Praktikumsbericht

Interaktive Visualisierung von RDF-Graphen

Dominique Bau
(Matrikelnummer 2314969)

Abgabedatum:	26. März 2020
Betreuer:	Prof. Dr. Joachim Baumeister
	Prof. Dr. Alexander Wolff

Julius-Maximilians-Universität Würzburg
Lehrstuhl für Informatik I
Algorithmen, Komplexität und wissensbasierte Systeme

Inhaltsverzeichnis

1 Rahmenbedingungen des Praktikums

Das Praktikum wurde zusammen mit Joