburst in Abhängigkeit des Tree Diagrammes [http://jcsites.juniata.edu/faculty/rhodes/ida/relations.html Zuletzt besucht am: 2012.09.02]

Kapitel 3

Anforderungen und Analyse der Daten

3.1 Kapitelüberblick

Das Kapitel teilt sich in zwei wesentliche Kategorien. Zum einen wird der Datensatz des Anforderungskataloges genauer analysiert zum anderen werden die Anforderungen an die Software mithilfe der Volere Anforderungs- und Spezifikationsvorlage (engl.: Volere Requirements Specification Template) erarbeitet.

3.2 Analyse der Daten

Wie schon in Kapitel 1 beschrieben, sind die gesammelten Daten in einem sogenannten Anforderungskatalog gespeichert. Die Abbildung 3.1 zeigt einen Ausschnitt des Anforderungskataloges. Aufbauend auf den Daten, welche im Excel Format gesammelt wurden, wurde eine manuelle Analyse durchgeführt. Das heißt, es wurde versucht Gemeinsamkeiten und Zusammenhänge zu finden bzw. Gruppierungen vorzunehmen. Das Dokument ist zeilenweise zu lesen, wobei jede Zeile auch als Wirkkette bezeichnet wird. Jeder Wirkkette ist ein bestimmtes Problemfeld und ein bestimmtes Szenario zugeordnet. Das Dokument besitzt mehr als 400 unterschiedliche Wirkketten. Damit unterscheidet sich jede Zeile zumindest in einer Spalte inhaltlich von allen anderen Zeilen.

Spalten wie zum Beispiel das Problemfeld oder der Szenarien eignen sich gut für Filtereigenschaften, da die Problemfelder in 22 unterschiedliche Kategorien und die Sze-

narien in nur 4 unterschiedliche Kategorien eingeteilt sind. Genauer gesagt gehört jede Wirkkette einem der 22 Problemfelder und einem der 4 Szenarien an. Anders verhält es sich bei den Spalten mit den Annahmen und den Anforderungen. Beide Spalten sind mit Text in prosaform gefüllt. Die Spalte *Gleich* drückt eine gewisse Verwandtschaft mit anderen Anforderungen aus. Haben zwei Anforderungen dieselbe Zahl in der Spalte *Gleich* stehen, so sind sie miteinander verwandt. Des Weiteren wird jede Wirkkette in verschiedene Ebenen eingeteilt. Per Definition können Wirkketten der Ebenen *Mensch*, *Organisation*, *IT*, *Prozess*, und *Methode* zugeteilt werden. Anders als bei den Szenarien, bei denen eine Wirkkette einem speziellen Szenario zugeordnet wird, kann eine Wirkkette mehreren Ebenen zugeteilt werden. Damit ergeben sich 32 verschiedene Ebenenzustände pro Wirkkette.

Alle weiteren Spalten, welche in dem Dokument noch vorkommen, werden von der Software aufgrund der vorgegebenen Anforderungen an die Software nicht berücksichtigt. Dies resultiert auch daraus, dass die oben erwähnten und von der Software erfassten Spalten einer höheren Priorität und Aussagekraft zugeteilt werden Es sei aber angemerkt, dass es wenig Implementieraufwand verursachen würde weitere Spalten mit Hilfe der Software einzulesen und auszuwerten. Möglicherweise könnten die zusätzlichen Daten Platzprobleme bei der Visualisierung verursachen und man müsste sich weitere Darstellungsarten, wie in Kapitel 2 ausführlich erklärt überlegen.

Der Inhalt der Spalten kann sich in anderen Zeilen wiederholen. Das gilt nicht nur für die Spalte Problemfeld, bei der dieser Umstand eindeutig erkennbar ist, da es nur 22 verschiedene Problemfelddefinitionen für mehr als 400 Wirkketten gibt. Es kommen auch viele Annahmen- und Anforderungsdefinitionen mehrfach in verschiedenen Wirkketten vor. In diesem Fall muss die Software während des Einlesevorganges automatisch erkennen, ob diese Textfrage schon einmal eingelesen und gespeichert wurde, oder eben nicht.

		Szenario 2020						Ebene						
Nr.	Problemfeld	1	2	3	4	Annahme 2020	Anforderung 2020	Gleich	Mensch	Orga	IT	Prozess	Methode	Cluster	Lösungsansatz
1	Stark domänenübergreifende Zusammenarbeit	x				Verschmelzung der Domänen	Interdisziplinäre Rollen müssen gefördert werden.		x	x					
2	Stark domänenübergreifende Zusammenarbeit	x				Verschmelzung der Domänen	Eine einheitliche Sicht auf Ergebnis („Blick aufs Ganze") muss vermittelt werden.		x				x		
3	Stark domänenübergreifende Zusammenarbeit	x				Verschmelzung der Domänen	Sozialisierungsmassnahmen müssen (z.B. bei Kick-Off, Projektauflösung) verstärkt durchgeführt werden.	5	x	x				3	Kick Off Meeting; Get-to-know Teambuildingevent; Abschluss übergreifende Projektassisten
4	Stark domänenübergreifende Zusammenarbeit	x				Echtzeit-Versionierung und Auto-DMU	Ein durchgängiges Datenmanagement muss gewährleistet werden und etabliert sein.	51			x			2	
5	Stark domänenübergreifende Zusammenarbeit	x				Echtzeit-Versionierung und Auto-DMU	Ein effizientes & effektives Versionierungssystem muss vorhanden sein.	56			x			2	Automatische Benachrichtigur Änderungen
6	Stark domänenübergreifende Zusammenarbeit	x				Echtzeit-Versionierung und Auto-DMU	Es muss neben einer generalisierten Visualisierung des Produktes auch die notwendigen Detailsichten geschaffen werden (Visualisierung von Auswirkungen).				x				

Abbildung 3.1: Auszug aus dem Anforderungskatalog

3.3 Softwarerequirements nach Volere

Um alle essentiellen Fragen bezüglich der Anforderung an die Software zu finden und zu stellen wurde die Volere Anforderungs- und Spezifikations Vorlage (engl.: Volere Requirements Specification Template) benutzt. Die Eckpunkte der Vorlage finden sich unter http://www.volere.co.uk/template.htm und ordentliche Studenten können das Spezifikationstemplate kostenfrei anfordern. In der hier vorliegenden Arbeit werden nicht alle Punkte aus der Vorlage verwendet beziehungsweise besprochen, sondern nur diese, die für die Arbeit relevant sind. Allgemein kann behauptet werden, dass die Vorlage eine Vielzahl von Kategorien zur Verfügung stellt, welchen den einzelnen Anforderungen des Projektes zugeordnet werden können. Dadurch ist es möglich sich einen guten Überblick über das Projekt und auch dessen potentiellen Problemen verschaffen.

Folgende Überschriften wurden von dem Volere Dokument entnommen und mit den Anforderungen der vorliegenden Arbeit befüllt.

3.3.1 Treibende Kräfte im Projekt - Project Drivers

Zweck des Projektes

Wie in Kapitel 1 beschrieben ist der Zweck dieser Software den vorhandenen Datensatz (Anforderungskatalog) aufzubereiten und graphisch darzustellen. Die Informationen sollen dahingehend so dargestellt werden, dass auch Personen, welche sich nur am Rande mit dem Anforderungskatalog beschäftigt haben, sich relativ einfach zurecht finden und in weiterer Folge Wissen daraus schöpfen können.

Spezialisten sollen mithilfe der Software anderen Personen, welche sich nicht mit der Thematik FuturePLM und dem daraus resultierendem Datensatz beschäftigt haben, einen visualisierten Überblick über das Thema geben können.

Ein weiterer Aspekt für den Zweck der Software ist es neue Technologien in diesem Bereich zu finden und einzusetzen.

Auftraggeber, Kunde und Interessensvertreter

Auftraggeber der vorliegenden Arbeit und damit implizit auch der Software ist das Kompetenzzentrum Das Virtuelle Fahrzeug (ViF) und die Partner im K2-Forschungsprojekt ”FuturePLM”.

Anwender des Projektes

Die Anwender des Projektes sind die Teilnehmer am K2-Forschungprojekt ”FuturePLM”.

Einschränkungen

Die Software soll als Client - Server Anwendung ausgeführt sein. Das heißt die Anwendung soll mit einem gängigen Browser auf Laptops oder Standrechner ausführbar sein. Einschränkungen bestehen dahingehend, dass die Software nicht auf mobilen Geräten wie zum Beispiel Smartphones oder Tablets lauffähig sein muss.

3.3.2 Funktionale Anforderungen

Nach Abklärung mit den Teilnehmern des Forschungsprojektes wurden folgende Kernfragen ermittelt. Das Softwaresystem soll in der Lage sein diese Fragen in graphischer Form zu beantworten.

- Welche Anforderungen haben ihren Ursprung in Szenario XY?
- Welchen Lösungsansatz hat die Anforderung XY?
- Welche Anforderungen sind mit anderen Anforderungen verwandt?
- Welche Anforderungen sind Teil eines bestimmten Problemfeldes?
- Welche Anforderungen sind einer bestimmten Ebene zugeteilt?

3.3.3 Nichtfunktionale Anforderungen

Look and Feel Anforderung

Look and Feel Eigenschaften in Bezug auf Anwender Interaktionen werden unter Punkt 2.4 näher erläutert. Mit Ausnahme der Punkte ”History” und ”Extract”, werden diese Punkte als Anforderungen an die Software gestellt. Weitere Look and Feel Eigenschaften, welche zum Beispiel Menüführung und Buttonanordnungen betreffen werden hier nicht in die Anforderungsliste aufgenommen, da das Hauptaugenmerk auf die Visualisierung der Daten gelegt wird.

Benutzerfreundlichkeit Anforderung

Der Benutzer der Software soll sich schnell und ohne langwierige Einschulung in dem Programm zurechtfinden. Vorkenntnisse zum Projekt FuturePLM und den daraus resultierenden Anforderungskatalog sind nötig um die visualisierten Daten richtig interpretieren zu können.

Leistungsanforderung

Da der Datensatz nicht all zu groß ist, und es auch nur einen kleinen Anwenderkreis gibt, werden keine großen Datendurchsätze und somit Leistungseinbußen erwartet. Clientseitig ist darauf zu achten mit den neuesten Browserversionen zu arbeiten, damit auch alle Funktionalitäten unterstützt werden.

Wartungsanforderung

Da es sich um einen Prototyp handelt werden keine Überlegungen seitens Wartungsanforderung angestellt.

Sicherheitsanforderung

Anforderungen an die Sicherheit werden aufgrund des prototypischen Charakters der Software ausser Acht gelassen.

Kapitel 4

Technologieauswahl und -beschreibung

4.1 Kapitelüberblick

Die Softwarelösung teilt sich in einen serverseitigen und clientseitigen Teil auf. In diesem Kapitel wird näher auf die jeweiligen benutzten Technologien, eingegangen. Das Kapitel dient dazu, Entscheidungen bei der Technologieauswahl zu begründen und die verwendeten Technologien in Bezug auf die vorliegende Arbeit zu erklären und auf etwaige Besonderheiten dieser Technologien einzugehen.

4.2 Nutzwertanalyse

Zur Entscheidungsfindung bei der Technologieauswahl wurde eine Nutzwertanalyse verwendet. Für diese Analyse wurden in erster Linie Programmiersprachen (server- und clientseitig) mit bestimmten, für diese Arbeit, wichtigen Kriterien herangezogen. Alle diese Programmiersprachen eignen sich für Netzwerkanwendungen und diesem Fall für die Anforderungen die in Kapitel 3 beschrieben werden. Die einzelnen Kriterien wurden unterschiedlich gewichtet und mit einem Notenschlüssel von 0 bis 9 für die jeweiligen Programmiersprache beurteilt (siehe Tabellen 4.1 und 4.2). In der Spalte ”Ges. Note” wird das Produkt aus Gewichtung und der vergebenen Note angeführt. Am Ende jeder Tabelle wird die Gesamtsumme aller einzelnen Gesamtnoten dargestellt. Die Programmiersprache mit der höchsten Gesamtpunktzahl wurde für die praktische Arbeit ausgewählt und verwendet.

Die Gewichtung wurde teilweise nach eigenem Ermessen und teilweise aufgrund der

vorgegebenen Anforderungen vorgenommen. So wurden die beiden Punkte ”Keine Lizenzkosten” und ”pers. Programmiererfahrung” sehr hoch gewichtet. Dies hat einerseits den Grund, dass die Software kostengünstig ohne jegliche zusätzlichen Kosten und andererseits in einem knapp bemessenen Zeitraum geschrieben werden soll.

Des Weiteren wurden keine Vergleiche in Bezug auf Performance angestellt, da die Software in erster Linie zu Demonstrationszwecken dienen soll und daher mit wenigen zeitgleichen Anwendern zu rechnen ist. Deshalb wurde angenommen, dass alle Programmiersprachen und die damit zu verwendeten Server, die zu erwartende geringe Auslastung ohne merkliche Geschwindigkeitseinbußen bewältigen können.

Kriterium	**Gewichtung**	**Java**		**Python**		**PHP**	
		Note 0-9	Ges. Note	Note 0-9	Ges. Note	Note 0-9	Ges. Note
Keine Lizenzkosten	35	9	315	9	315	9	315
Objektorientiert	15	9	135	9	135	9	135
Pers. Programmiererfahrung	35	8	280	1	35	2	70
Erweiterbarkeit Application Server	15	9	135	5	75	5	75
Summe			**865**		560		595

Tabelle 4.1: Nutzwertanalyse für serverseitige Software

Kriterium	**Gewichtung**	**Javascript**		**Actionscript**	
		Note 0-9	Ges. Note	Note 0-9	Ges. Note
Keine Lizenzkosten	30	9	270	0	0
Objekt-orientiert	10	3	30	9	90
Pers. Programmiererfahrung	15	1	15	3	45
InfoVis Libraries	20	8	160	8	160
Keine Plug-Ins	25	8	200	4	100
Summe			**675**		395

Tabelle 4.2: Nutzwertanalyse für clientseitige Software

Das Ergebnis in den Tabellen 4.1 und 4.2 zeigt eindeutig, dass serverseitig Java und clientseitig Javascript verwendet werden sollen. Dies hat nicht nur Vorteile für die vorliegende Software mit den geforderten Anforderungen unter Punkt 3.3, sondern auch für zukünftige auf diese Arbeit aufbauende Projekte. Denn mit Java und der Java EE (Java Platform Enterprise Edition) Spezifikation kann serverseitig die Funktionalität einfach und leistungsfähig erweitert werden. Mit der Wahl Javascript clientseitig einzusetzen, wird ebenfalls auf eine Technologie gesetzt, die sich aller Voraussicht nach in den nächsten Jahren weiter etablieren wird. Dies kann auch im Fall von Actionscript möglich sein, jedoch sind die Lizenzkosten sowie ein browserseitiges Plug-In für diese Anwendung nicht akzeptabel.

4.3 Serverseitige Technologien

Aufgrund des Ergebnisses unter 4.2 wurden alle weiteren Technologien, welche serverseitig nötig sind, nach dem Kriterium der Programmiersprache ausgewählt. Serverseitig wurden die Technologien dahingehend ausgewählt, um mit Java arbeiten zu können. Der Server soll als Webserver dienen und Java Code in Servlet Container ausführen können.

4.3.1 Tomcat Server

Wie unter 4.3 gefordert, bietet der Apache Tomcat Server die Funktionalität Java Code in Servlet Container auszuführen und bietet einen integrierten Webserver. Zudem ist der Apache Tomcat Server auf allen gängigen Betriebssystemen lauffähig und unterliegt der Apache-Lizenz Version 2.0 (`http://www.apache.org/licenses/LICENSE-2.0`). Software, welche der Apache Lizenz unterliegt, kann frei verwendet werden. Es muss lediglich eine Kopie der Lizenz bei der Weiterverteilung mitgeliefert werden.

4.3.2 Stripes Framework

Um Anwenderaktionen, welche vom Browser aus getätigt werden, möglichst effizient am Server verarbeiten zu können, wurde ein Framework gesucht, welches leicht zu verwenden und zu konfigurieren ist. Dabei fiel die Wahl auf das Stripes-Framework, welches zum einen ebenfalls wie der Apache Tomcat Server, der Apache Lizenz Version 2 unterliegt und andererseits extrem einfach in die Webapplikation einzubetten und zu konfigurieren ist.

Das Stripes Framework basiert auf einem Model-View-Controller Framework. Wobei das Framework am meisten in den View und Controller Komponenten eingreift. Es ergänzt sich gut mit dem zustandslosen HTTP-Protokoll. Grundsätzlich basiert das HTTP-Protokoll auf einem Anfrage-Antwort (engl.: request-response) Zyklus. Das heißt, wenn der Anwender im Browser ein klickbares Element anklickt, wird eine Anfrage an den Server gestellt, welche vom Server verarbeitet und die Antwort wieder an den Browser geschickt wird. Nach dem Erhalt der Antwort aktualisiert sich der Browser und der Zyklus ist abgeschlossen. Danach ist das System wieder für einen neuen Zyklus bereit. Das Stripes Framework bindet sich in diesen Anfrage-Antwort Zyklus mit ein, indem die Anfrage vom Browser in eine sogenannte Aktion (engl.: action) umgewandelt wird. Diese Aktion triggert eine beliebig definierbare Java Methode, welche in einem Java Servlet läuft. Damit lassen sich alle logischen Funktionalitäten in Java implementieren und das Ergebnis kann wieder mit Hilfe des Stripes Frameworks und HTTP Protokoll an den Browser weitergeleitet werden. Durch die transparente Einbindung des Stripes Frameworks ist es möglich, auch andere Softwarebibliotheken parallel zu verwenden. Ein weiterer Vorteil des Stripes Frameworks ist der sehr geringe Aufwand an Konfiguration. Dafür ist nur die web.xml-Datei zu verändern. In dieser muss das Package angegeben werden, in dem sich die sogenannten Action Beans befinden. Diese Anweisung reicht aus, damit alle in diesem Package und auch in den Subpackages befindlichen Action Bean-Klassen registriert werden. Die Action Beans sind jene Java-Klassen, welche die Java Methoden beinhalten, die dann nach einem benutzerseitigen Aufruf ausgeführt werden. (Daoud [2008])

Abbildung 4.1 zeigt eine Standardkonfiguration mit Hilfe von Stripes. Der Browser sendet eine Anfrage, diese wird mittels Stripes Filter (Einstellung über web.xml) und dem Dispatcher Servlet von Stripes an das richtige Action Bean weitergeleitet. Die Antwort des Action Bean wird entweder direkt an den Browser zurückgesendet, oder an eine JSP Datei weitergeleitet, welche weitere Operationen durchführen kann, bevor eine Antwort an den Browser gesendet wird.

Model-View-Controller Muster

In Anlehnung an Gamma et al. [1994] kann ein Software-Muster (engl.: software-pattern) in vier Elemente - *Problem*, *Lösung*, *Konsequenzen* und *Namen* unterteilt werden. Das *Problem* beschreibt wann man ein bestimmtes Muster einsetzen kann. Dies kann zum Beispiel eine Objektstruktur sein, welche immer zu einem inflexiblen Design in einem bestimmten Kontext führt. Die *Lösung* eines bestimmten Musters beschreibt die Elemente und deren Beziehungen, Kollaborationen bzw. Verantwortlichkeiten. Dabei ist die

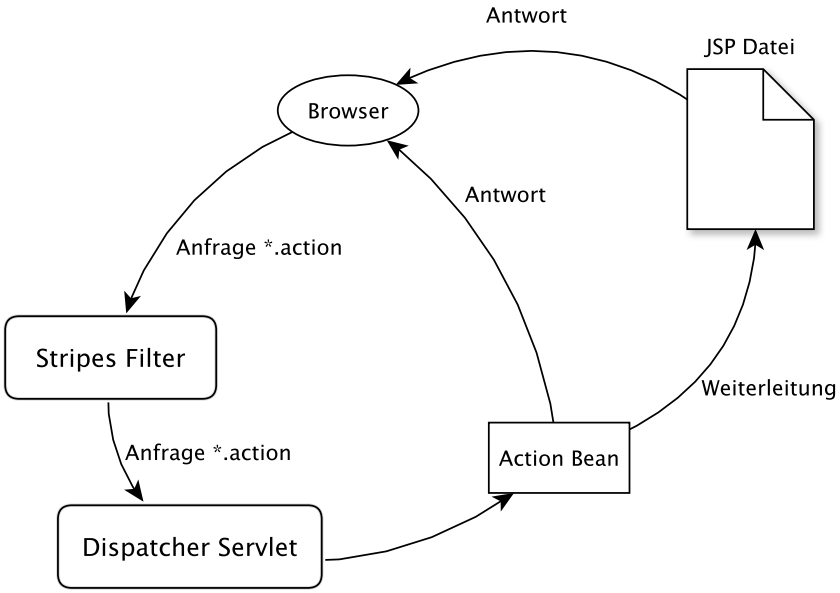


Abbildung 4.1: Stripes Konfiguration [Daoud [2008, Seite 26])]

Beschreibung abstrakt, um eine Lösung anzubieten welche in verschiedenen Anwendungsfällen und Kontexten einsetzbar ist. Die *Konsequenzen* beschreiben das Resultat bzw. auch die Einschränkungen welche das Muster mit sich bringt. Letztendlich besitzt jedes Muster einen eindeutigen Namen, um sicher zu gehen, dass jeder von demselben Sachverhalt spricht. Die nachfolgend kurze Beschreibung des Model-View-Controller Musters hält sich an die vier Elemente von Gamma et al. [1994] und wurde von Buschmann et al. [1996] übernommen. Die Abbildung 4.2 zeigt die Zusammenhänge zwischen den einzelnen Komponenten. Die durchgezeichneten Pfeile kennzeichnen eine direkte Assoziation und die strichlierten Pfeile eine indirekte Assoziation. Eine indirekte Assoziation kann ein weiteres Muster wie zum Beispiel ein, Observer Muster (siehe Gamma et al. [1994]) sein. Die zwei dickeren Pfeile vom bzw. zum Anwender zeigen die Interaktionsmöglichkeiten zwischen dem Anwender und dem System.

Problem

- Die gleiche Information soll in verschiedenen Fenstern bzw. Diagrammen darstellbar sein.
- Verschiedene ”look and feel” Standards sollen den Code in der Kern-Applikation nicht beeinflussen.
- Veränderungen von Daten sollen im User Interface gleich angezeigt werden.

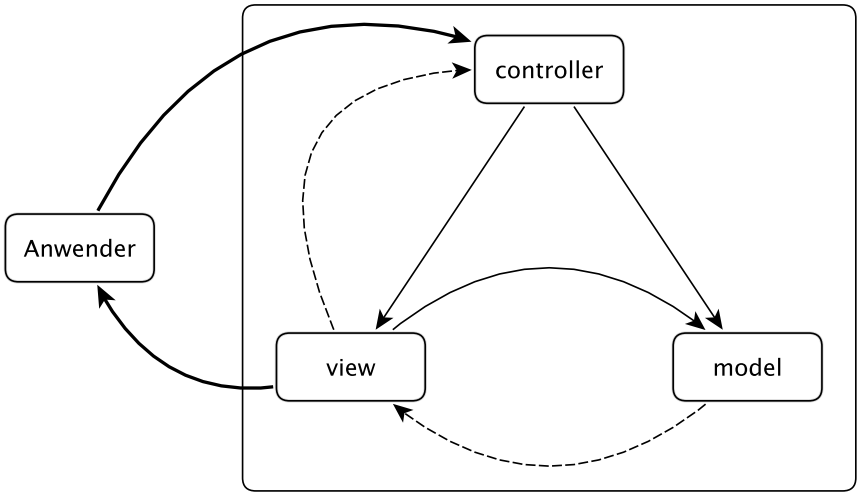


Abbildung 4.2: Model-View-Controller

Lösung

Das Model-View-Controller-Muster teilt die Applikation in drei verschiedene Komponenten auf. Es wird zwischen einer Eingangs-, Ausgangs- und Prozesskomponente unterschieden.

- Das **Model** beinhaltet nur die Kerndaten und Kernfunktionalität, jedoch nicht Programmteile zur Datenvisualisierung oder Anwendereingabe.
- Die **View** Komponente ist für die Visualisierung verantwortlich. Die Daten zur Visualisierung werden von der Modellkomponente übergeben.
- Die **Controller** Komponente ist für die Anwenderinteraktion verantwortlich. Das heißt, es werden Mausklicks oder Tastatureingaben verarbeitet und entsprechend an die View- oder Modellkomponente als Service weitergeleitet.

Konsequenzen

- Durch die Trennung der Daten und der anwenderspezifischen Funktionalität lassen sich mehrere Darstellungen mit einem einzigen Datenmodell implementieren.
- Unter zu Hilfenahme eines weiteren Musters namens Observer, welches ebenfalls in Gamma et al. [1994] und Buschmann et al. [1996] beschrieben wird, können mehrere View Komponenten gleichzeitig aktualisiert werden, wenn sich der Datensatz in der Modellkomponente verändert.

- Erhöhung der Komplexität der Software. Schlecht geeignet ist das Model-View-Controller-Muster für graphische Menüs oder einfache Textelemente.
- Starke Bindung der Komponenten zwischen View, Controller und Model. Bei Änderung der Modellkomponentenschnittstelle müssen die View- und Controller Komponenten angepasst werden.

4.4 Clientseitige Technologien

Aufgrund der Nutzwertanalyse (siehe Tabelle 4.2) wurde auf der Clientseite Javascript in Verbindung mit HTML ausgewählt und verwendet. Nachfolgende Beschreibungen in diesem Kapitel zeigen eine Zusammenfassung der Recherche in Zusammenhang mit diesen clientseitigen Technologien.

In Bezug auf HTML wurde im speziellen eine Recherche über die neuen Funktionalitäten, welche HTML5 unterstützt oder in Zukunft unterstützen soll, durchgeführt. Die Recherche kann auf diesem sehr großem Gebiet nicht als vollständig angesehen werden, sondern bezieht sich großteils auf Funktionalitäten, welche für die vorliegende Arbeit von Interesse ist.

Die Recherche bezüglich Javascript und den angeführten Bibliotheken beschränkt sich ebenfalls auf Anforderungen der vorliegenden Arbeit. Da Javascript den objektorientierten Ansatz, vor allem wenn es um Vererbung mit privaten und öffentlichen Variablenzugriffe geht, nur sehr rudimentär unterstützt, werden einige Beispiele von Entwurfsmuster gezeigt, welche sich dieser Thematik annehmen. Weiterführende Beispiele sind auch im Anhang A angeführt.

4.4.1 HTML5

Nachfolgend soll ein kurzer Überblick über die neuen Funktionalitäten von HTML 5 gegeben werden, obwohl die Spezifikation des HTML5 Standards zum Zeitpunkt der Fertigstellung dieser Arbeit noch nicht beendet ist.

Es arbeiten zwei unterschiedliche Arbeitsgruppen WHATWG (Web Hypertext Application Technology Working Group) und W3C (World Wide Web Consortium) an den Spezifikationen. Die Arbeitsgruppe WHATWG wird von mehreren Unternehmen betrieben obwohl jeder an der Mailingliste teilnehmen kann. Die Spezifikation wird nur als HTML bezeichnet, mit dem Argument, dass es sich um eine lebende und ständig weiterentwickelnde Spezifikation handelt. Auf der anderen Seite arbeitet das W3C als un-

ternehmensunabhängiges Konsortium am HTML5 Standard. Dieser Standard wird auch in mehrere Teilspezifikationen aufgeteilt, um einen bestimmten Spezifikationsprozess pro Teilgebiet durchlaufen zu können. Das Ziel von W3C ist eine stabile Version unter dem Namen HTML5 als W3C Empfehlung (engl.: recommendation) zu veröffentlichen. Beide Arbeitsgruppen arbeiten mittlerweile eng zusammen auch aufgrund der Tatsache, dass die W3C Arbeitsgruppe 2007 den damaligen Entwurf von WHATWG als Grundlage und Startpunkt für den neuen HTML5 Standard übernommen hat.

Neue Funktionalitäten Übersicht

Der HTML 4 Standard unterstützte wenig bis keine Funktionalitäten in Bezug auf Multimedia oder Webanwendungen, welche clientseitige Datenbanken oder Multithreading unterstützen. Abhilfe brachten sogenannten Browser Plug-Ins, die von verschiedenen Hersteller in eigenen Entwicklungsumgebungen mit unterschiedlichen Programmiersprachen angeboten wurden. Dieser Trend wird mit der neuen HTML5 Spezifikation abgeschwächt, da diese nun viele dieser Funktionalitäten selbst definiert und Browserhersteller die Spezifikation in ihren Produkten umsetzen. Somit ist es nun möglich mithilfe der neuen Funktionen von HTML5 und in Verbindung mit Javascript und CSS, clientseitige hochwertige Browseranwendungen zu implementieren. Die nachfolgende Auflistung gibt eine Übersicht der neuen Funktionalitäten. Die Liste ist nicht vollständig, erklärt dennoch die wichtigsten Neuerungen.

Webformulare

Webformulare gab es auch schon in der Spezifikation HTML 4, jedoch wurde diese um zusätzliche HTML5 Elemente und Attribute erweitert. Es würde den Rahmen dieser Arbeit sprengen, alle neuen HTML5 Elemente und Attribute in Bezug auf Webformulare anzuführen. Eine vollständige Liste findet sich in der Spezifikation unter Hickson [2011a] in Kapitel 4. Mithilfe dieser neuen Attribute und Elemente ist es zum Beispiel möglich, Email- oder Url-Adressen zu validieren. Es ist nun auch möglich einen graphischen Kalender oder Fortschrittsanzeigen in HTML5 darzustellen.

Multimedia

Ein wichtiger Bestandteil warum HTML5 an Bedeutung zunimmt, ist die Möglichkeit Audio oder Videodaten im Browser ohne Plug-Ins wiederzugeben. Jedoch unterstützen die verschiedenen Browserhersteller nicht alle gängigen Audio- und Videoformate primär aus

lizenztechnischen Gründen. Es wird auch versucht einen neues Format mit neuen Codecs zu etablieren, um gerade lizenz- und patentrechtliche Hürden zu umgehen. Derzeit ist davon auszugehen, dass noch einige Zeit vergehen wird, bis sich die Browserhersteller auf ein gemeinsames Format einigen. HTML5 unterstützt aber auch alle gängigen Bedienelemente, welche für die Steuerung eines Audio- oder Videoplayers nötig sind.

Clientseitige Datenbanken

Bis zum Standard HTML 4 und vor allem wegen des zustandslosen HTTP-Protokolls war es bislang schwer Informationen zu Sessions zu speichern. Dies wurde nur durch sogenannte Cookies, welche clientseitig mit einer maximalen Größe von 4kByte gespeichert werden konnten, ermöglicht. Sollten Sessions mit vielen angefallenen Daten gespeichert werden, musste der Großteil der Daten am Server verwaltet werden. Bei vielen Nutzern muss der Serviceanbieter dementsprechend viel Speicher serverseitig zur Verfügung stellen. Abhilfe sollen mit der HTML5 Version verschiedene Ansätze bringen. Zum einen wurde der *Web Storage* spezifiziert, welcher grundsätzlich eine Erweiterung der Cookies ist, zum anderen arbeiten die Arbeitsgruppen und Browserhersteller an einer Spezifikation für eine clientseitige Datenbank.

Der Vorteil von *Web Storage* gegenüber Cookies besteht zu allererst in der Größe des zu verwendeten Speichers. Der W3C Standard [Hickson, 2011b] sieht einen willkürlich festgelegten Speicherbereich von 5MByte vor, der aber auf Rückfrage an den Anwender jederzeit erweiterbar sein soll. Web Storage Daten, welche als Key-Value Paare gespeichert werden, können nur clientseitig angelegt und bearbeitet werden. Die dafür nötigen Skripte müssen für einen bestimmten Web Storage immer vom gleichen Hostnamen, Port und Protokoll zur Verfügung gestellt werden. Dies ist auch unter dem Begriff Same-Origin-Policy bekannt. Die Spezifikation des Web Storage ([Hickson, 2011b]) gibt zwei Speichermöglichkeiten an, die sich auf die Lebenszeit der gespeicherten Daten auswirkt. Zum einen gibt es Daten, welche so lange erhalten bleiben bis die Session durch Schließen des Browserfensters beendet wird. Diese Session wird *sessionStorage* genannt und hat den Vorteil, dass der Anwender mehrere Transaktionen in verschiedenen Browserfenstern zur gleichen Zeit, desselben Anbieters durchführen kann. Die zweite definierte Session heißt *localStorage* und hat keine zeitliche Begrenzung, obwohl die Same-Origin-Policy ebenfalls gilt.

Bei der clientseitigen Datenbank gibt es zwei Spezifikationsentwürfe, wovon der *Web SQL Datenbank* Entwurf (siehe Hickson [2010]) seit November 2010 von der W3C nicht mehr weiterverfolgt wird. Als Grund für die Beendigung wird angeben, dass alle Brow-

serhersteller dasselbe SQL-Backend verwendet haben, jedoch für den W3C Standardisierungsprozess mehrere unabhängige Implementierungen nötig sind. Das heißt, viele Browserhersteller haben eine Web SQL-Datenbank implementiert, jedoch scheint es, als würde es nie eine fertige Empfehlung vonseiten der W3C geben. Anders verhält es sich beim der *Indexed Database* Spezifikationsentwurf (siehe [Mehta et al., 2012]). Dieser wird von der W3C weiterentwickelt und auch Browserhersteller haben, oder werden diesen implementieren. Die Frage stellt sich jedoch, ob diese Funktionalitäten schon von Anwendern wirtschaftlich genutzt werden sollen.

Web Socket

Web Sockets ermöglichen eine neue Kommunikationsart zwischen Server und Client. War es mit HTML4 und HTTP nur über Umwegen möglich Daten vom Server an den Client ohne eine Anfrage des Clients zu schicken, so wird das mit dem Web Socket ermöglicht. Dabei baut der Web Socket auf das HTTP Protokoll auf. Das heißt, zuerst muss der Client eine Anfrage an den Server schicken, danach kann man mithilfe des HTTP - *Upgrade* Headers in einen anderen Modus, im Falle der Web Sockets, auf das Web Socket Protokoll umschalten. Wenn diese Verbindung aufgebaut wurde ist es möglich vom Server Daten an einen oder mehrere Clients zu schicken, ohne auf eine Anfrage der Clients warten zu müssen. Auch in umgekehrter Richtung ist dies möglich (bidirektional). Das Web Socket Protokoll unterstützt eine unverschlüsselte und eine verschlüsselte Version, welche entweder auf HTTP oder HTTPS aufbaut. Voraussetzung für dieses Feature ist neben der Unterstützung der Browser auch die serverseitige Unterstützung. Aktuell wird dieses Protokoll schon von vielen gängigen Servern unterstützt.

Canvas

Canvas erlaubt in HTML5 das Zeichnen von Rastergraphiken, einfache Animationen, Kompositionen von Rastergraphiken und Fotos im 2-dimensionalen Raum.

Für das Zeichnen von Graphiken stehen Javascript API-Funktionen (Application Programming Interface Funktionen) für Rechtecke, Bögen, Pfade und Linien zur Verfügung. Es ist auch möglich Bé Zierkurven darzustellen. Darüber hinaus kann man die Transparenz, Linienstärke, Farben und Gradientenübergänge angeben. Zusätzlich stehen die Standardtransformationen wie Translation, Rotation und Skalierung zur Verfügung und können ebenfalls auf gezeichnete Graphiken oder Bilder wie zum Beispiel, Fotos angewandt werden. Durch Kompositionsoperationen können zwei überlappende Flächen

verschiedenartig dargestellt werden. Dabei können überlappende Fläche in anderer Farbe dargestellt werden, oder es wird nur ein Körper, dessen Flächen sich nicht mit dem anderen Körper überschneidet, gezeichnet. In Summe gibt es 12 verschiedene Kompositionsoperationen. Es kann auch explizit ein clipping Pfad angegeben werden, welcher bestimmte Flächen aus dem Canvas Element ausschneidet.

Einfache Animationen können mit der Funktion *setInterval* ausgeführt werden, welche als Parameter eine Funktion und ein Zeitintervall entgegennimmt. Die übergebene Funktion wird zyklisch mit dem angegeben Zeitintervall ausgeführt, damit lassen sich die einzelnen Frames der Animation zeichnen. Grundsätzlich besteht eine Animation mit Canvas aus vier Schritten:

- Jedes Canvas Element, welches Teil der Animation ist, muss in jedem neuen Frame zuerst gelöscht werden.
- Falls bei jedem neu gezeichneten Frame der gleiche Canvas Status verändert werden soll, aber der Status innerhalb eines Frames für gewisse Elemente verändert werden muss, werden zwei Funktionen (*save* und *remove*) angeboten. Mit *save* wird der Canvas Status auf einen gewissen Stack gelegt, danach kann der aktuelle Canvas Status verändert und benutzt werden. Mit der Funktion *restore* kann der letzte gespeicherte Canvas Zustand wieder vom Stack geholt werden. Zum Canvas Status zählen Einstellungen der Transformation (Translation, Skalierung und Rotation), Clipping Region und Attribute wie zum Beispiel, Linienstärke, Fülleigenschaften, etc.
- Im dritten Schritt werden die einzelnen animierten Elemente gerendert.
- Vorausgesetzt man hat in Schritt zwei den original Canvas Zustand mit *store* am Stack gespeichert, kann man als letzten Schritt mithilfe der Funktion *restore* den original Canvas Zustand wiederherstellen.

SVG

SVG (Scalable Vector Graphics) baut auf XML auf und wird von den meisten aktuellen Browsern unterstützt, wenn auch nicht die komplette SVG Spezifikation [Dahlström et al., 2011] implementiert ist. Grundsätzliche Vorteile von skalierbaren Vektorgraphiken gegenüber den Pixelgraphiken ist die Möglichkeit, unbeschränkt die Graphik zu vergrößern ohne Einbußen in der Auflösung hinnehmen zu müssen.

SVG wird nicht vom Internet Explorer 8 unterstützt, welcher sich unter Windows XP aber nicht auf eine neuere Version updaten lässt. SVG Graphiken können direkt in der HTML Datei mit dem <svg>Element eingebettet werden. Wie beim Canvas Element unterstützt SVG grundlegende Graphikelemente wie zum Beispiel, Rechtecke, Kreise, Ellipsen, Linien, Polygone, Texte und Pfade. Wobei die Pfade, die am meisten verwendeten und auch flexibelsten Elemente darstellen und unter anderem auch Bézier Kurven darstellen können. Pfade können auch als clipping Elemente definiert werden. Des Weiteren können Linienstärke, Füllelemente, Gradientenverläufe und Muster sowie Rastergrafiken eingebunden und verwendet werden. Auf die graphischen Elemente, welche auch gruppiert werden können, kann Rotation Skalierung, Translation und Scherung als Form der Transformation angewandt werden. Jedem Element kann auch ein Wert für die Lichtdurchlässigkeit (engl.: opacity) zugewiesen werden, um damit die Transparenz zu steuern. Unter anderem stehen auch Maskierungs- und Filteroptionen zur Verfügung.

Animationen können, wie auch mit dem Canvas Element, mittels Scripting ausgeführt werden. Dabei kann ebenfalls die Funktion *setInterval* verwendet werden. Vorteil von SVG gegenüber Canvas ist, dass der Inhalt des Elementes nicht gelöscht, sondern nur mit den neuen Daten aufgefrischt werden muss. Weitere Möglichkeiten eine Animation mit SVG zu implementieren ist die Verwendung von SMIL (Synchronized Multimedia Integration Language), welche seit 2008 eine W3C Empfehlung ist, oder CSS Animations.

Grundsätzlich sind die angeboten Möglichkeiten von HTML5 für SVG weitreichender als jene von Canvas.

4.4.2 Javascript

Javascript ist eine Skriptsprache die von allen großen Browserherstellern verwendet wird, um den Inhalt, welcher mit HTML geschrieben wird, Logik hinzuzufügen. Dabei bietet die Sprache funktionale und objektorientierte Programmierparadigma. Jedoch unterscheidet sich Javascript in der Objektorientiertheit sehr stark von klassenorientierten Programmiersprachen wie Java oder C++. Dieser Absatz zeigt einen kleinen Ausschnitt der wichtigsten Merkmale von Javascript, welche teilweise für die vorliegende praktische Arbeit verwendet wurden.

Objekte in Javascript

Objekte sind in Javascript grundsätzlich Datentypen, welche aus einem oder mehreren primitiven Werten und/oder aus anderen Objekten bestehen können. Diese Daten können

über Namen (Strings) gespeichert und wieder abgerufen werden. Damit entspricht ein Objekt in Javascript grundsätzlich einem assoziativen Array als Datenstruktur, welche als String-Werte Paare gespeichert sind. Jedoch ist ein Objekt in Javascript mehr als nur ein assoziatives Array, denn bei jeder Initialisierung eines neuen Objektes erbt das Objekt die Eigenschaften eines anderen übergeordneten Objektes. Dieser Mechanismus ist auch als prototypenbasierte Programmierung (engl. prototype based programming oder prototypal inheritance) bezeichnet und baut auf dem Prototyp Entwurfsmuster (Gamma et al. [1994]) auf. Dabei werden die Objekte nicht zur Kompilierzeit (wie in Java oder C++ üblich) erstellt, sondern erst während der Laufzeit. Die Objekte können dynamisch um weitere Eigenschaften erweitert bzw. gegebene Eigenschaften verändert werden. In Javascript sind nahezu alle Datentypen Objekte (auch Funktionen) bis auf Strings, Zahlen, True, False, Null oder Undefined. (Flanagan [2011])

Closures

Closures stellen in Javascript eine wichtige Spracheigenschaft dar. Dabei wird das Prinzip des *lexical scoping* angewandt. Unter *scope* versteht man einen Sichtbarkeitsbereich, in dem die Variable über einen definierten Namen verfügbar ist. *Lexical scoping* heißt in diesem Fall, dass bei einem Funktionsaufruf der gleiche Variablen Sichtbarkeitsbereich definiert ist, wie zum Zeitpunkt des Definierens der Funktion. Das Codebeispiel Listing 4.1 zeigt einen Funktionsaufruf von *checkscope*, welcher implizit die verschachtelte Funktion f aufruft. Diese liefert als Return-Wert die lokale Variable der Funktion *checkscope*. Diesen Return-Wert würde auch ein C-Programm liefern.

```
1  var scope = "global scope";   //globale Variable
2  function checkscope() {
3    var scope = "local scope";  //lokale Variable der Funktion checkscope()
4    function f(){       //verschachtelte Funktion
5      return scope;
6    };
7    return f();
8  };
9  checkscope();         //Funktionsaufruf
10                     //Ergebnis: "local scope"
```

Listing 4.1: Closure Prinzip [Flanagan [2011], Seite 181]

Der Code in Listing 4.2 würde mit dieser Art der Programmierung in einem C-Programm die globale Variable liefern, jedoch nicht mit Javascript. Hier gilt der Sichtbarkeitsbereich der Variablen, so wie die Funktionen definiert wurden. Das heißt, die verschachtelte Funktion wurde definiert mit der Variablen *var scope = "local scope"* und dieser Bereich ist nach wie vor gültig, ganz egal wann und wo die Funktion *f()* aufgerufen wird.

```
1  var scope = "global scope";   //globale Variable
2  function checkscope() {
3    var scope = "local scope";  //lokale Variable der Funktion
4    function f(){
5      return scope;     //verschachtelte Funktion
6    }
7    return f;          //return Wert ist die Funktion selbst
8  };
9  checkscope()();          //Aufruf von checkscope und Aufruf von f()
10                  //Ergebnis: "local scope"
```

Listing 4.2: Closure Prinzip [Flanagan [2011], Seite 181)]

Listing 4.3 zeigt ein Beispiel wie Closures effektiv in Javascript eingesetzt werden können. Es ist damit möglich, Variablen in Funktionen *private* zu definieren. Damit ist es nicht mehr möglich, die Variablen direkt von außen abzufragen bzw. zu ändern. Dies ist nur mehr mit verschachtelten Methoden (wie in Listing 4.3 mit *increment()* und *decrement()* möglich. Dies Sprachfunktionalität wird auch bei Vererbung von Objekten verwendet, um gewissen Daten von außen nicht zugänglich zu machen. (Flanagan [2011])

```
1  var count = 1000;   //globale Variable
2  function counter() {
3    var count = 0;    //private Variable, kann von aussen nicht zugegriffen werden
4    return {
5      increment: function() { //public Funktion
6        return ++count;
7      },
8      decrement: function() { //public Funktion
9        return --count;
10     }
11   }
12 };
13 var c1 = counter(); //c1 hat seine eigene count Variable
14 var c2 = counter(); //c2 hat seine eigene count Variable
15 console.log(c1.increment());  //Ergebnis: 1
16 console.log(c2.decrement()); //Ergebnis: -1
17 console.log(c1.count);    //Variable nicht erreichbar -> private
```

Listing 4.3: Closure Prinzip

Vererbung in Javascript

Aufgrund des Closures Prinzips ist es in Javascript möglich, ähnlich wie in klassenorientierten Sprachen Vererbung zu benutzen. Wie Eingangs schon unter 4.4.2 erwähnt, leiten sich alle neu instanzierten Objekte von ihrem übergeordneten Objekt ab. Dafür können in Javascript zwei Sprachkonstrukte verwendet werden. Zum einen gibt es einen Konstruktor, welcher bei der Objekterzeugung mit dem Schlüsselwort *new* aufgerufen wird. Zum anderen hat jedes Objekt einen Prototypen mit gewissen Eigenschaften. Jedes Objekt erbt bei der Erzeugung den Prototypen, welcher ebenfalls gewisse Eigenschaften besitzt, des übergeordneten Objektes. Mit den Closure- und Vererbungsprinzipien, lassen sich Objekthierarchien aufbauen, welche den Klassenhierarchien sehr ähnlich sind. Zum Beispiel

ist es möglich *private*, *public* Variablen, sowie *private*, *public* und *privileged* Funktionen bzw. Methoden zu erzeugen. Listing 4.4 zeigt ein einfaches Codebeispiel, wie Vererbung in Javascript implementierbar ist. Im Anhang A werden weitere Beispiele gezeigt, welche Eigenheiten im Vergleich zu klassenorientierten Programmiersprachen aufzeigen.

```
1  function Parent() {       //Konstruktor des Elternobjektes
2    var privateVar_ = "private variable";   //private Variable
3    this.publicVar = "public variable"; //public Variable
4    this.init = function() {  //Privileged Function -> erlaubt Zugriff auf private Variablen
5      console.log(privateVar_);
6      this.printText(); //es kommt darauf an, welches Objekt die init Funktion ruft.
7    };
8    this.printText = function(){};    //dummy Funktion, damit der this Zeiger nie auf das window Objekt zeigen. Auch
         nicht im Falle, dass Kind-Objekte die Funktion "printText" nicht implementiert haben.
9  };
10 Parent.prototype.publicFunction = function(){ //public Funktion des Parent-Objektes
11   console.log("I am a public function");
12   console.log(this.publicVar);
13 };
14
15 Child.prototype = new Parent(); //fuer alle spaeter instanzierten Child-Objekte wird nur einmal der Parent-
       Konstruktor aufgerufen
16 Child.constructor = Child;  //Ohne diese Anweisung zeigt, aufgrund der vorhergehenden Anweisung, der Child-
       Konstruktor auf den Parent-Konstruktor
17 function Child() {
18   this.printText = function(){
19     console.log("Child Funktion");
20   };
21 }
22 var child = new Child();
23 child.init(); //Ergebnis: "private variable" und "Child Funktion"
24 child.publicFunction(); //Ergebnis: "I am a public function" und "public variable"
```

Listing 4.4: Vererbung

Method Chaining

Method Chaining ist keine spezielle Funktion von Javascript, sondern eine gängige Methode für objektorientierte Programmiersprachen. Jedoch wird diese Methode sehr intensiv von Javascript Bibliotheken wie zum Beispiel von jQuery (siehe 4.4.3) und d3js (siehe 4.4.5) verwendet. Die Idee von Method Chaining ist, dass jede Methode eines Objektes, das Objekt selbst als Returnwert zurückliefert. Der Vorteil ist, dass zum einen wie im Falle von jQuery nur einmal eine Selektion durchgeführt werden muss, um dann alle anstehenden Operationen nacheinander durchzuführen. Zum anderen erhöht es auch die Lesbarkeit des Codes und minimiert diesen gleichzeitig.

4.4.3 jQuery und jQuery UI

Das jQuery Framework hat sich in den letzten Jahren enorm verbreitet und wird heutzutage von vielen großen Firmen eingesetzt. Mit dem jQuery Framework ist es möglich, auf einfache Weise auf das DOM (Document Object Model) zuzugreifen und die Elemente des DOM zu verändern bzw. zu erweitern. Diese sogenannten Selektoren sind syntaktisch

an die CSS Selektoren angelehnt. Außerdem gibt es noch immer browserseitige Unterschiede, welche mithilfe des Frameworks gekapselt werden und somit für den Anwender ”versteckt” werden. jQuery unterstützt Method Chaining (siehe 4.4.2), indem jeder Funktionsaufruf eines jQuery Objektes wieder das spezielle jQuery Objekt zurück liefert. Das jQuery Framework unterstützt neben den Selektoren für das DOM auch das Event-Handling für Anwenderinteraktion. Es können Maus-Events mit bestimmten Elementen verknüpft und weitere Logik hinzugefügt werden. Ein weiteres Feature des Frameworks ist die Erweiterbarkeit durch Plug-Ins von jedermann. Ein weitläufig anerkanntes und nützliches Plug-In ist das jQuery-UI-Plug-In, welches das jQuery Framework mit Drag and Drop-, Datepicker-, Tabs- und vielen anderen Userinterface Elementen erweitert.

Ebenfalls Bestandteil der Library ist die Unterstützung von AJAX (Asynchronous Javascript and XML). AJAX basiert auf XMLHttpRequests, welche asynchrone Daten vom Server für den Client anfragen können. Wird eine Anfrage vom Client an den Server gestellt, muss der Client nicht auf dessen Antwort warten, infolgedessen kann der Anwender weiterhin mit dem Client interagieren. Wenn die Antwort vollständig vom Server an den Client übermittelt wurde, wird eine vom Programmierer bekanntgegebene Callback-Funktion aufgerufen, und die gesendeten Daten können verarbeitet werden. Ein Vorteil dieser Vorgehensweise ist, dass die HTML-Seite und deren statische Elemente nicht neu an den Client geschickt werden müssen, und damit die Netzwerkkommunikation und Serverbelastungen gering gehalten werden. Üblicherweise wird einfacher Text, HTML, JSON, Javascript oder XML mittels jQuery-AJAX Anfragen unterstützt. Wichtig in diesem Zusammenhang ist noch zu erwähnen, dass mit AJAX die sogenannte Same-Origin-Policy gilt. Das heißt, es dürfen nur Anfragen an den gleichen Server gestellt werden, von welchem auch die Javascript Datei gesendet wurde. Dieses Sicherheitsmerkmal wird von allen gängigen Browsern unterstützt.

tipsy - jQuery Plugin

Das jQuery Plug-In ist eine kleine Javascript Bibliothek [Frame, 2012], welche es ermöglicht Tooltips im Browser anzeigen zu lassen. Die Bibliothek bietet Einstellmöglichkeiten hinsichtlich der Position des Tooltips, sowie Fade-In und Fade-Out Varianten. Außerdem kann der angezeigte Text innerhalb des Tooltips als HTML Text definiert werden. Somit ist es möglich, innerhalb des Tooltips weitere Verlinkungen anzugeben.

4.4.4 Javascript Object Notation - JSON

JSON (siehe auch http://json.org/json-de.html) ist ein Datenformat, welches für Maschinen leicht zu parsen, aber auch für Menschen lesbar ist. JSON ist Programmiersprachen unabhängig und dadurch für den Austausch von Daten zwischen zwei verschiedenen Systemen geeignet. Die Struktur des JSON Formats entspricht der Struktur von Objekten in Javascript. Auch JSON Objekte bestehen aus Name(String)/Werte Paaren. Als Wert kann wiederum ein Objekt angegeben werden oder aber auch zum Beispiel Arrays, Numbers, boole'sche Werte und null.

Es gibt mehrere Open-Source Bibliotheken, um Java Objekte in JSON Objekte umzuwandeln bzw. zu serialisieren. In dieser Arbeit wird mit der *google-gson* Bibliothek gearbeitet. Weitere Informationen und eine Beschreibung der API (Application Programming Interface) sind unter http://code.google.com/p/google-gson/ beschrieben.

4.4.5 Vergleich der Javascript Bibliotheken für Informations Visualisierung

Wie in Kapitel 2 ausführlich beschrieben, welche Arten der Informationsvisualisierung für die vorhandenen Daten in Frage kommen können, wurde für den praktischen Teil eine Internetrecherche durchgeführt, um vorhandene Bibliotheken zu finden und zu evaluieren. Dabei sind in Bezug auf Javascript zwei Libraries gefunden worden, welche hier kurz näher erläutert werden. Generell kann gesagt werden, dass die beiden Bibliotheken zwei unterschiedliche Strategien verfolgen. Die eine Bibliothek verwendet Canvas und eine klassenhierarchische Implementation, die andere bedient sich der SVG Elemente und einem Selektionsmechanismus ähnlich dem der in jQuery zur Anwendung kommt.

Javascript InfoVis Toolkit

Das Javascript InfoVis Toolkit [Belmonte, 2012] baut auf dem HTML-Canvas Element (siehe 4.4.1) auf. Das Toolkit beinhaltet eine Canvas Klasse, welche die Basis für alle Visualisierungen ist. Mithilfe dieser Klasse kann das Canvas Element, unter Angabe von verschiedenen Optionen wie zum Beispiel, die Höhe oder Weite, in die HTML Datei eingebettet werden. Die Bibliothek bietet verschiedene Visualisierungsoptionen von Knoten und Kanten über Effekte, Events, Navigation und vielen mehr, in Form von Klassen, an. Diese Klassen besitzen verschiede Eigenschaften, welche der Anwender zu seinen Gunsten verändern kann. So bietet die Knotenklasse beispielsweise Eigenschaften zum

Ändern der Knotenfarbe, Transparenz oder Form an. Die von der Bibliothek angebotenen Visualisierungsarten reichen von verschiedenen Diagrammtypen (z.B. Balken- oder Tortendiagramm) über TreeMaps, Force-Directed-Layouts, radiale Graphen und vielen anderen Darstellungsarten. Das heißt, für die vorliegende praktische Anwendung deckt die Bibliothek alle theoretischen Darstellungen ab.

Die Daten, welche von der Bibliothek verarbeitet werden, müssen im JSON Format aufbereitet sein. Je nachdem, ob man eine Baum- oder Graphenstruktur darstellen will, gibt es verschiedene Formate. Wobei das Format für die Graphenstruktur eine Erweiterung der Baumstruktur ist. Listing 4.5 zeigt das Format für Graphenstrukturen. Somit ist ein Knoten aus einer eindeutigen ID, einem Namen, benutzerdefinierten Daten und den benachbarten Knoten definiert. Bei der JSON Struktur für Bäume werden die ID, der Name, benutzerdefinierte Daten und die Kinder angegeben.

```
1  var json = [
2    {
3      "id": "aUniqueIdentifier",
4      "name": "usually a nodes name",
5      "data": {
6        "some key": "some value",
7        "some other key": "some other value"
8       },
9      "adjacencies": ["anotherUniqueIdentifier", "yetAnotherUniqueIdentifier", 'etc']
10   },
11
12   'other nodes go here...'
13 ];
```

Listing 4.5: Javascript Infovis Toolkit JSON Format [http://thejit.org/static/v20/Docs/files/Loader/Loader-js.html Zuletzt besucht am: 2012.08.30)]

Die gegenwärtig aktuellste Version von Javascript InfoVis Toolkit lautet 2.0.1 und wurde am 5.Mai 2011 veröffentlicht.

Data Driven Documents - D3 Javascript Bibliothek

Die Daten Driven Documents – D3 Bibliothek [Bostock, 2012a] baut auf HTML SVG Elementen auf. Wie in Abschnitt 4.4.1 beschrieben, bietet HTML SVG, im Vergleich zu Canvas, mehr Möglichkeiten in Bezug auf graphische Elemente sowie deren Animation. Des Weiteren basiert SVG auf XML Elementen, welche im HTML Dokument eingebettet sind. Deshalb liegt es Nahe, dass diese Elemente, ähnlich wie in jQuery bzw. CSS3 (Cascading Style Sheets), mit Selektoren ausgewählt und verändert werden. Listing 4.6 zeigt die grundsätzliche Verwendung von D3 in Bezug auf Selektoren, welche HTML Elemente erzeugen und verändern können. Der Code in Listing 4.6 fügt einen Kreis und ein Rechteck als SVG Grafik hinzu. Beiden Elementen werden Attribute hinzugefügt und

wieder abgefragt. Die letzte Anweisung in Listing 4.6 zeigt eine simple Animation, welche den zuvor gezeichneten Kreis auf eine neue Position versetzt. Dabei wird zuerst eine Sekunde gewartet bevor die Animation, welche mit zwei Sekunden Dauer angegeben ist, gestartet wird.

```
1  window.onload = function(){
2    var circle = d3.select("body")  //Selektion mit Tag Element
3      .append("svg")          //dem "body" Tag wird ein "svg" Tag hinzugefügt
4      .append("circle")       //dem "svg" Tag wird ein "circle" Tag hinzugefügt
5      .attr("cx", 90)         //hinzufügen von Attributen an das "circle Element
6      .attr("cy",90)
7      .attr("r",30)
8      .attr("fill","red")
9      .attr("id", "circleElement");
10   var test1 = d3.select("[fill = red]"); //Selektion mittels Attributen
11   console.log(test1.attr("id"));       //Ergebnis: "circleElement"
12   var circleData = d3.select("svg").     //Daten an das "circle" Element hinzufügen
13     selectAll("circle")
14     .data([50,100]);
15   var enter = circleData.enter().append("rect");  //Daten die keinem vorhandenen Element zuordenbar sind, können
         mit enter() einem neuen Element zugeordnet werden
16   enter.attr("width", function(d){
17         return d;})
18     .attr("height", 20)
19     .attr("id","rectElement");
20   circle.attr("r", function(d){
21     return d;
22   });
23   var test2 = d3.select("svg rect");  //Selektion mittels Hierarchie "parent child"
24   console.log(test2.attr("id"));  //Ergebnis: "rectElement"
25   d3.selectAll("#circleElement").transition().delay(1000).duration(2000).attr("cx", 300).attr("cy", 300);
26 };
```

Listing 4.6: Selektoren und grundsätzlich Nutzung von D3

Die aktuelle Version der Daten Driven Documents – D3 Bibliothek lautet 2.10.0 und wurde im August 2012 veröffentlicht. Im Vergleich zum Javascript InfoVis Toolkit wird an der D3 Bibliothek aktiv gearbeitet und ständig erweitert.

Kapitel 5

Implementierung

5.1 Kapitelüberblick

Im vorhergehenden Kapitel wurde die Technologieauswahl und einige Eigenschaften der ausgewählten Technologien beschrieben. Dieses Kapitel setzt auf das letzte Kapitel auf und beschreibt im Detail wie die Software für die Visualisierung implementiert wurde. Dabei wird auf das Einlesen der Rohdatendatei und die darauf folgende interne Datenverwaltung eingegangen. Außerdem wird in diesem Kapitel der typische Datenfluss der Software nach einer Interaktion des Anwenders beschrieben.

5.2 System Architektur

Der grundsätzliche Aufbau des Systems besteht aus einem Server und einem oder mehreren Clients. Die Kommunikation zwischen Client und Server wird mittels HTTP Protokoll realisiert. Wobei zu unterscheiden ist, ob eine komplette Seite vom Anwender angefordert wird (mittels HTTP Anfrage), oder ob die Seite schon geladen wurde und nur noch Daten, in Bezug auf die Visualisierung, asynchron nachgeladen werden (mittels AJAX Anfrage).

5.2.1 Entwicklertools

Zur Systemimplementierung wurde Eclipse mit Java EE Unterstützung als Editor verwendet. Eclipse unterstützt die Einbettung des Tomcat Servers. Somit ist sichergestellt, dass der serverseitige Code auch mithilfe von Breakpoints und anderen Debugging Funktionalitäten analysiert werden kann. Für die Erstellung des clientseitigen Codes wurde

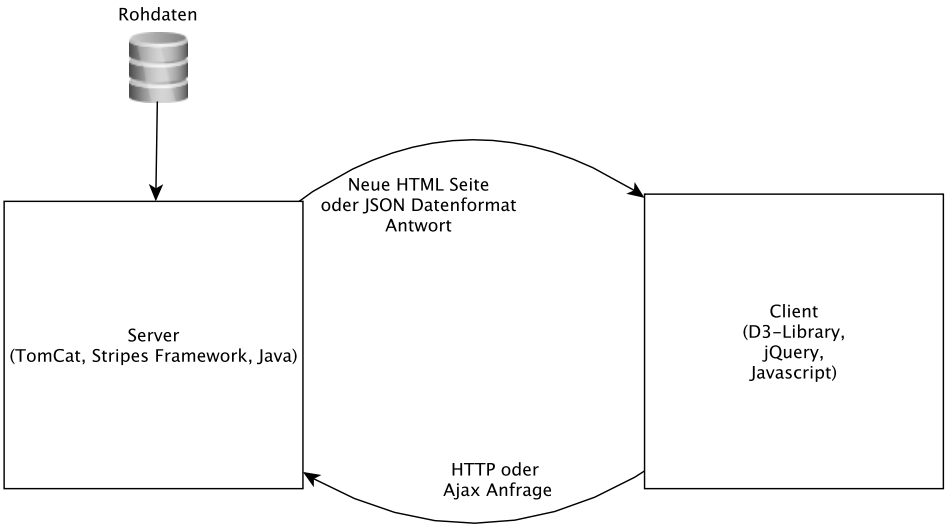


Abbildung 5.1: Systemübersicht

ebenfalls Eclipse, jedoch zu Debugging Zwecken der Chrome Browser und dessen inkludierten Entwicklungstools verwendet. Damit ist es ebenso möglich, Breakpoints und Variablenanalysen auf der Javascript Seite durchzuführen. Des Weiteren kann die HTTP Kommunikation und auch der dynamische Aufbau von HTML Elementen online überprüft werden.

5.3 Server

5.3.1 Überblick

Wie schon unter 4.3 beschrieben, wird auf der Seite des Servers ein Apache Tomcat Server verwendet, um Java Code in Java Servlets und Java Server Pages zu benutzen. Des Weiteren ist es möglich, das unter 4.3.2 beschriebene Stripes Framework mit einzubinden und verwenden zu können. Da die Rohdaten in Form von einer Excel Datei vorliegen, wurden diese in ein CSV Format umgewandelt, damit das Java Programm die Daten automatisch einlesen kann. Die Daten in der CSV Datei werden von dem Programm ausgelesen und in eine logische interne Struktur des Java Programmes abgelegt. Abbildung 5.2 zeigt den Ablauf innerhalb des Servers. Wie in Abbildung 5.1 abgebildet, wird über den Client eine HTTP Anfrage gestellt und der Server antwortet mit einer HTTP Antwort, welche auch ein JSON Format bei einer AJAX Anfrage sein kann.

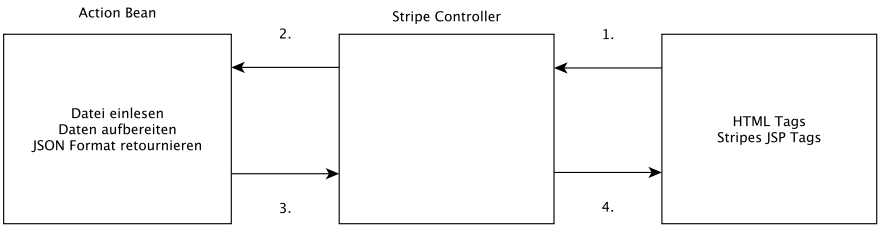


Abbildung 5.2: Server Übersicht

5.3.2 Einlesen der Rohdaten

Die CSV-Datei wird Zeilenweise eingelesen. Jede Zeile entspricht einer Wirkkette, welche aus einem Problemfeld, einem Szenario, einer Annahme, einer Anforderung, mehreren Ebenenfeldern und einem Feld, welches die Verwandtschaft mit einer anderen Anforderung, beinhalten kann (siehe Abbildung 3.1). Auch in der serverseitigen Software findet sich diese Struktur wieder. Das heißt, es gibt ein Objekt namens Wirkkette, welches die aufgezählten Elemente beinhaltet. Da der Inhalt der Elemente einer Wirkkette auch in anderen Wirkketten vorkommen kann, muss beim Einlesen darauf geachtet werden, Inhalte nicht mehrfach zu speichern. Konkret bedeutet dies, dass ein Mechanismus vorgesehen sein muss, der bei Elementen, welche öfter vorkommen, diese nur einmal speichert und von allen Wirkketten, welche diese Elemente beinhalten, referenziert werden kann. In der Software wird dies erreicht, indem der Inhalt für jede einzulesende Spalte mit einer korrespondierenden HashMap verglichen wird. Wenn der Inhalt noch nicht vorgekommen ist, wird ein neues Objekt angelegt und auch in der HashMap eingetragen. Somit kann sichergestellt werden, dass sich wiederholender Inhalt in der Rohdatendatei, nur einmal in der internen Struktur des Java Programmes gespeichert wird.

Klassendiagramm

Abbildung 5.3 zeigt die Implementierung in Bezug auf das Einlesen und interne Speichern der Rohdaten im Detail. Dabei ist die Funktion *readFile()* in der Klasse *CSVStore* von Bedeutung. Die Funktion öffnet die Rohdatendatei und liest diese zeilenweise ein. Dies wird unter zu Hilfenahme einer externen Java-Bibliothek namens *opencsv* (`http://opencsv.sourceforge.net/` Zuletzt besucht am: 2012.05.03) ausgeführt.

Pro Zeile und Zelle wird mittels der privaten HashMaps (*anforderung*, *annahme*, etc) in der Klasse *CSVStore* festgestellt, ob der aktuell gelesene Rohdateninhalt schon einmal eingelesen wurde. Je nach Ergebnis wird entweder ein neues Objekt, entsprechend

der Spalte instanziert, oder aus der HashMap entnommen und dem aktuellen Wirkketten Objekt zugewiesen.

Pro Zeile wird ein neues Wirkketten Objekt instanziert und zu einer Liste (*Wirkkette*) hinzugefügt. Des Weiteren halten alle Spalten Objekte (z.B. *Anforderung*, *Annahme*) die Information zu welcher Wirkkette diese gehören. Aus Platzgründen wurden in der Abbildung 5.3 nur vier Objekte (*Annahme*, *Anforderung*, *Prolemfeld* und *Szenario*) dargestellt. Das zu analysierende Dokument enthält jedoch mehr Spalten. Außerdem kann mithilfe dieser Struktur das Programm leicht erweitert werden, um beliebig viele Spalten einzulesen. Am Ende dieses Prozesses wird die Liste mit allen darin befindlichen Wirkketten an den Funktionsaufrufer retourniert. Die *readFile()* Funktion sollte nicht direkt aus einem Action Bean aufgerufen werden, da diese Objekte nur für die Zeit des AJAX-Requests im Speicher angelegt werden. Das heißt, es müsste bei jedem AJAX-Request immer von neuem die Rohdatendatei eingelesen und in die interne Struktur gebracht werden. Um dies zu vermeiden wird ein Singleton verwendet, welcher nach einmaligem Aufruf der *readFile()*, die retournierte Struktur speichert und auf Anfrage von außen weiterleitet. Dadurch ist gewährleistet, dass die Datei nur einmal gelesen und verarbeitet werden muss.

Objektstruktur

Abbildung 5.4 zeigt einen typischen Aufbau der Wirkketten Listenstruktur. Die gelben Kreise stellen Java Objekte dar und die Pfeile die Verlinkungen untereinander. Die Abbildung 5.4 zeigt zum Beispiel, dass das Szenario1 der Wirkkette1 und der Wirkkette2 zugeordnet sind. Dasselbe gilt für das Objekt mit dem Namen Problemfeld1. Die bidirektionalen Pfeile zeigen die Verbindung zum einen, dass jede Wirkkette mit den ihr ermittelten Daten verbunden ist, zum anderen aber auch, dass jedes Objekt den Bezug zu einer oder mehreren Wirkketten besitzt. Diese Struktur hilft bei der Analyse beziehungsweise bei der Abfrage der Daten.

Strukturierung der Daten von Zell-Objekten

Zell-Objekttypen wie *Anforderung* oder *Annahme* besitzen mehrere Eigenschaften, die einerseits die Information der Rohdaten, also den textuellen Zelleninhalt enthalten, andererseits aber zusätzliche Informationen, die entweder für die Visualisierung oder zum Parsen des Datensatzes verwendet werden. Zum Beispiel hält die Objekteigenschaft *description* die Information wie sie auch in den Rohdaten spaltenweise eingetragen ist. Im Gegensatz dazu halten zum Beispiel die Objekteigenschaften *color* oder *position* Infor-

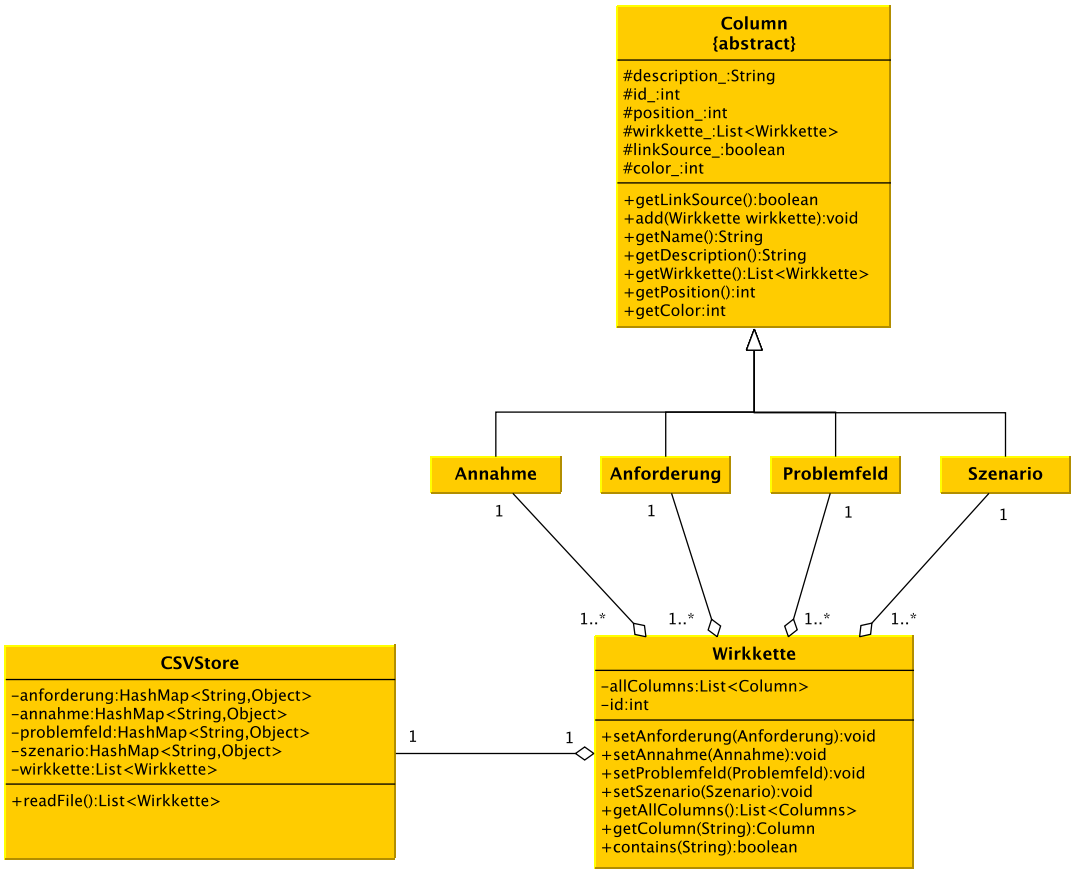


Abbildung 5.3: Systemübersicht zum Einlesen der Daten

mationen, welche nicht direkt aus der Rohdatendatei hervorgehen, jedoch für die Visualisierung benötigt werden.

5.3.3 AJAX-Request Funktionen

Die folgenden aufgezählten und beschriebenen Funktionen werden über AJAX-Requests vom Client aufgerufen. Alle Funktionen haben gemeinsam, dass diese die Antwort im JSON Format (siehe 5.4) an den Client senden.

Funktionsbeschreibung: problemfeldSelected(), szenarioSelected(), ebeneSelected()

Alle drei Funktionen bekommen drei Parameter (*pfSelection*, *szSelection*, *ebSelection*) vom Client über HTTP übertragen. Die Parameter beziehen sich auf die Auswahl, die der Anwender auf der jeweiligen HTML Seite (siehe 5.5.4) clientseitig getroffen hat. Der

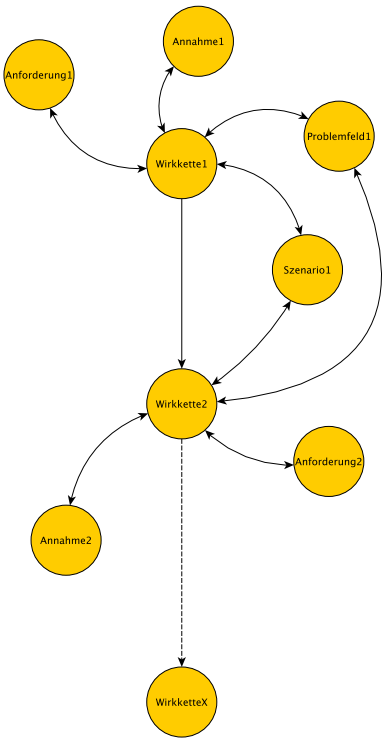


Abbildung 5.4: Objektansicht der Wirkketten Listenstruktur

Inhalt der Parameter ist entweder *no_selection*, wenn keine Auswahl getroffen wurde. Damit werden alle vorhandenen Möglichkeiten in Betracht gezogen und ausgewertet oder es steht ein konkreter Wert wie zum Beispiel, *Szenario1* oder *Arbeitsplatz der Zukunft* in dem jeweiligen Parameter. Da diese Texte in der internen serverseitigen Softwarestruktur einem Objekt eindeutig zuordenbar sind, kann die Software die dazugehörigen Wirkketten ermitteln und an den Client retourschicken. Im Falle der Abbildung 5.10 wurde vom Client die Funktion *problemfeldSelected* mit den zugeordneten Werten *aeltere Arbeitnehmer in Unternehmen*, *no_selection* und *no_selection* der Parameter *pfSelection*, *szSelection* und *ebSelection* aufgerufen.

Funktionsbeschreibung: anforderungSelected()

Diese Funktion wird auf den HTML Seiten, welche unter 5.5.4 beschrieben sind, aufgerufen. Dazu muss der Anwender auf einen graphischen Knoten, welcher eine Anforderung darstellt, klicken. Danach wird die Funktion *anforderungSelected* mit dem Beschreibungstext, welcher dem Zelleninhalt der Rohdatendatei entspricht, als Wert des *description* - Parameters mitübergeben. Mithilfe dieses Parameters werden alle Anforderungen, welche mit der selektierten Anforderung verwandt sind, mitsamt den zugehörigen Wirk-

ketten an den Client zurückgeschickt. Abbildung 5.12 zeigt eine mögliche Darstellung. Im Falle der Abbildung 5.12 wurde auf eine der Anforderungen geklickt, welche im rechten Fenster textuell aufgelistet sind.

Funktionsbeschreibung: getWirkkette()

getWirkkette() wird bei jedem MouseOver über einen graphischen Knoten, welcher eine Anforderung darstellt, auf den HTML Seiten beschrieben und unter 5.5.4, aufgerufen. Als Wert des Parameters *description* wird ebenfalls wie bei der Funktion *anforderungSelected()* der Beschreibungstext aus der Rohdatendatei mitgeliefert. Serverseitig wird aus der *description* der Anforderung die Wirkkette ermittelt und an den Client geschickt. Die Abbildung 5.11 zeigt im unteren Fenster die visualisierte Antwort des Servers. In diesem Beispiel wurde die Anforderungsbeschreibung:„Eine umfangreiche Sortier- und Filtermöglichkeit muss vorhanden sein, um relevante Informationen ohne Zeitverlust zu finden.” an den Server geschickt.

Funktionsbeschreibung: ebAssign()

Diese Funktion wird am Server aufgerufen, wenn der Anwender sich für die Ebenenzuteilung (siehe auch 5.5.5) entscheidet und dessen Seite aufgebaut werden soll. Der Funktion werden keine Parameter übergeben. Serverseitig findet eine Auswertung statt, indem jede Anforderung ihrer Ebenengruppe zugeordnet wird. Es gibt fünf verschiedene Ebenen *Mensch*, *Organisation*, *IT*, *Prozess* und *Methode* und jede Anforderung kann entweder keiner dieser Ebenen oder maximal allen fünf Ebenen zugeordnet werden. Dadurch ergeben sich 2^5 verschiedene Gruppierungen. Die Anforderungen werden je nach Eintragung im Rohdatenformat einer bestimmten Gruppe zugeordnet und dieses Ergebnis wird im JSON Format pro Knoten eingetragen, damit der Client die graphische Anordnung der verschiedenen Gruppen vornehmen kann (siehe Abbildung 5.14).

5.4 JSON - Format

5.4.1 Allgemein

Die Grundstruktur des JSON Formates wird durch die D3-Library (siehe 4.4.5) vorgegeben, kann aber mit zusätzlichen Informationen leicht erweitert werden. Nachfolgende

Beschreibungen sind auch als konkretes Beispiel in Abbildung 5.5 als debugging Fenster des Chrome Browser zusammengefasst.

Für die vorliegende Arbeit werden 3 JSON-Objekte *nodes*, *links* und *addData* bei AJAX-Anfragen vom Server an den Client zurückgesendet.

5.4.2 nodes - Objekt

Das nodes-Objekt besteht aus weiteren Elementen, welche nachfolgend kurz beschrieben werden.

name

Der Name entspricht dem Spaltennamen aus den Rohdaten. Somit kann der Name des Knotens beispielsweise Problemfeld, Szenario, Annahme oder einer der weiteren Spaltennamen aus der Rohdatendatei sein.

ebenenGroup

Diese Daten werden nur im Fenster ”Ebenenzuteilung” (siehe 5.5.5) verwendet. Da es 32 verschiedene Kombinationen aus Ebenenzuweisungen geben kann, wird in diesem Feld die jeweilige Gruppennummer übertragen.

color

Dieses Datenfeld gibt in Form einer Zahl die Farbe des darzustellenden Knotens an.

description

Unter der Beschreibung des Knotens ist der Inhalt der Zelle der Rohdaten gemeint. Aufgrund der Einleseprozedur (beschrieben unter 5.3.2) ist jede Beschreibung einzigartig. Das heißt, bei einer Antwort des Servers im JSON-Format werden keine zwei Knoten mit derselben Knotenbeschreibung geschickt.

position

Die Position gibt die Spaltenposition der Rohdaten, in Form einer Nummer, wieder. Damit wird es erleichtert, gleiche Spalten graphisch in der Nähe zu positionieren. Dies würde auch mit dem Namen der Spalte, welcher ebenfalls übertragen wird (siehe oben), funktionieren jedoch ist es softwaretechnisch etwas leichter mit Zahlen zu arbeiten im Gegensatz zu Strings.

hierarchy

Die Hierarchie wird numerisch übermittelt, wobei die innerste Position des radialen Graphen mit 0 angegeben wird und damit das Zentrum bildet. Dementsprechend werden die äußeren Radien in aufsteigender Reihenfolge durchnummeriert.

wkIndex

Da gleiche Knoten in mehreren Wirkketten vorkommen können, ist es wichtig alle zugehörigen Wirkketten in Form von Indizes zu übertragen. Die Information der Wirkketten Indizes wird zum Beispiel, beim Mouseover eines Knotens verwendet, um jene Pfade hervorzuheben, welche Teil des selektierten Knotens sind.

gleichNodes

Mit dem gleichNodes Objekt werden alle Knoten welche über die ”Gleich”-Spalte Ähnlichkeiten mit dem aktuellen Knotenobjekt aufweisen, übertragen.

5.4.3 links-Objekt

Ebenso wie das nodes-Objekt beinhaltet auch das links-Objekt mehrere Teilelemente auf die in den folgenden Punkten näher eingegangen wird.

source

Der Zahlenwert der in source eingetragen wird, entspricht der fortlaufenden Nummer der Knoten im nodes-Objekt. Mithilfe des target Elements kann somit eine gerichtete Verbindung beschrieben werden.

target

Der Zahlenwert der in target eingetragen wird, entspricht der fortlaufenden Nummer der Knoten im nodes-Objekt. Mithilfe des source Elements kann somit eine gerichtete Verbindung beschrieben werden.

value

Mithilfe des value Elements kann die Linienstärke der Links angegeben werden.

type

Das type Element gibt die Art der Linie an. Damit kann der Server vorgeben ob, zum Beispiel eine Linie durchgezogen (Schlüsselwort: ”solidLine”) oder strichliert (Schlüsselwort: ”dashedLine”) gezeichnet wird.

wkIndex

Mit dem wkIndex wird angegeben zu welcher Wirkkette der Link gehört.

5.4.4 addData Objekt

Dieses JSON Objekt ist für zusätzliche Daten gedacht und implementiert worden. In der vorliegenden Arbeit wird nur ein Wert übergeben.

pointGleichToCenter

Dieser bool'sche Wert gibt an, wenn es Verbindungen zu ähnlichen Anforderungen gibt (über die Gleich Spalte der Rohdaten definiert), ob diese zum Kreismittelpunkt oder von diesem weg gezeichnet werden sollen.


```
▼{,…}
 ▼addData: {pointChildrenToCenter:false}
    pointChildrenToCenter: false
 ▼links: [{source:0, target:1, value:15, type:solidLine, wkIndex:0},…]
  ▼0: {source:0, target:1, value:15, type:solidLine, wkIndex:0}
     source: 0
     target: 1
     type: "solidLine"
     value: 15
     wkIndex: 0
  ▼1: {source:1, target:2, value:15, type:dashedLine, wkIndex:1}
     source: 1
     target: 2
     type: "dashedLine"
     value: 15
     wkIndex: 1
  ▼2: {source:0, target:3, value:15, type:solidLine, wkIndex:2}
     source: 0
     target: 3
     type: "solidLine"
     value: 15
     wkIndex: 2
 ▼nodes: [{name:Problemfeld, ebenenGroup:0, color:7, description:Integration Menschentypen, position:1,…},…]
  ▼0: {name:Problemfeld, ebenenGroup:0, color:7, description:Integration Menschentypen, position:1,…}
     color: 7
     description: "Integration Menschentypen"
     ebenenGroup: 0
     hierarchy: 0
     name: "Problemfeld"
     position: 1
   ▶wkIndex: [0, 2]
  ▼1: {name:Anforderung, ebenenGroup:0, color:0,…}
     color: 0
     description: "Wissen muss persoenlich vermittelt werden;  Datenbanken, gespeicherte Projektkommunikation, Arbeitsvorschriften, I
     ebenenGroup: 0
   ▶gleichNodes: [{name:Anforderung, ebenenGroup:0, color:0,…}]
     hierarchy: 1
     name: "Anforderung"
     position: 5
   ▶wkIndex: [0, 1]
  ▼2: {name:Anforderung, ebenenGroup:0, color:0,…}
     color: 0
     description: "Wissen muss persoenlich vermittelt werden;  Datenbanken, gespeicherte Projektkommunikation, Arbeitsvorschriften, I
     ebenenGroup: 0
     hierarchy: 2
     name: "Anforderung"
     position: 5
   ▶wkIndex: [1]
  ▼3: {name:Anforderung, ebenenGroup:0, color:0,…}
     color: 0
     description: "Die zunehmende Individualisierung erfordert mehr Gesamtintegration der einzelnen Mitarbeiter. "
     ebenenGroup: 0
     hierarchy: 1
     name: "Anforderung"
     position: 5
   ▶wkIndex: [2]
```


Abbildung 5.5: Verwendete JSON Objekte

5.5 Client

5.5.1 Allgemein

Als Zielplattform für die Anwendung sind PC Systeme mit vielen gängigen Browsern wie zum Beispiel, Firefox Version 16.0.2, Safari Version 5.1.7, Chrome Version 21.0.1180.89 oder Opera Version 12.02 verwendet und getestet worden. Beim Firefox und beim Internet Explorer fiel auf, dass das Rendering der Graphik länger dauert und resourcenintensiver ist, als es bei den anderen Browsern der Fall ist. Dies hat mit der verwendeten D3-Javascript Library zu tun, da auch alle Demobeispiele, welche es auf Webseiten zu testen gibt, das gleiche Verhalten zeigen. Bis auf den Internet Explorer 9 wurden alle Browser unter MacOS 10.6 getestet und auch alle oben angegebenen Versionsnummern beziehen sich auf das MacOS Betriebssystem.

5.5.2 Überblick

Auf der Clientseite wird die Applikation auf mehrere HTML-Seiten aufgeteilt. Jede Seite repräsentiert eine andere Art der Visualisierung. Der Anwender kann mittels Buttons oder durch selektieren gewisser graphischer Elemente die Ansicht ändern bzw. mit der Mouse-Over Funktion zusätzliche Informationen erhalten. All diese Funktionen sind mittels Javascript implementiert.

Als Startseite der Anwendung dient ein sehr einfaches Fenster (siehe Abbildung 5.6), welches aus sechs verschieden Buttons besteht. All diese Buttons stehen für eine gewisse Visualisierungsart. Klickt man auf einen der Buttons wird über die Java Servlet Page (JSP) eine bestimmte Funktionalität am Server aufgerufen (siehe 5.3) und ausgeführt. Als Antwort sendet der Server Daten über Knoten und Links im JSON Format zurück.

Die Antwort des Servers (siehe 5.4) wird in den Aufbau der HTML Seite per Javascript eingebaut. Dabei ist das gesendete HTML File vom Server nahezu leer. Es beinhaltet lediglich Informationen über die Sektionen, wie der Bildschirm aufgeteilt wird (mittels JQuery UI siehe auch 4.4.3) und ein paar Bedienelemente in Form von Buttons oder Drop-Down Elementen. Der Rest der HTML Seite wird mittels Javascript dynamisch über das Document Object Model und Javascript Library Funktionen erzeugt, abhängig von den gesendeten JSON-Daten des Servers.

Alle Seiten bauen den graphischen Inhalt in einem HTML Element *div* mit dem ID Namen *chart* auf.

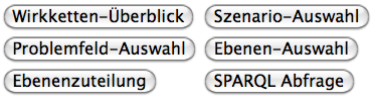


Abbildung 5.6: Startseite der Applikation

5.5.3 Wirkketten-Überblick Seite

Wird dieser Button geklickt, wird über einen AJAX Request die Funktion *getFromSelection* (siehe 5.3.3) mit drei Parameter (*problemfeld*, *szenario* und *hideColumns*) am Server aufgerufen. Aufgrund von voreingestellten Parametern wird die Seite wie in Abbildung 5.7 aufgebaut. Diese, beim ersten Aufruf der Seite, voreingestellten drei Parameter können vom Anwender nach dem erstmaligen Aufbau der Seite verändert werden. Entsprechend den Auswahlmöglichkeiten bezüglich Szenario, Problemfeld und ob gewisse Gruppen, mittels Checkboxen, ausgeblendet werden sollen (ebenfalls in Abbildung 5.7 ersichtlich). Das heißt, klickt der Anwender beispielsweise auf den Checkbutton ”Anforderungen”, wird ein neuer AJAX-Request an den Server gesendet, mit der Information im Parameter *hideColumns*, dass in der Antwort des Servers die Gruppe ”Anforderungen” nicht beinhalten soll. Das Ergebnis der Visualisierung ist in Abbildung 5.8 dargestellt. Mit dieser Art der Kommunikation wird die Logik auf der Clientseite minimiert und zusätzlich wird pro AJAX-Request nur soviel Information an den Client gesendet, welche augenblicklich dargestellt werden soll.

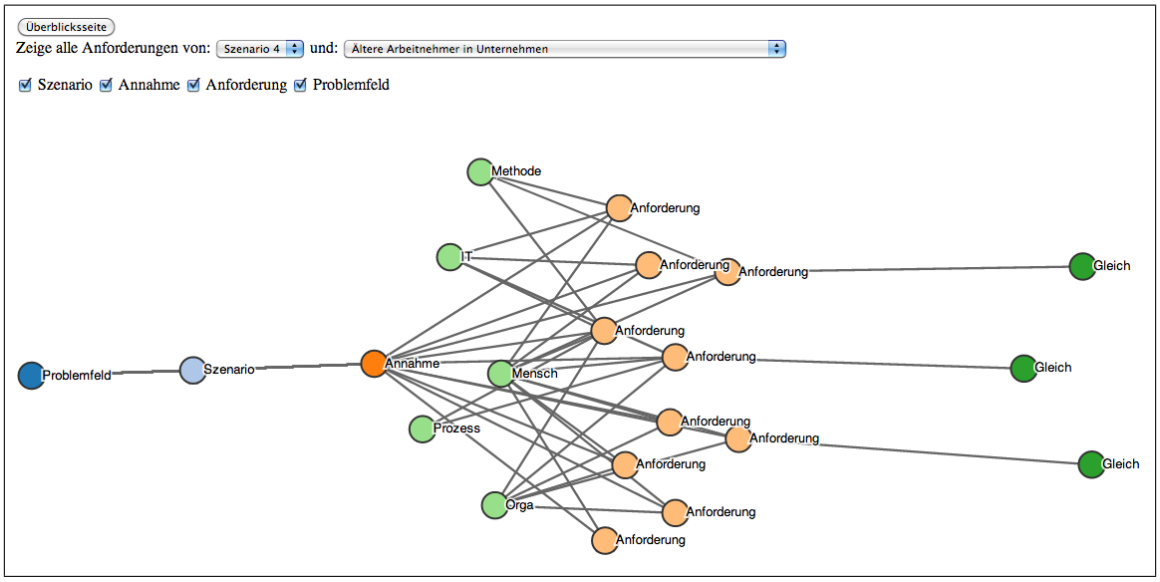


Abbildung 5.7: Wirkketten-Überblick

Die Antwort vom Server bezüglich des AJAX-Requests entspricht dem JSON-Aufbau

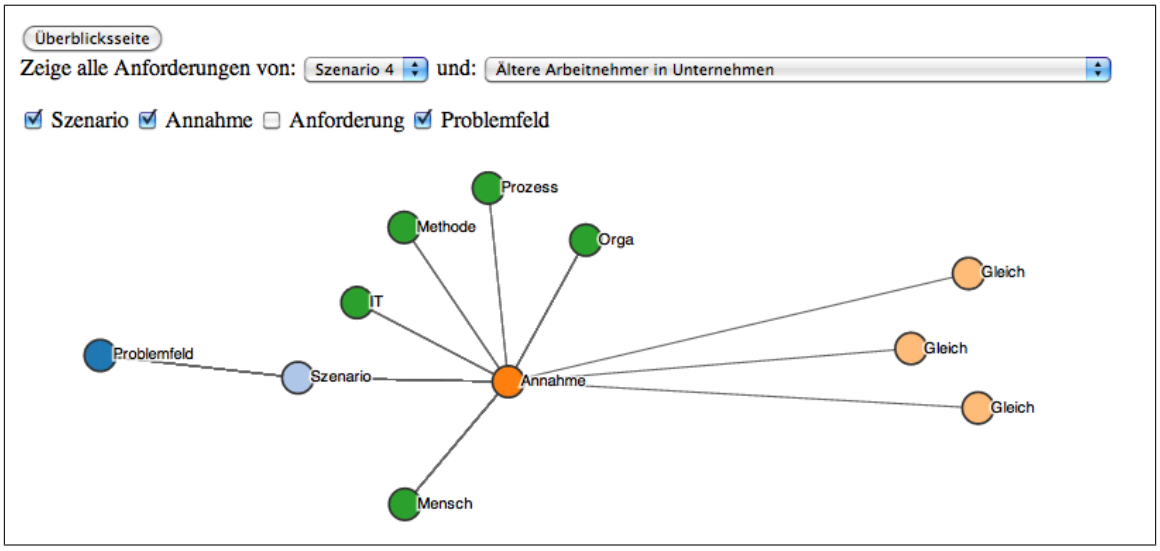


Abbildung 5.8: Wirkketten-Überblick ohne Anforderungen

wie unter Punkt 5.4 beschrieben. Im JSON-Objekt wird ein Feld ”Position” übermittelt. Im wesentlichen entspricht der Wert dieses Feldes der Spalte in der Rohdatendatei. Mithilfe dieses Wertes werden im Javascript Programmteil über das Force-Directed-Layout von der Data-Driven-Documents Library die Knoten an bestimmte x- und y-Koordinaten angenähert. Angenähert und nicht einem fixen Koordinatenpaar zugeordnet werden die Knoten deshalb, weil über das Force-Directed-Layout auch andere Parameter wie zum Beispiel, eine Abstoßungskraft zwischen den Knoten und ein Mindestabstand zu den Knoten angegeben wird. Das Force-Directed-Layout bietet darüber hinaus noch einige Parameter mehr, um die Eigenschaften des Layouts zu bestimmen [Bostock, 2012b], welche aber für die Wirkkette-Überblick Seite nicht verwendet wird.

In Abbildung 5.9 wird die Funktionalität bei einem MouseOver Event eines Knotens gezeigt. Dieses Event löst zwei graphische Darstellungen aus. Zum einen wird der Zelleninhalt der Rohdatendatei als Tooltip eingeblendet. Dafür wurde ein zusätzliches jQuery Plug-In names Tipsy (siehe 4.4.3) eingebunden und verwendet. Dieses Plug-In stellt mehrere Parameter zur Verfügung, um zum Beispiel die Position des Tooltips in Abhängigkeit des Tooltip Objektes anzugeben. Es ist auch möglich HTML Text in dem Tooltip zu übergeben, um dann mittels Referenzen auf andere Seiten zu verweisen. Die zweite graphische Darstellung bei einem MouseOver Event hebt alle Elemente, welche zu einer bestimmten Wirkkette gehören hervor, beziehungsweise trübt jene Elemente ein, welche nicht zur Wirkkette gehören. Um diese Funktionalität auf der Clientseite zu realisieren muss der Server über das JSON Objekt bestimmte Informationen liefern. Das JSON Objekt stellt dazu das *wkIndex* Element zur Verfügung. Dieses Element ist im ”links”-Objekt

als einfacher numerischer Wert und im ”nodes” Objekt als Array definiert. Daraus folgt, dass ein bestimmtes ”links”-Objekt immer nur einer bestimmten Wirkkette zugeordnet ist, jedoch ein ”nodes” Objekt mehreren Wirkketten zugeordnet werden kann. Näherers wird unter 5.3.2 beschrieben. Für die Implementierung auf der Clientseite ist es wichtig alle Elemente, welche die gleichen Nummern im *wkIndex* Element eingetragen haben, zu finden und dementsprechend darzustellen. Bei einem Element handelt es sich entweder um eine Verbindung zwischen den Knoten (*links*) oder den Knoten (*nodes*). Durch diese MouseOver Funktionalität ist es für den Anwender leichter den Teilgraphen zu erkennen und gleichzeitig detailliertere Informationen über einen bestimmten Knoten zu erfahren.

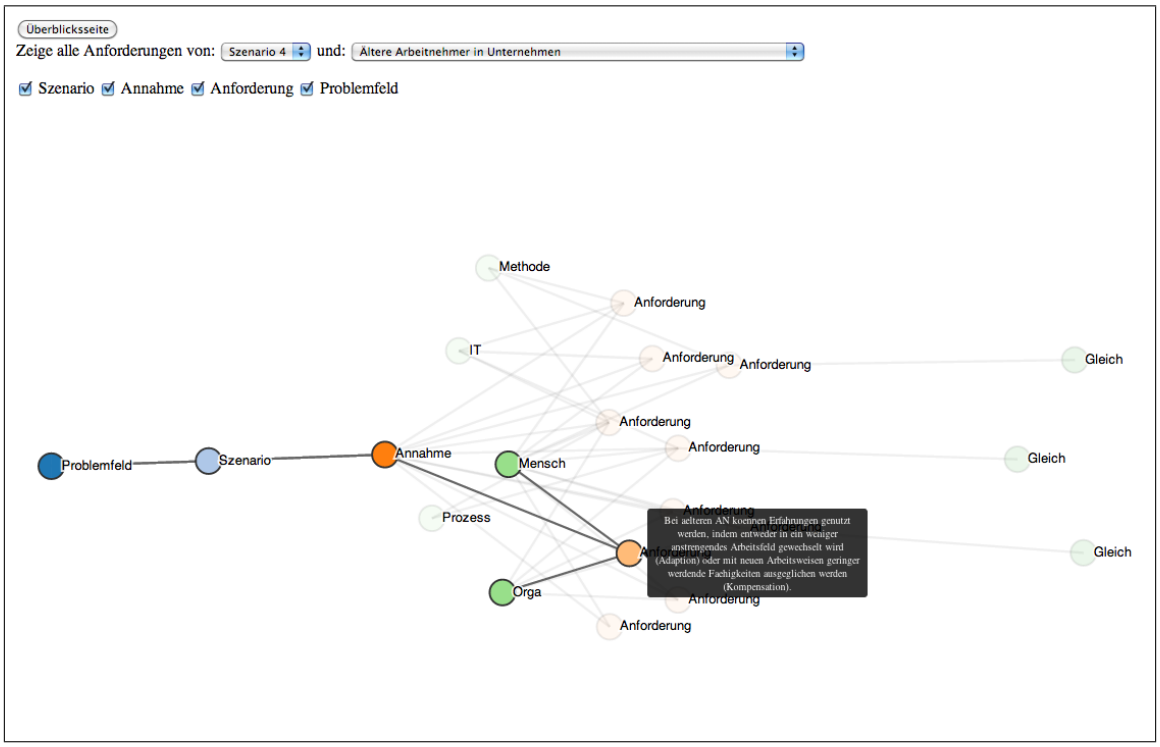


Abbildung 5.9: Wirkketten-Überblick mit MouseOver

5.5.4 Szenario-, Problemfeld- und Ebenen-Auswahl Seiten

Allgemein

Geht man von der Startseite (Abbildung 5.6) aus, so werden drei Buttons (Szenario-, Problemfeld- und Ebenen-Auswahl) angeboten. Die Funktionen und Visualisierungen der einzelnen Seiten, welche durch Anklicken einer der genannten Buttons clientseitig verwendet wird, ähneln einander sehr, weswegen diese hier unter einem Punkt zusammenge-

fasst beschrieben werden. Der Unterschied besteht im Wesentlichen darin, dass entweder das Szenario, das Problemfeld oder eine bestimmte Ebenenkategorie in den Mittelpunkt der Visualisierung rückt. Die weitere Beschreibung unter diesem Punkt bezieht sich auf die Problemfeld-Auswahl und steht stellvertretend auch für die anderen beiden Auswahl Seiten.

Durch den Klick auf den ”Problemfeld-Auswahl” Button wird ein entsprechender AJAX-Request mit voreingestellten Parametern an den Server geschickt (siehe 5.3.3). Grundsätzlich werden auf der neu aufgebauten Seite (siehe Abbildung 5.10) die Verbindungen zwischen einem auszuwählenden Problemfeld und den zugehörigen Anforderungen gezeigt. Die Seite ist in 5 verschiedene Sektoren unterteilt. Im obersten Sektor kann ein bestimmtes Problemfeld und ein bestimmtes Szenario ausgewählt werden. Zusätzlich ist es möglich durch deaktivieren der Checkbox ”Keine Ebenenauswahl”, eine Ebenenauswahl vorzunehmen. Bei jeder Auswahl, die durch den Anwender getroffen werden, wird stets ein neuer AJAX-Request mit der zugehörigen Funktion an den Server abgesetzt. Zusätzlich zum Namen der Funktion die am Server aufgerufen werden soll, sind die 3 Parameter (*pfSelection*, *szSelection* und *ebSelection*) für die Server Logik entscheidend. Diese drei Parameter enthalten die Informationen, welche der Anwender mit Hilfe der Graphikelemente (Drop-Down Listen und Check Boxen) auf der Seite ausgewählt hat.

Die serverseitige Antwort im JSON-Format wird im mittleren Fenster in eine graphische Visualisierung umgesetzt. Dabei wird das ausgewählte Problemfeld immer in die Mitte gesetzt und die direkt verbundenen Anforderungen, welche sich in der gleichen Wirkkette befinden, radial nach außen angeordnet. Zusätzlich zu den Anforderungen, welche sich in der gleichen Wirkkette zum selektierten Problemfeld befinden, werden in einem zweiten äußeren Radius gegebenenfalls Anforderungen visualisiert, welche über die *Gleich* Spalte aus der Rohdatendatei eine Verwandtschaft zu den Anforderungen aus dem ersten Radius besitzen.

Radiale Anordnung der Knoten

Durch die radiale Anordnung entsteht eine Hierarchie der Visualisierung. Der Knoten im Mittelpunkt der Visualisierung wird im JSON Format des *nodes*-Objektes im *hierarchie*-Element mit einer ”0” übertragen. Damit kann im Javascript Code eindeutig der Mittelpunktsknoten ermittelt werden. Alle Anforderungen welche direkt zum Problemfeld zugeordnet sind, haben im *hierarchie*-Element des *nodes*-Objekts im übertragenen JSON Format eine ”1” eingetragen. Der Javascript Code ermittelt alle Knoten Objekte mit Hierarchie ”1” und dividiert die Anzahl der Knoten durch 360° um so einen gleichmässigen

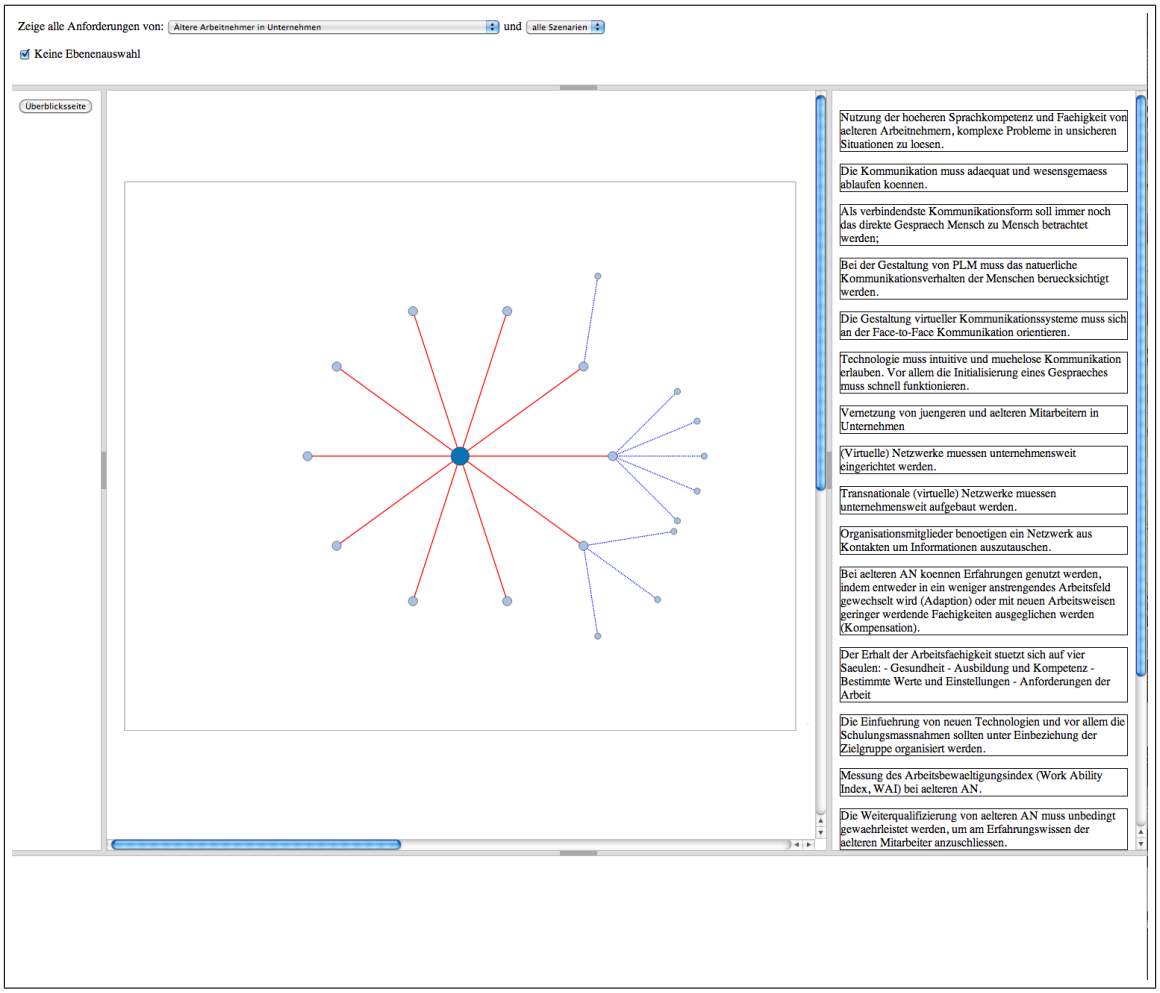


Abbildung 5.10: Problemfeldauswahl Startseite

Abstand der Knoten auf dem voreingestellten Radius zu erhalten.

Bei Anforderungen, welche keine direkte Verbindung mit dem selektiertem Problemfeld aufweisen, dennoch mit den Anforderungen der ersten Hierarchie verwandt sind, werden serverseitig mit einer ”2” im *hierarchie*-Element an den Client übertragen. Ausserdem hat das *nodes*Objekt ein *gleichNodes*-Element, welches als Array alle verwandten Knoten von der Serversoftware eingetragen bekommt. Damit wird im Javascript Code die zweite graphische Hierarchie aufgebaut. Jedes Knotenelement, welches die Hierarchie ”1” besitzt wird kontrolliert, ob es auch verwandte Knoten im JSON Objekt eingetragen hat. Ist dies der Fall, wird abhängig von der jeweiligen Knotenposition mit der ersten Hierarchie ein neunzig Grad goßes Feld nach außen aufgespannt und in diesem werden auf einem vordefinierten Radius alle Kindknoten gezeichnet. Diese Anordnung kann bei vielen Knoten am 1. und 2. Radius zu Überschneidungen der Kanten führen, jedoch bleibt die Struktur und somit die Lesbarkeit erhalten.

MouseOver - Event

Die Mouseover Funktionalität über einen graphischen Knoten zeigt im Unterschied zu 5.5.3 keinen Tooltip an. Auf der Problemfeld-Auswahl Seite wird unterschieden über welchen Knoten die Maus bewegt wird. Steht die Maus über dem Problemfeld-Knoten (welcher immer im Zentrum gezeichnet wird), dann wird der Zelleninhalt der Rohdatendatei im linken Teilfenster angezeigt. Bei einem Mouseover über einen der Anfordungs-Knoten wird zusätzlich zur Information im linken Fenster (gleich wie bei einem Mouseover über den Problemfeld-Knoten), der Teilgraph im mittleren Fenster hervorgehoben und die Zugehörigkeit in der Wirkkette oder den Wirkketten im unteren Teilfenster gezeichnet (siehe Abbildung 5.11). Um die graphische Information im unteren Fenster darstellen zu können wird beim MouseOver - Event auch ein AJAX-Request an den Server gesendet (siehe 5.3.3). Dieser Request beinhaltet die Beschreibung der Anforderung, welche das MouseOver - Event ausgelöst hat. Anhand der Anforderungsbeschreibung errechnet der Server die Knoten welche der Wirkkette(n) angehören und liefert wieder ein JSON Objekt nach 5.4 zurück. Mithilfe dieses JSON Objektes und dem Force-Directed-Layout wird die animierte Graphik im unteren Fenster dargestellt.

Die Abbildung 5.11 zeigt ebenfalls ein Manko im unteren Fenster in der Darstellung von Text innerhalb eines SVG Tags. Die SVG Spezifikation [Dahlström et al., 2011] berücksichtigt keine automatischen Textumbrüche. Daher ist auch im unteren Teilfenster der Text zu den jeweiligen Knoten in einer einzigen Zeile dargestellt und es kann dazu führen, dass verschiedene Textzeilen zu lange für die bereitgestellte Displayfläche sind, oder Textzeilen sich gegenseitig überlagern. Ein möglicher Lösungsansatz wäre, dass der vom JSON Protokol empfangene Text zuerst in seiner Länge analysiert und danach dynamisch mit Hilfe des tspan-Elementes zeilenweise zerlegt wird. In der vorliegenden Arbeit wird aber nicht näher darauf eingegangen.

Das rechte Teilfenster zeigt alle Anforderungen, welche auch im mittleren Teilfenster graphisch dargestellt werden, in textueller Form an. Fährt der Anwender mit der Maus über eine Textpassage wird der dazugehörige Knoten in der graphischen Darstellung andersfarbig angezeigt. Damit kann eine Verbindung zwischen dem Text und in der graphischen Darstellung hergestellt werden.

Zoom - Event

Die graphische Visualisierung (mittleres Fenster) kann mithilfe des Mausrades gezoomt und mithilfe der gedrückten linken Maustatse innerhalb des mittleren Fensters verschoben

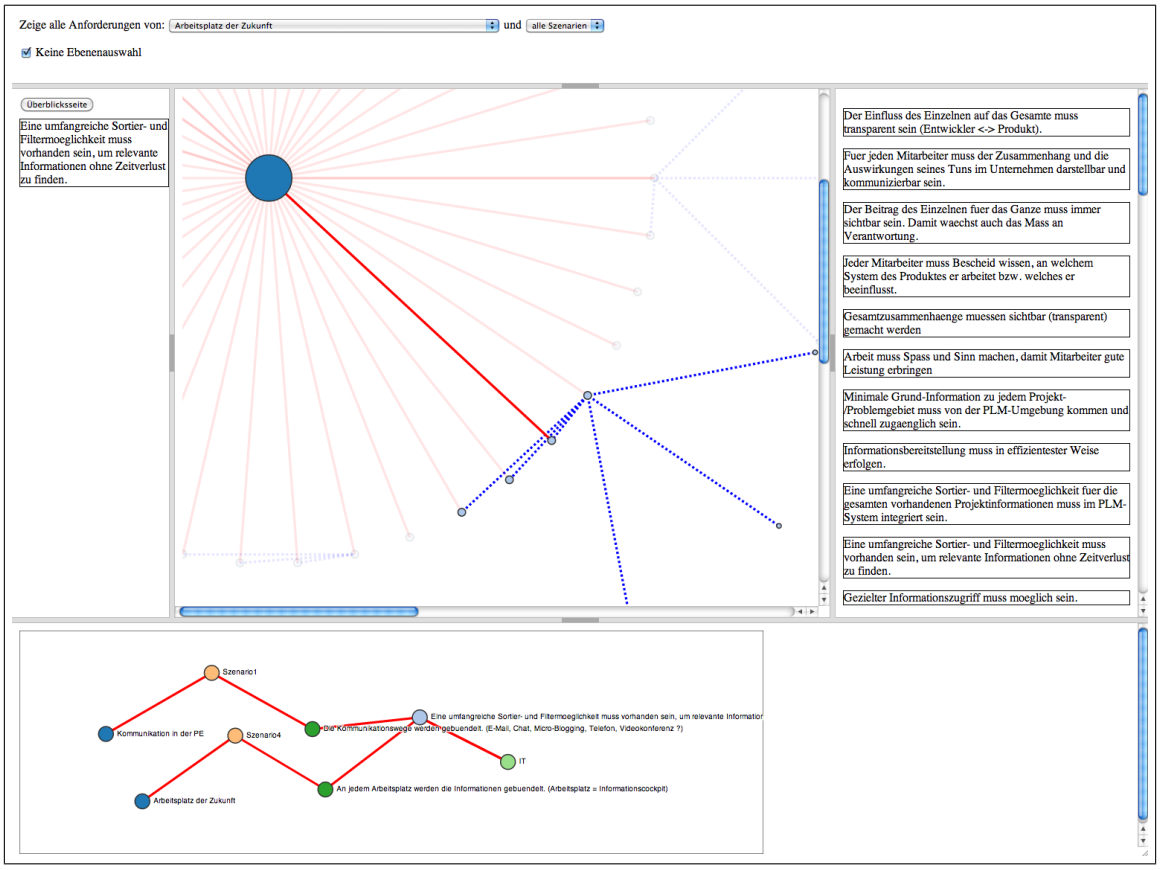


Abbildung 5.11: Problemfeldauswahl Mouseover

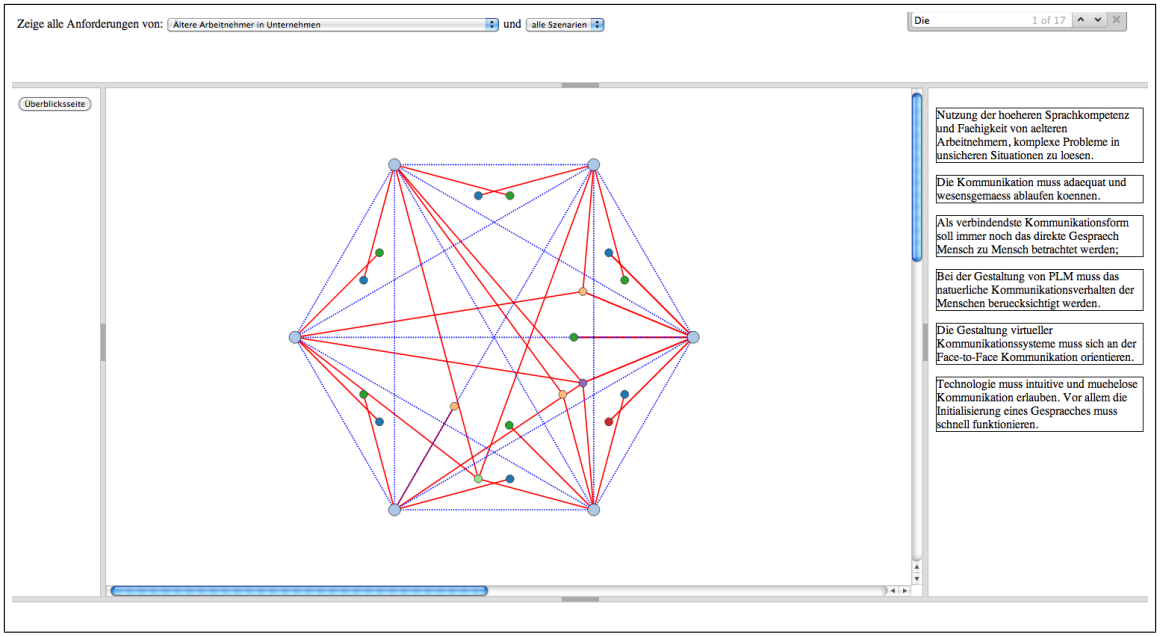


Abbildung 5.12: Problemfeldauswahl Anforderung ausgewählt

werden.

Ausblenden von Details

Des Weiteren wurde versucht Teilaspekte des Visualisierungsmottos (siehe Kapitel 2.4) umzusetzen. Zum Beispiel können mit der 'c' Taste, jene Kindknoten ein- bzw. ausgeblendet werden, welche über die "Gleich" Spalte verbunden sind. In der Visualisierung sind dies die Knoten am äußeren Radius. Wenn die Knoten ausgeblendet sind, werden die Elternknoten in einer anderen Farbe angezeigt, um anzuzeigen, dass sich weitere Information dahinter verbirgt. Diese Funktionalität wurde clientseitig programmiert und sendet keinen AJAX-Request an den Server ab. Das Ergebnis ist in Abbildung 5.13 dargestellt.

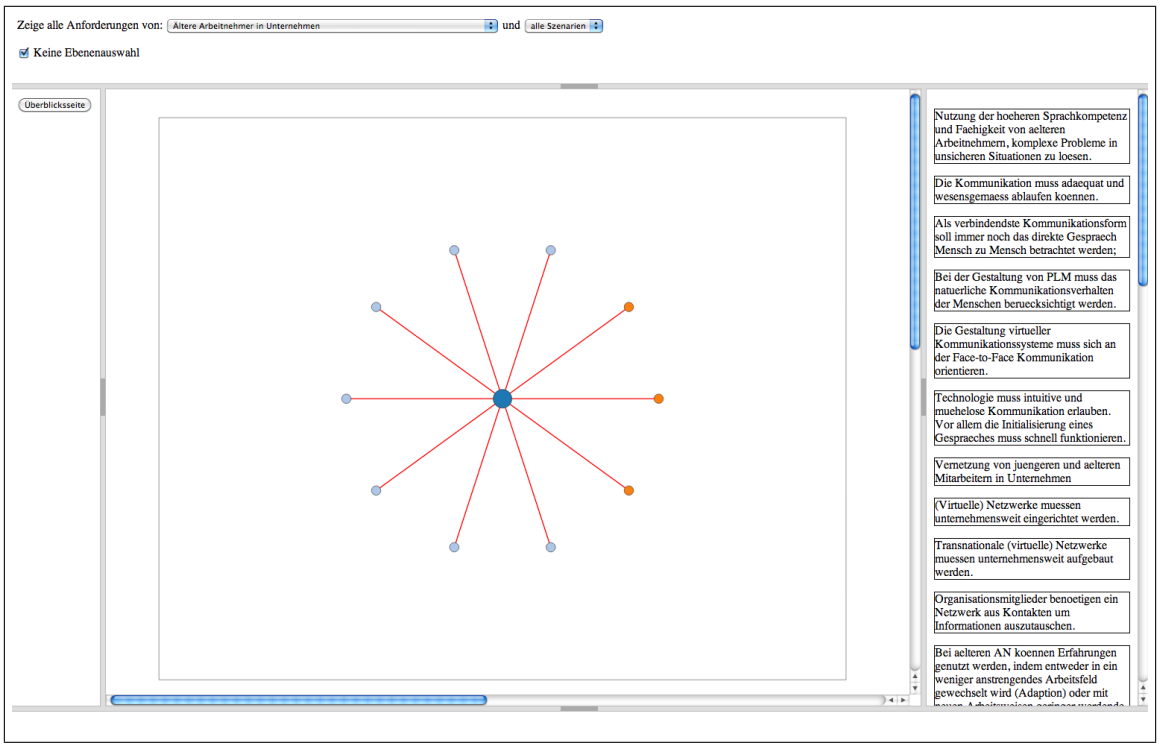


Abbildung 5.13: Problemfeldauswahl - Details ausgeblendet

Navigieren

Eine andere Visualisierungsart wird angezeigt, wenn der Anwender auf einen der Anforderungsknoten klickt. In dieser Darstellung (siehe Abbildung 5.12) wird wieder die gesamte Wirkkette angezeigt und falls vorhanden, alle verwandten Anforderung inklusive deren Wirkketten. In dieser Darstellung werden mittels der Mouseover Funktion alle Wirkketten, in welchen der selektierte Knoten vorkommt, hervorgehoben. Dies gilt

natürlich nur für die Wirkketten, welche in der Darstellung aufgrund der eingestellten Selektion angezeigt werden.

5.5.5 Ebenenzuteilung

Über die Überblicksseite (siehe Abbildung 5.6) kann auch die Funktionalität der Ebenenzuteilung ausgewählt werden. Diese dient dazu sich einen Überblick über die Anforderungen und deren Zuweisung zu den verschiedenen Ebenen zu verschaffen. Anforderungen können entweder der Ebene *Mensch*, *Organisation*, *IT*, *Prozess*, *Methode* sowie allen möglichen Kombinationen aus diesen fünf Ebenenkategorien zugewiesen werden. Im konkreten heißt dies, dass alle gegebenen Anforderungen in einer der 32 möglichen Gruppen eingeteilt sind. Um einen schnellen Überblick über diese Gruppeneinteilung zu erhalten wurde diese Ebenenzuteilung visualisiert. Abbildung 5.14 zeigt die Visualisierung. In der oberen Displayhälfte in Abbildung 5.14 entspricht jeder Kreis einer bestimmten Anforderung, die Farbe des Kreises symbolisiert die Ebenenkategorie. Anforderungen gleicher Ebenenkatogorie sind geclustert und befinden sich am Bildschirm nahe nebeneinander. Die untere Displayhälfte zeigt eine Art Legende in Tabellenform. Die Tabelle ist gleich angeordnet wie die Anforderungscluster in der oberen Displayhälfte. Zusätzlich zur gleichen Anordnung ist der Text, welcher die einzelnen Gruppen beschreibt, auch in der gleichen Farbe geschrieben wie die Kreise welche die Anforderungen symbolisieren. Die Abbildung 5.14 zeigt auch die Mengenverteilung der einzelnen Ebenenkategorien.

5.5.6 SPARQL Abfrage

Die Funktionalität, welche sich hinter diesem Button verbirgt wird eingehend im nächsten Kapitel behandelt, da der verwendete Datensatz sowie die Datensuche, sich von den bisher jetzt beschriebenen Verfahren unterscheidet.

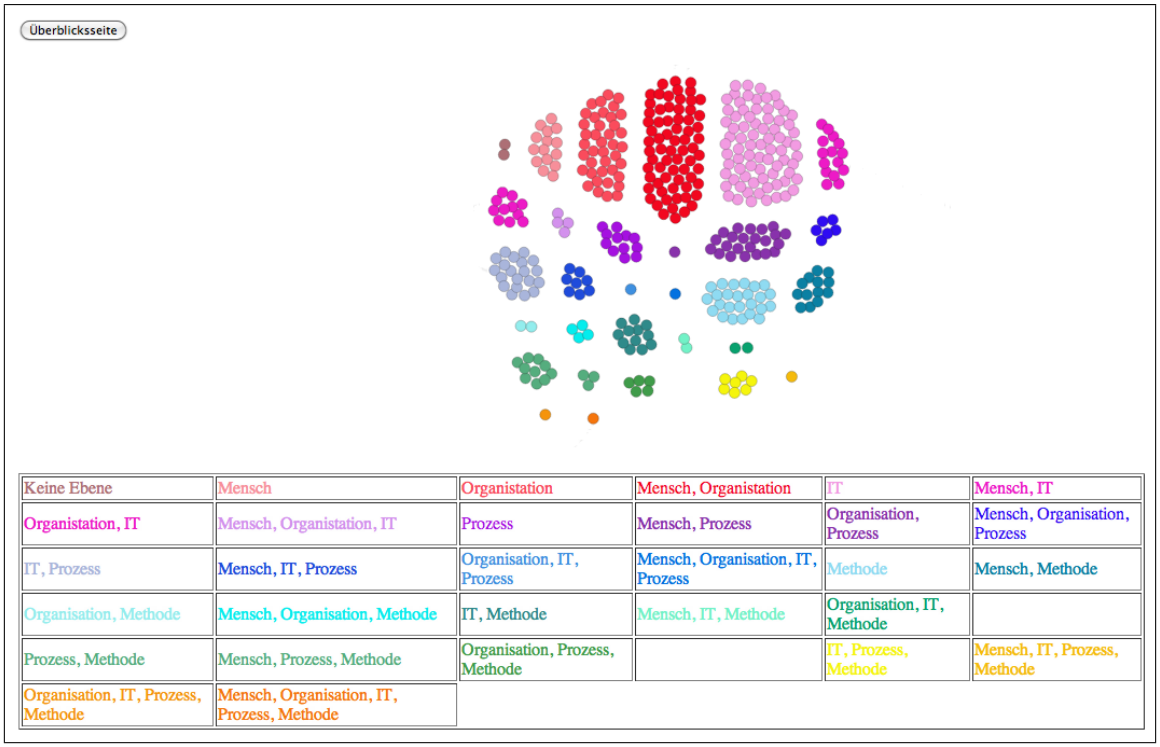


Abbildung 5.14: Ebenenzuteilung der Anforderungen

Kapitel 6

Praktischer Ansatz mithilfe von semantischen Daten

6.1 Kapitelüberblick

Wie im vorigen Kapitel beschrieben, wird der Datensatz von der Rohdatendatei eingelesen und intern in einer zuvor definierten logischen Struktur im Speicher gehalten. Um aus dieser logischen Struktur wieder bestimmte Daten abfragen zu können, muss mit relativ hohem Aufwand mittels verschachtelten Schleifen nach den Ergebnissen gesucht werden. Diese Methode benötigt einerseits einen hohen Programmieraufwand, andererseits ist dieser auch sehr fehleranfällig.

Um diese Fehleranfälligkeit, aber auch den Programmieraufwand zu minimieren und dadurch auch die Erweiterbarkeit der vorliegenden Software zu erhöhen, wurde ein Datenmodell mit semantischen Ansätzen entworfen. Dieses Datenmodell wurde zuerst definiert, um es danach in Software so nachzubilden, sodass die automatisch eingelesene Rohdatendatei (siehe 3.1) in ein OWL Format (Web Ontology Language) [Smith et al., 2004] umgewandelt wird. Diese Triple werden in ein OWL Datei ausgelagert und gespeichert. Werden Fragen an dieses Triple-Datenmodell gestellt, in der vorliegenden Arbeit über die Clientsoftware durch Interaktion des Anwenders, so geschieht dies serverseitig mithilfe von sogenannten SPARQL Abfragen [Prud'hommeaux und Seaborne, 2008].

Dieses Kapitel gibt keine genaue Beschreibung über das OWL Format noch über SPARQL Abfragen beziehungsweise wird auch nicht näher auf den Begriff und Hintergrund des Semantischen Web eingegangen. Jedoch wurden alle relevanten Informationen

aus Hitzler et al. [2008] und von den W3C Seiten für OWL [Smith et al., 2004] und SPARQL [Prud'hommeaux und Seaborne, 2008] bezogen.

6.2 Triple Model

Ausgangspunkt für das Triple Model sind die Rohdatendatei und deren Spalten. Da die Anforderungen während der ganzen Projektdauer und bei der Erstellung der Rohdaten eine zentrale Rolle spielten, wurden diese auch im Triple Model, welches auch einen Graphen darstellt, in den Mittelpunkt gerückt. Somit hat die Anforderung mit jeder anderen Spalte aus der Rohdatendatei eine Verbindung. Manche dieser Verbindungen sind bidirektional ausgeführt, um für nachfolgende Arbeiten einen umfangreichen und leicht abzufragenden Datensatz zur Verfügung zu stellen.

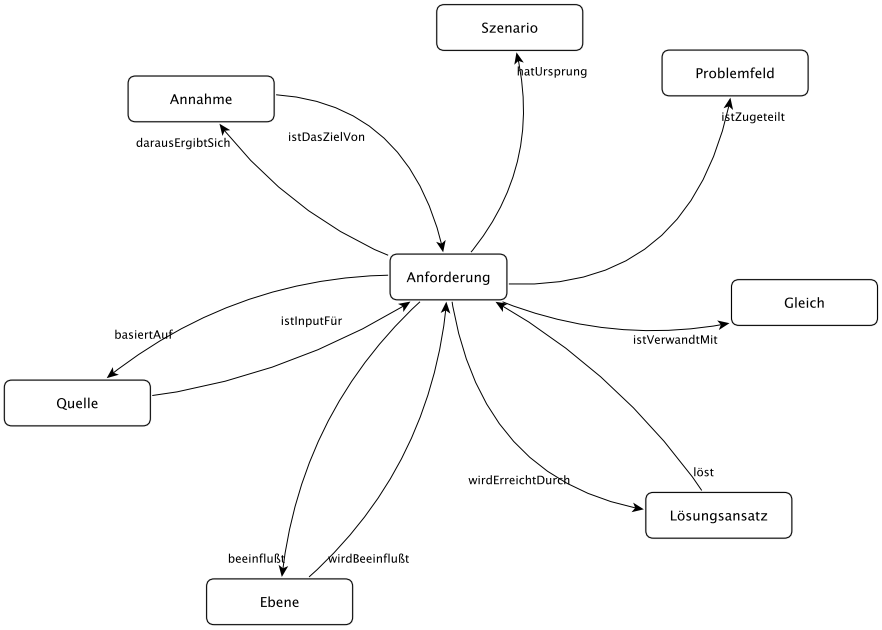


Abbildung 6.1: Triple Datenmodell

6.3 Umwandeln der CSV Daten in RDF Format

6.3.1 Allgemein

Das Ziel der Software ist es, eine CSV Datei mit ihren Spalten einzulesen und daraus ein Datenmodell zu gestalten. In der vorliegenden Arbeit wurde die Software dahingehend

geschrieben, dass das Model (siehe Abbildung 6.1) in OWL abgebildet wird. Jedoch wurde versucht die Software anhand von Design Pattern relativ allgemein zu halten, um mit geringem Aufwand diese verändern zu können um auch andere CSV Rohdatendateien einzulesen und deren Spalten anders, als in Abbildung 6.1, zu verlinken.

6.4 Beschreibung der Softwarearchitektur

6.4.1 Allgemein

Abbildung 6.2 zeigt die Grobstruktur, welche gefunden und implementiert wurde. Der Startpunkt (*main()* - Methode) befindet sich in der *CSV2OWL* Klasse. Unter Zuhilfenahme eines ”Creational Pattern” dem sogenannten ”Abstract Factory Pattern” (siehe Gamma et al. [1994]) werden in der *CSV2OWL* Klasse Objekte vom Typ *AnforderungsFactory* und *AnnahmeFactory* erzeugt. Es werden noch mehr Factories, welche immer einer Spalte in der CSV Datei entsprechen, erzeugt, diese wurden aber in Abbildung 6.2 aus Platzgründen nicht eingezeichnet. Daher stehen *AnforderungsFactory* und *AnnahmeFactory* repräsentativ für alle anderen Factory Objekte, welche entweder in der Software implementiert wurden oder aufgrund des Design Patterns leicht hinzugefügt werden könnten.

Die Factory Objekte (z.B. *AnforderungsFactory* oder *AnnahmeFactory*) erzeugen ihrerseits die zuvor definierten Verbindungen (siehe Abbildung 6.1) wie zum Beispiel *HatUrsprungConnection* oder wie in Abbildung 6.2 eingezeichnet die Verbindungen *IstDasZielVonConnection*, *BasiertAufConnection* und *WirdErreichtDurchConnection*. Bezogen auf Gamma et al. [1994] entspricht die *AnforderungsFactory* einer ”ConcreteFactory” und die Klassen *IstDasZielVonConnection*, *BasiertAufConnection* usw. den verschiedenen ”Products”.

Die eingezeichnete UML-Komposition in Abbildung 6.2 zwischen *ConcreteProperties* und *ConnectionFactory* ist wichtig, da mithilfe der public Methoden von den konkreten Objekten der *ConnectionFactory* Klasse die OWL-Verbindungseigenschaften zusammengebaut werden. Diese OWL-Verbindungseigenschaften (wie z.B. *transitive* oder *inverse functional*) sind Eigenschaften der *ConnectionFactory* Klasse.

Jede konkrete Verbindung (Connection) wie zum Beispiel die *IstDasZielVonConnection* hält ihrerseits die konkreten *OWLIndividual* Objekte. Mit dieser Struktur ist sichergestellt, dass die Software sehr einfach um neue Verbindungen mit deren OWL - Verbindungseigenschaften zwischen den OWL Individuen hergestellt werden kann.

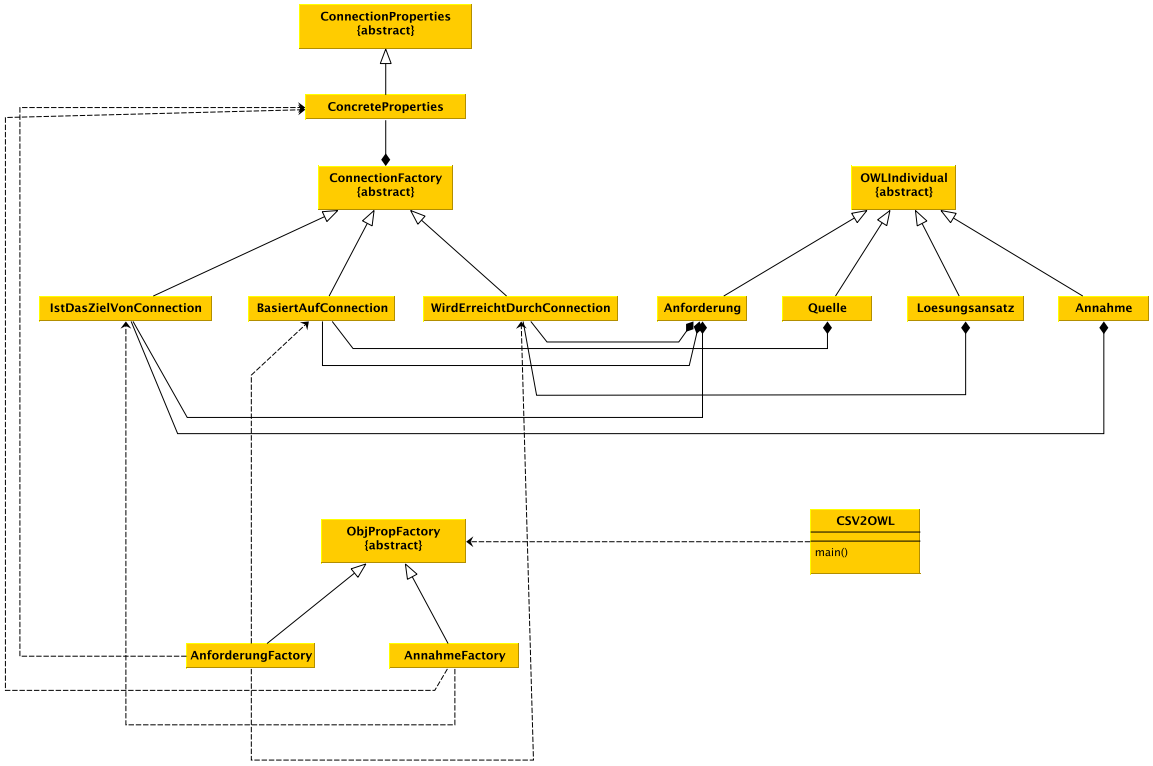


Abbildung 6.2: UML Diagramm Überblick

6.4.2 Erklärung des Softwareablaufs anhand der Erzeugung der *IstDasZieVon* Verbindung

Nachfolgende Ablaufbeschreibung orientiert sich an der Erstellung einer *IstDasZielVonConnection*, welche eine Verbindung zwischen Anforderung und Annahme (siehe Abbildung 6.1) herstellt. Die Abbildung 6.3 zeigt die Struktur für alle dafür erforderlichen Klassen, jedoch detailierter als es in der Abbildung 6.2 dargestellt ist.

Über die *main*-Methode in der Klasse *CSV2OWL* wird zuerst die CSV Datei eingelesen. Dafür wird, wie schon unter 5.3.2 beschrieben, eine externe Java-Bibliothek (`http://opencsv.sourceforge.net` Zuletzt besucht am: 2012.05.03) verwendet, welche zeilenweise aus einer CSV Datei ausliest und diese Daten in einem String Array zur Verfügung stellt. Dieses String Array wird beim Aufruf der Methode *connectTo* an die konkrete Factory, im Fall der *IstDasZielVonConnection*, an die *AnnahmeFactory* weitergegeben. Die *connectTo* Methode, welche die Daten aus dem String Array ausliest, überprüft mittels HashMap, ob die entsprechenden Objekte zuvor schon eingelesen wurden oder nicht. Als Schlüssel für die HashMap, wird gleich wie unter Punkt 5.3.2 beschrieben, der textuelle Inhalt der jeweiligen Zelle aus dem Rohdatenformat verwendet und der

Wert der HashMap entspricht einem *OWLIndividual* Objekt. Die HashMap wird in einem Singleton gehalten und ist somit als globale Struktur im gesamten Programm les- und veränderbar. Dies ist von Vorteil, weil nicht nur die konkreten Factories sondern auch andere Objekte, wie zum Beispiel alle Connection Objekte darauf zugreifen und verändern können.

Unabhängig davon, ob die Daten schon in der HashMap existieren oder nicht, wird ein neues *IstDasZielVonConnection* Objekt, innerhalb der *AnnahmeFactory*, angelegt. Dieses Connection Objekt zusammen mit den Daten der Rohdatendatei wird dem Konstruktor der *ConcreteProperties* Klasse übergeben. In diesem Konstruktor werden alle relevanten Funktionen der konkreten *ConnectionFactory*, in dem hier beschriebenen Ablauf ist damit das *IstDasZielVonConnection* Objekt gemeint, aufgerufen und deren Rückgabewerte in dem *ConcreteProperties* Objekt gespeichert. Die Rückgabewerte sind Listen von Objekten des Typs *OWLIndividual*. Je nach Funktionsaufruf werden entweder die Objekte von denen die Verbindung ausgeht, oder jene Objekte in denen die Verbindung endet, zurückgeliefert. Damit werden alle Eigenschaften der *ConcreteProperties* befüllt und können zu einem späteren Zeitpunkt über die angebotenen public Methoden abgefragt werden.

Sind die oben beschriebenen Schritte ausgeführt, wird in der *connectTo* Methode des *AnnahmeFactory* Objektes das erzeugte und befüllte *ConcreteProperties* Objekt jenem *OWLIndividual*, welches zuvor entweder in der HashMap gefunden wurde oder neu angelegte Objekt, als Eigenschaft zugewiesen. Damit entsteht in den einzelnen *OWLIndividual* Objekten eine Liste von *ConcreteProperties* Objekten mit genau jenen Eigenschaften die durch die konkrete *ObjPropFactory* in der *connectTo* Methode vorgegeben werden.

Als Rückgabewert an die *CSV2OWL* Klasse wird ein konkretes *OWLIndividual* Objekt retourniert. In der CSV2OWL Klasse werden alle konkreten *OWLIndividuals* eines bestimmten Typs in einer Liste zusammengefasst. Ist der Lesevorgang und das Zuweisen der Objekte durch das vordefinierte Datenmodell beendet muss das interne Datenmodell in eine owl Datei übertragen werden. Dafür wird ebenfalls eine externe Java-Bibliothek von Apache Jena [The Apache Software Foundation, 2012] verwendet. Die Bibliothek steht unter der Apache Licence 2.0 und bietet Funktionalitäten in Bezug auf Lesen, Verarbeiten und Schreiben von OWL Dateien. Des Weiteren ist es mit der Bibliothek möglich SPARQL Abfragen an OWL - Fileformate zu richten. Dies wird weiter unten in dieser Arbeit beschrieben, da diese Funktionalität zur Visualisierung der Daten genutzt wird.

In der *CSV2OWL* Klasse wird mittels Apache Jena die Klassenstruktur und danach die einzelnen Individuen und deren Verbindungen, welche vom internen Datenmodel gelesen werden, in das OWL Dateiformat übertragen.

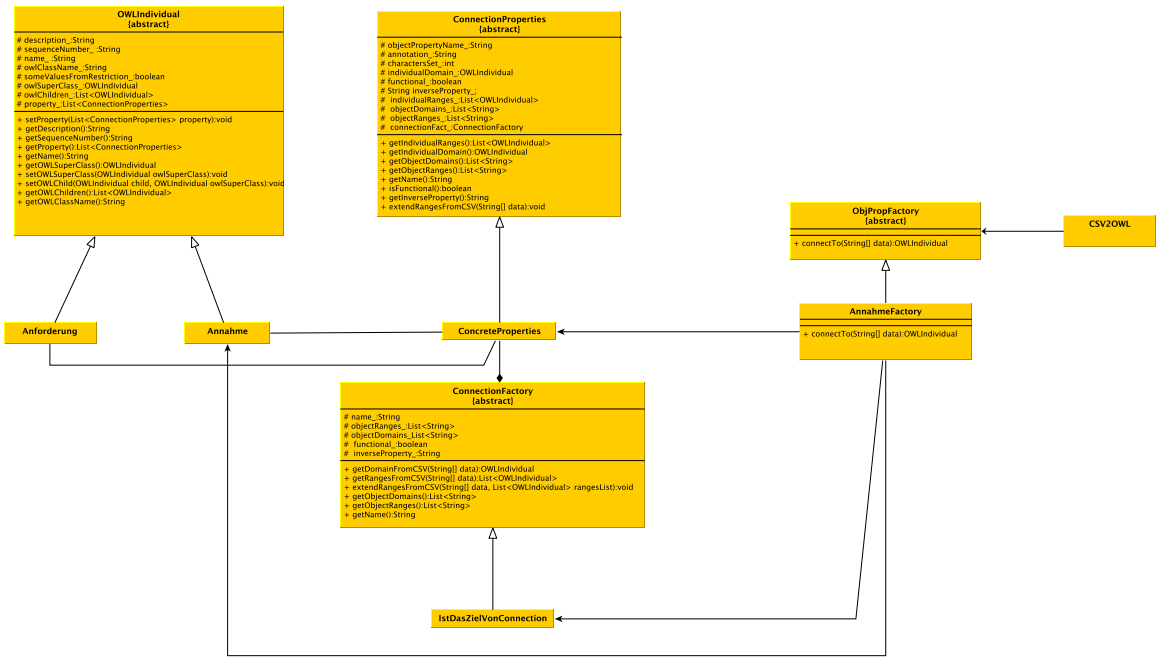


Abbildung 6.3: UML Diagramm im Detail

6.5 Auswertung und Überprüfung mittels Protege

Protege ist ein freier und unter der Mozilla Public Licence 1.1 lizenzierter, open source Ontologie Editor [Stanford Center for Biomedical Informatics Research, 2012]. Mithilfe dieses Editors können OWL Dateien erstellt oder eingelesen werden. Über verschiedene Plugins kann die OWL Datei auch per SPARQL Abfragen getestet, oder graphisch aufbereitet werden. Protege half bei der Evaluierung der, vom eigenen Javaprogramm erstellten, OWL Datei auf deren Sinnhaftig- und Vollständigkeit.

Datenmodell in Protege

Abbildung 6.4 das automatisch generierte Datenmodel aus der erzeugten owl Datei mittels Protege. Vergleicht man das automatisch generierte Datenmodel mit 6.1 so ist zu erkennen, dass diese beiden ident sind.

Individuen in Protege

Abbildung 6.5 zeigt einen Ausschnitt der generierten OWL Individuals. Der Name der Individuals wurde mit dem Spaltennamen und einer fortlaufenden ID der jeweiligen Gruppe vergeben. Das Bild zeigt links unten mit welchen Individuen die Anforderung320 verbun-

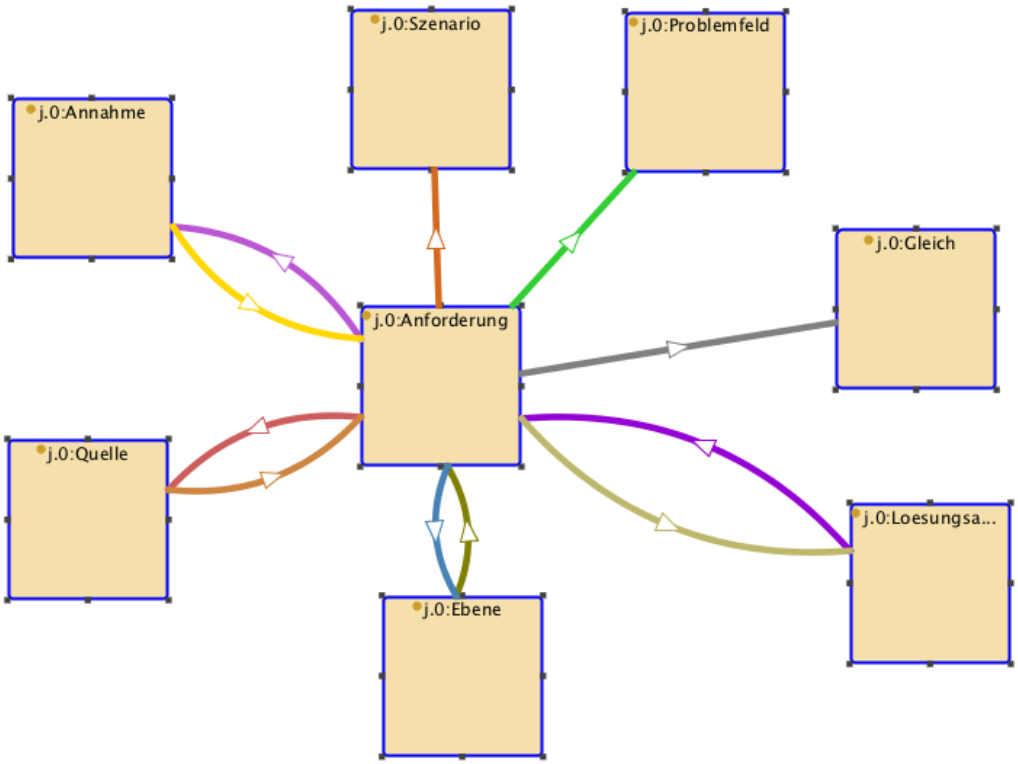


Abbildung 6.4: OWL Datemodell mittels Protege - Jambalaya

den ist. Ausserdem wurde unter Sektion *data property* eine eindeutige ID und der textuelle Inhalt der Spalte als *Beschreibung* automatisch durch das Javaprogramm zugewiesen.

SPARQL Abfragen in Protege

SPARQL Abfragen können in Protege relativ einfach in einem eigenen Editor Fenster eingegeben und danach ausgeführt werden. Die Ergebnisse werden in Tabellenform (siehe Abbildung 6.6) ausgegeben.

Die Zeilen 4 bis 8 in der Abrfage (siehe Listing 6.1) legt fest welche Individuen bzw. welche Eigenschaften gesucht und angezeigt werden sollen. Die Zeile 9 filtert jene Wirkkette, welche in der Problembeschreibung ”aeltere Arbeitnehmer in Unternehmen” beinhalten. In Zeile 10 wird angegeben, dass falls eine Verbindung zwischen den gefundenen Anforderungen und Gleich besteht, dieser Inhalt ebenfalls angezeigt werden soll. Die gezeigte Abfrage in Listening 6.1 führt bei Anwendung auf die erzeugte OWL Datei zum Ergebnis, welches in Abbildung 6.6 dargestellt ist.

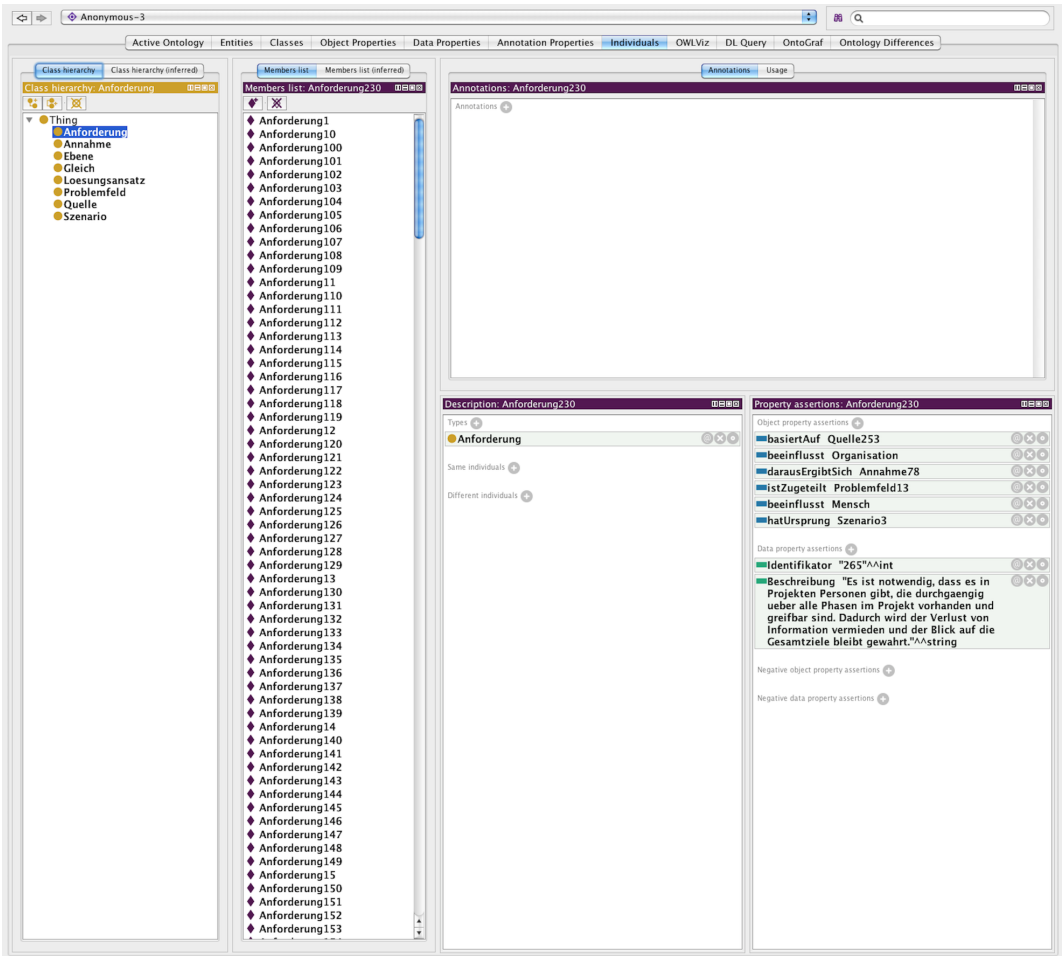


Abbildung 6.5: OWL Individuals mittels Protege

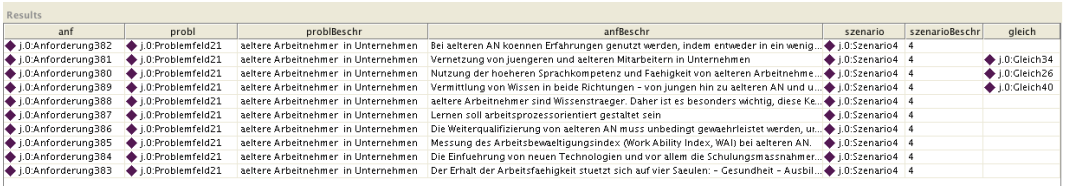


Abbildung 6.6: Ergebnis der SPARQL Abfrage (siehe Listing 6.1) in Protege

6.6 Datenvisualisierung mittels SPARQL Abfragen

In Abbildung 5.6 ist ein ”SPARQL-Abfrage” Button zu sehen. Wird der Button gedrückt, löst dieser clientseitig einen Funktionsaufruf (problemfeldSelectedWithSPARQL()) am Server aus. Die aufgerufene Funktion ist den Funktionen, welche unter Punkt 5.3.3 beschrieben werden, sehr ähnlich, nur mit dem Unterschied, dass die Daten aus der automatisch generierten owl-Datei mittels SPARQL-Abfragen erhalten werden. Sind innerhalb

```
1  PREFIX vif: <http://www.example.com/ontologies/anforderungsliste/anforderungsliste.owl#>
2  SELECT *WHERE
3  {
4    ?anf  vif:istZugeteilt  ?probl.
5    ?probl vif:Beschreibung ?problBeschr.
6    ?anf vif:Beschreibung ?anfBeschr.
7    ?anf vif:hatUrsprung ?szenario.
8    ?szenario vif:Beschreibung ?szenarioBeschr.
9    FILTER (?problBeschr= "aeltere Arbeitnehmer  in Unternehmen").
10   OPTIONAL{?anf vif:istVerwandtMit ?gleich}.
11 }
```

Listing 6.1: SPARQL - Abfage in Protege

dieser Funktion alle Daten aus der OWL Datei verarbeitet, werden diese mit dem beschriebenen JSON-Format (siehe 5.4) an den Client übertragen. Somit sind clientseitig keine Veränderungen vorzunehmen, obwohl sich die Datenaquirierung auf der Serverseite maßgeblich verändert haben.

Die Abbildung 6.7 zeigt die gleiche Information wie in Abbildung 5.10, jedoch mit dem Unterschied das serverseitig die Information in Abbildung 6.7 durch SPARQL Abfragen erhalten wurden. Der Grund für die Verschiebung der Anordnung der einzelnen Knoten liegt daran, dass bei Abbildung 6.7 die Daten so verarbeitet werden, wie sie durch die SPARQL Abfragen erhalten werden. Dies kann zu extremen Visualisierungseigenschaften führen, wie es in Abbildung 6.8 dargestellt wird. Der Grund für den Unterschied der Visualisierung ist aber nicht in der Art der Abfrage begründet, sondern wie die Daten nach der Abfage angeordnet werden. D.h. die Reihenfolge im JSON Objekt ist dafür verantwortlich. Da diese JSON Objekt Reihenfolge vom Javascript direkt übernommen wird, wirkt sich dies in der Visualisierung aus.

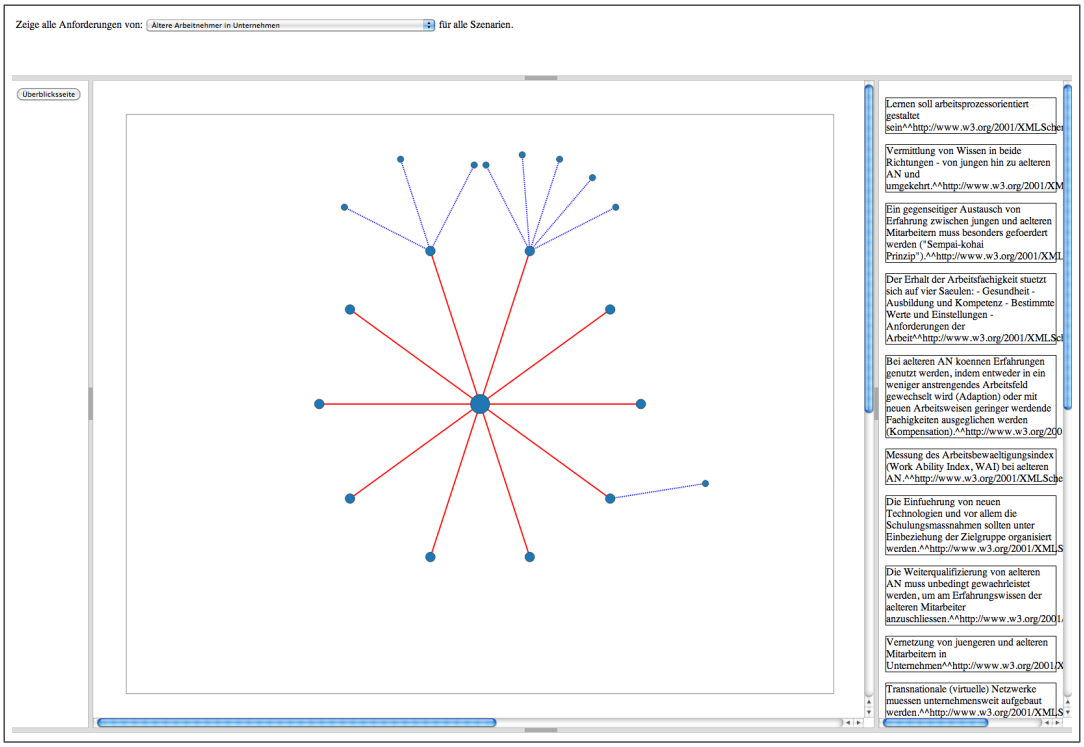
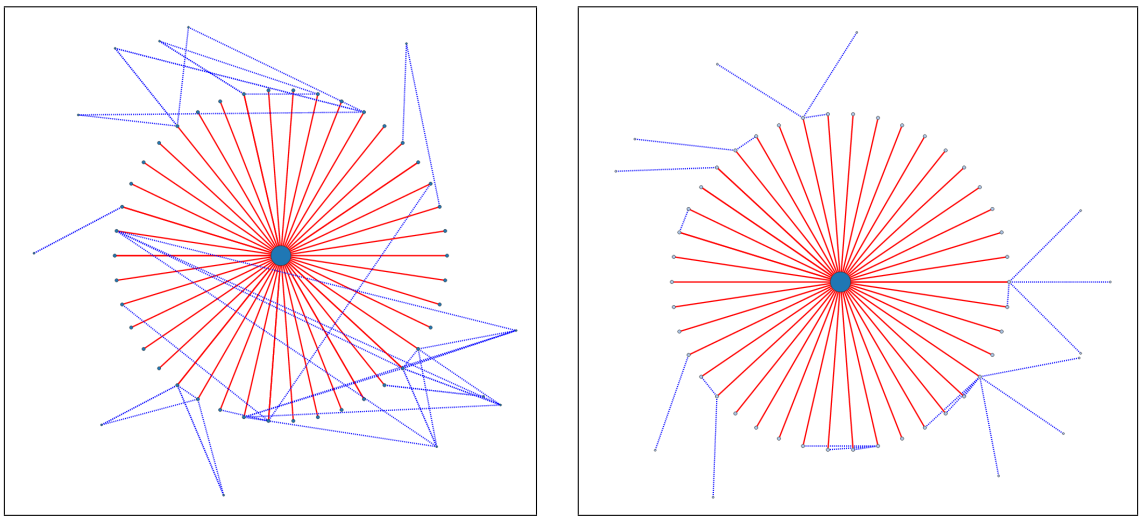


Abbildung 6.7: Visualisierung der Daten mittels SPARQL-Abfrage



(a) Visualisierung mittels SPARQL-Abfrage **(b)** Darstellung anhand interner Datenstruktur

Abbildung 6.8: Vergleich zweier Visualisierungen (ungeordnet/geordnet)

Kapitel 7

Evaluierung und Ausblick

7.1 Evaluierung

Die Software wurde einzelnen Projektteilnehmern von FuturePLM vorgestellt und das Ergebnis diskutiert. Allen Beteiligten gefiel die Art und Weise, wie die Daten des Anforderungskataloges visuell umgesetzt wurden. Generell ist zu sagen, dass diese Art der Informationsvisualisierung auch bei den Projektteilnehmern großen Anklang gefunden hat. Die Software in ihrem jetzigen Stand dient dazu zu zeigen, wie eine gewisse Art von Daten aufbereitet und dargestellt werden kann.

Es konnte festgestellt werden, dass diese Art der Software weitere Ideen und Vorstellungen in den Köpfen der Projektpartner hervorgerufen hat. Damit dient das Tool nicht nur zur Präsentation und Erklärung von Daten, sondern auch zur Demonstration von Datenvisualisierung mit modernen Technologien.

7.2 Ausblick

Als Ausblick können mehrere Ideen und Gedanken festgehalten werden. Ganz allgemein ist festzustellen, dass das Kompetenzzentrum - Das Virtuelle Fahrzeug - sich für die Zukunft vorgenommen hat, mehr Wissen in Richtung Informationsvisualisierung mit neuen Technologien aufzubauen. Dementsprechend ist zu erwarten, dass aufbauend auf diese Arbeit weitere wissenschaftliche Arbeiten folgen werden. Deshalb soll hier nochmals eine Auflistung von Themen und Kategorien gegeben werden, welche für nachfolgende Projekte und Arbeiten von Relevanz sein könnten.

Die Einführung und Verwendung von Datenbanken vereinfacht die Handhabung und Abfrage der Daten, vor allem wenn es sich um einen großen Datensatz handelt. In der vorliegenden Arbeit wird mit Java gearbeitet somit ist der Schritt hin zur Einbindung einer Datenbank sehr einfach durchzuführen. Ein weiterer großer Vorteil in der Verwendung von Datenbanken liegt darin, dass Anwendern die Möglichkeit gegeben werden kann die Daten zu ändern.

Erweitern der Visualisierungsmöglichkeiten. Wie im Kapitel 2 beschrieben, gibt es viele unterschiedliche Arten der Visualisierung. Die Literatur bietet aber noch viele weitere Beispiele und somit ergibt sich ein weites Spektrum an verschiedenen Möglichkeiten, die hier vorliegenden Daten graphisch aufzubereiten.

Da die hier vorliegenden Daten aus Text bestehen, ist eine Erweiterung hinsichtlich Volltextsuche und die damit verbundene graphische Darstellung der Ergebnisse ein weiterer wichtiger Erweiterungspunkt. Die Volltextsuche hilft dem Anwender sich schneller in der Thematik zurechtzufinden, vor allem dann, wenn das Ergebnis graphisch aufbereitet und mit verwandten Themen und Wörtern verlinkt wird.

Die Punkte ”History” und ”Extract” des Visualisierungsmottos von Shneiderman [1996] sind in dieser Arbeit nicht implementiert worden. Für den Anwender wäre dies eine Erleichterung in der Handhabung der Software, sowie in der Datenextraktion.

Für spätere industrielle Einsätze muss auch der Sicherheitsaspekt bedacht werden. Wer darf sich in das System einloggen und mit welchen Rechten. Fragen wie zum Beispiel, darf jeder Anwender immer alle Daten einsehen und/oder ändern, müssen zu diesem Zeitpunktbeantwortet sein.

Kapitel 8

Zusammenfassung

Die Arbeit beschäftigte sich mit der graphischen softwareunterstützten Visualisierung von gesammelten Daten in Form eines Anforderungskataloges, welcher durch das Projekt FuturePLM in mehrjähriger Arbeit entstanden ist.

Als erster Schritt war es wichtig eine Analyse der Daten durchzuführen, um sich mit der Thematik des Product Lifecycle Management in der Automobilindustrie eingehend zu beschäftigen. Dafür war es auch nötig, an mehreren Projektmeetings und Podiumsdiskussionen der Projektpartner teilzunehmen. Aufbauend auf diesem neu erworbenen Wissen und einer ausführlichen Literaturrecherche bezüglich Informationsvisualisierung, wurde eine Technologieauswahl für die vorzunehmende Implementierung getroffen. Persönliche Erfahrung, sowie Lizenzaspekte und Funktionalitäten bildeten den Schwerpunkt der Auswahl. Bei der Implementierung wurden mehrere Schwerpunkte an die Client-/Serversoftware gesetzt. Zum einen wurde eine Version implementiert, welche den Datensatz im internen serverseitigen Speicher hält und durch selbstgeschriebene Abfragen die Ergebnisse an den Client liefern. Als zweite Version wurde der komplette Datensatz automatisch in eine Ontologie bzw. in einen semantischen Datensatz mittels selbsterstelltem Framework umgewandelt, um in weiterer Folge Ergebnisse mittels SPARQL Abfragen zu liefern. Clientseitig kamen neue Technologien, die allgemein unter dem Begriff HTML5 bekannt sind, zum Einsatz. Diese Technologien wurden primär zum Visualisieren der Daten verwendet.

Anhang A

Weiterführende Javascriptbeispiele

Das Beispiel welches in Listing 4.4 gezeigt wird, wird hier im Appendix etwas ausgebaut, um auf gewisse Unterschiede zu klassenorientierten Programmiersprachen hinzuweisen.

A.1 Vererbte private Variablen

Es wird in dem Eltern Konstruktor eine private und ein public Variable (*privateVar* und *publicVar*) deklariert. Die Eigenschaften des Elternobjektes (*Parent*) werden an ein Kindobjekt (*Child*) vererbt. Die zwei Instanzen *child1* und *child2* erben alle Eigenschaften von *Child*. Jedoch gibt es einen Unterschied zwischen der privaten und public Variable. Die public Variable ist mit dem Schlüsselwort *this* definiert. Wird nun die Funktion getVariable() aufgerufen bekommt jede *child* Instanz eine eigene publicVar, jedoch wird die private Variable unter allen abgeleiteten Objekten von *Parent* geteilt. Das heißt, die private Variable von *Parent* verhält sich wie eine statische Variable.

A.2 Vererbte public Arrays

A.2.1 Statische vererbtes Array

Ähnlich wie im vorherigen Fall für private Variablen verhält es sich für abgeleitete public Objekte (wie zum Beispiel in Listing A.2 bei einem Array). Da der Konstruktor des Elternobjektes nur einmal mit *new* aufgerufen wird, teilen sich alle abgeleiteten Objekte dasselbe Objekt.

```
1  function Parent() {
2    var privateVar = 1;          //private Variable
3    this.publicVar = 1;    //public Variable
4    this.incrementVariables = function(){
5      privateVar++;
6      this.publicVar++;
7    };
8    this.getVariables = function(){
9      return {
10       priv : privateVar,
11       pub : this.publicVar
12     };
13   };
14 };
15
16 Child.prototype = new Parent();
17 Child.constructor = Child;
18 function Child() {};     //Kind Konstruktor
19
20 var child1 = new Child();
21 var child2 = new Child();
22 child1.incrementVariables();
23 console.log(child1.getVariables()); //Ergebnis: privateVar = 2, publicVar = 2
24 console.log(child2.getVariables()); //Ergebnis: privateVar = 2, publicVar = 1
```

Listing A.1: Vererbung

```
1  function Parent() {
2    this.publicArray = [];
3    this.fillPublicArray = function(){
4      this.publicArray[0] = 3;
5    };
6  };
7
8  Child.prototype = new Parent();
9  Child.constructor = Child;
10 function Child() {};
11
12 var child1 = new Child();
13 var child2 = new Child();
14 child1.fillPublicArray();
15 console.log(child1.publicArray);  //Ergebnis: 3
16 console.log(child2.publicArray);  //Ergebnis: 3
```

Listing A.2: Vererbung

A.2.2 Vervielfältigte public Arrays

Ist dies nicht gewünscht kann man mit Hilfe eines Patterns, beschrieben auf [Tanner, 2004], jedem abgeleiteten Objekt sein persönliches Array zuweisen. Listing A.3 zeigt, dass mit Hilfe der Funktion *setupData*, welche vom *Child* überschrieben wird für jedes neue Objekt ein neues Array angelegt wird. Der ”Trick” besteht darin, dass die *call* Funktion als ersten Parameter ein Objekt entgegen nimmt, welches dann im Funktionskörper der aufgerufenen Funktion mit *this* ansprechbar ist. Damit lässt sich für jedes abgeleitete Objekt ein eigenes Array erzeugen.

```
1  //Objekt Definitionen
2  Parent = function(){
3    this.setupData(); //privileged Methode
4  };
5
6  Parent.prototype.setupData = function(){  //public Methode
7    this.data = [];
8  };
9
10 Parent.prototype.init = function(val){
11   this.data[0] = val;
12 };
13
14 Child = function(){
15   this.setupData();   //Diese Funktion muss zu allererst aufgerufen werden
16 };
17
18 Child.prototype = new Parent();   //hier basiert der wesentliche Schritt bzgl. Vererbung
19 Child.prototype.constructor = Child;  //Ansonsten hat Child den Parent Konstruktor
20 Child.Parent = Parent.prototype; // Als Ersatz fuer super()
21
22
23 Child.prototype.setupData = function(){
24   Child.Parent.setupData.call(this);  //Damit bekommt jedes Child Objekt sein eigenes data Array
25 };
26
27 child1 = new Child();
28 child1.init("child1");
29 child2 = new Child();
30 console.log(child1.data); //Ergebnis: "child1"
31 console.log(child2.data); //Ergebnis: leeres Array
32 child2.init("child2");
33 console.log(child1.data); //Ergebnis: "child1"
34 console.log(child2.data); //Ergebnis: "child2"
```

Listing A.3: Vererbung

Literaturverzeichnis

Ball, W.W. R. [1905]. *Mathematical Recreations and Essays*. 4 Auflage. The MacMillan Company, New York. (Zitiert auf den Seiten 8 and 9.)

Battista, G., P. Eades, R. Tamassia, und I. G. Tollis [1999]. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, Inc., New Jersey. (Zitiert auf den Seiten 8, 21, 23, 24, 26, 27, 28, 29, 31, 33, 34 and 35.)

Belmonte, N. G. [2012]. *Javascript InfoVis Toolkit*. `http://philogb.github.com/jit/`. Zuletzt besucht am: 2012.12.02. (Zitiert auf Seite 62.)

Bostock, M. [2012a]. *Data-Driven Documents*. `www.d3js.org`. Zuletzt besucht am: 2012.12.02. (Zitiert auf Seite 63.)

Bostock, M. [2012b]. *Force Layout*. `https://github.com/mbostock/d3/wiki/Force-Layout`. Zuletzt besucht am: 2012.11.25. (Zitiert auf Seite 77.)

Bruls, M., K. Huizing, und J. van Wijk [1999]. *Squarified Treemaps*. In *In Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization*, Seiten 33–42. Press, Eindhoven. (Zitiert auf den Seiten 39 and 40.)

Burkhard, R. A. [2005]. *Knowlede Visualization*. Dissertation, Swiss Federal Institue of Technology Zurich. (Zitiert auf Seite 10.)

Buschmann, F., R. Meunier, H. Rohnert, P. Sommerlad, und M. Stal [1996]. *Pattern-Oriented Software Architecture Volume 1*. Wiley, West Sussex. (Zitiert auf den Seiten 50 and 51.)

Card, S.K., J.D. Mackinlay, und B. Shniderman [1999]. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers, San Francisco, CA. (Zitiert auf Seite 9.)

Cormen, Th. H., Ch. E. Leiserson, R. Rivest, und C. Stein [2009]. *Introduction to Algorithms*. 3rd edition Auflage. MIT Press, U.S. (Zitiert auf Seite 24.)

Dahlström, E., P. Dengler, A. Grasso, C. Lilley, C. McCormack, D. Scherpers, J. Watt, J. Ferraiolo, J. Fujisawa, und D. Jackson [2011]. *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. http://www.w3.org/TR/SVG/. Zuletzt besucht am: 2012.09.30. (Zitiert auf den Seiten 56 and 81.)

Daoud, F. [2008]. *Stripes... and Java web development is fun again*. First Edition Auflage. Pragmatic Bookshelf, U.S. (Zitiert auf den Seiten 49 and 50.)

Denger, A., M. Schmeja, W. Unzeitig, W. Schlüter, und F. Peschke [2011]. *Product Lifecycle Management (PLM) - Ein Blick auf den Mitarbeiter von Morgen*. *Produkt Daten Journal*, 18(2), Seiten 55–99. (Zitiert auf den Seiten 1 and 3.)

Denger, A., A. Stocker, und M. Schmeja [2012]. *Future Workplace - Eine Untersuchung sozio-technischer Einflüsse auf den Arbeitsplatz der Zukunft*. Aachen Shaker Verlag, Graz. (Zitiert auf Seite 2.)

Eades, P. [1984]. *A heuristic for graph drawing*. *Congressus Nutnerantiunt*. (Zitiert auf Seite 37.)

Eppstein, D. [2005]. *Nineteen Proofs of Euler's Formula: V-E+F=2*. http://www.ics.uci.edu/~eppstein/junkyard/euler/. Zuletzt besucht am: 2012.12.04. (Zitiert auf Seite 27.)

Flanagan, D. [2011]. *Javascript: The definitve Guide*. Sixth Auflage. O Reilly, U.S. (Zitiert auf den Seiten 58 and 59.)

Frame, J. [2012]. *tipsy - jQuery UI*. http://onehackoranother.com/projects/jquery/tipsy/. Zuletzt besucht am: 2012.12.02. (Zitiert auf Seite 61.)

Fruchterman, T. M. J. und E. M. Reingold [1991]. *Graph Drawing by Force-directed Placement*. *Software-Practice and Experience*, 21, Seiten 1129–1164. (Zitiert auf den Seiten 35, 36 and 37.)

Gamma, E., R. Helm, R. Johnson, und J. Vlissides [1994]. *Design Patterns*. Addison-Wesley, U.S. (Zitiert auf den Seiten 49, 50, 51, 58 and 88.)

Herman, I., G. Melancon, und M. S. Marshall [2000]. *Graph visualization and navigation in information visualization: A survey*. *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS*, 6(1), Seiten 24–43. (Zitiert auf Seite 21.)

Hickson, I. [2010]. *Web Database*. http://www.w3.org/TR/webdatabase/. Zuletzt besucht am: 2012.09.09. (Zitiert auf Seite 54.)

Hickson, I. [2011a]. *HTML5*. http://www.w3.org/TR/2011/WD-html5-20110525/. Zuletzt besucht am: 2012.11.29. (Zitiert auf Seite 53.)

Hickson, I. [2011b]. *Web Storage*. http://www.w3.org/TR/webstorage/. Zuletzt besucht am: 2012.09.09. (Zitiert auf Seite 54.)

Hitzler, P., M. Krötzsch, S. Rudolph, und Y. Sure [2008]. *Semantic Web*. Springer-Verlag, Berlin Heidelberg. (Zitiert auf Seite 87.)

IBM Research [2012]. *Many Eyes*. http://www-958.ibm.com/software/data/cognos/manyeyes/. Zuletzt besucht am: 2012.10.02. (Zitiert auf Seite 18.)

Johnson, B. und B. Shneiderman [1991]. *Tree-maps: a space-filling approach to the visualization of hierarchical information structures*. *IEEE Conference Publications*, Seiten 284 –291. (Zitiert auf den Seiten 38 and 39.)

Kuratowski, C. [1930]. *Sur the problème des courbes gauches en Topologie*. *Fund. Math.*, 15, Seiten 271–283. (Zitiert auf Seite 28.)

Mehta, N., J. Sicking, E. Graff, A. Popescu, und J. Orlow [2012]. *Indexed DB - W3C*. http://www.w3.org/TR/IndexedDB/. Zuletzt besucht am: 2012.09.09. (Zitiert auf Seite 55.)

Paley, W.B. [2002]. *TextArc: Showing Word Frequency and Distribution in Text*. *IEEE Symposium on Information visualization*. (Zitiert auf Seite 19.)

Prud'hommeaux, E. und A. Seaborne [2008]. *SPARQL Query Language for RDF*. http://www.w3.org/TR/rdf-sparql-query/. 2012.10.15. (Zitiert auf den Seiten 86 and 87.)

Purchase, H. [1997]. *Which Aesthetic has the Greatest Effect on Human Understanding?* In *Proceedings of the 5th International Symposium on Graph Drawing*, Seiten 248–261. GD '97, Springer-Verlag, Austrialia. (Zitiert auf den Seiten 29, 30 and 31.)

Rhyne, T., M. Tory, T. Munzner, M. Ward, C. Johnson, und D.H. Laidlaw [2003]. *Information and scientific visualization: separate but equal or happy together at last*. North Carolina, 611 -614 Seiten. (Zitiert auf Seite 10.)

Schmeja, M. und A. Denger [2011]. *Future PLM - Ansätze für ein mitarbeiterzentriertes PLM*. In Abromovice, M. (Herausgeber), *Product Life live 2011 Anwenderkongress für PDM und PLM 20.-21. September 2011, Stuttgart, Tagungsband*, Seiten 129–136. VDE VERLAG, Stuttgart. (Zitiert auf Seite 2.)

Shneiderman, B. [1996]. *The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations*. *Proc. IEEE Symp. Visual Languages, IEEE Press*, Seiten 336–343. (Zitiert auf den Seiten 5, 9, 13, 14 and 97.)

Smith, M. K., C. Welty, und D. L. McGuinness [2004]. *OWL Web Ontology Language Guide*. `http://www.w3.org/TR/2004/REC-owl-guide-20040210/`. Zuletzt besucht am: 2012.10.10. (Zitiert auf den Seiten 86 and 87.)

Spence, R. [2007]. *Information Visualization*. PEARSON Prentice Hall, U.S. (Zitiert auf den Seiten 6, 9 and 14.)

Stanford Center for Biomedical Informatics Research [2012]. *The Protege Ontology Editor and Knowledge Acquisition System*. `http://protege.stanford.edu/`. Zuletzt besucht am: 2012.08.15. (Zitiert auf Seite 91.)

Stasko, J., R. Catrambone, M. Guzdial, und K. McDonald [2000]. *An Evaluation of Space-Filling Information Visualizations for Depicting Hierarchical Structures*. *INTERNATIONAL JOURNAL OF HUMAN-COMPUTER STUDIES*, 53, Seiten 663–694. (Zitiert auf Seite 40.)

Stasko, J. und E. Zhang [2000]. *Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations*. In *Information Visualization, 2000. InfoVis 2000. IEEE Symposium on*, Seiten 57 –65. Georgia Institute of Technology, Atlanta, GA. (Zitiert auf Seite 39.)

Sugiyama, K. und K. Misue [1995]. *Graph Drawing by the Magnetic Spring Model*. *Journal of Visual Languages and Computing*, 6(3), Seiten 217–231. (Zitiert auf Seite 38.)

Sugiyama, K., S. Tagawa, und M. Toda [1981]. *Methods for Visual Understanding of Hierarchical System Structures*. *Systems, Man and Cybernetics, IEEE Transactions on*, 11(2), Seiten 109 –125. (Zitiert auf den Seiten 32 and 33.)

Tamassia, R., G. Di Battista, und C. Batini [1988]. *Automatic graph drawing and readability of diagrams*. *Systems, Man and Cybernetics, IEEE Transactions on*, 18(1), Seiten 61 –79. (Zitiert auf Seite 32.)

Tanner, M. A. [2004]. *OOP in Javascript*. http://www.brain-dump.org/docs/oop-in-js/inheritance.html. Zuletzt besucht am: 2012.08.30. (Zitiert auf Seite 101.)

The Apache Software Foundation [2012]. *Apache Jena*. http://jena.apache.org/. Zuletzt besucht am: 2012.07.09. (Zitiert auf Seite 90.)

Tory, M. und T. Möller [2004]. *Rethinking Visualization: A High-Level Taxonomy*. *Information Visualization, 2004. INFOVIS 2004. IEEE Symposium*, Seiten 151–158. (Zitiert auf den Seiten 10, 12 and 13.)

Ward, M., G. Grinstein, und D. Keim [2010]. *Interactive Data Visualization*. A K Peters, Ltd, U.S. (Zitiert auf den Seiten 16, 17 and 21.)

Warfield, J. N. [1977]. *Crossing Theory and Hierarchy Mapping*. *Systems, Man and Cybernetics, IEEE Transactions on*, 7(7), Seiten 505 –523. (Zitiert auf den Seiten 32, 33 and 34.)

Wattenberg, M. und F.B. Viegas [2008]. *The Word Tree, an Interactive Visual Concordance*. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6), Seiten 1221 –1228. (Zitiert auf den Seiten 17 and 18.)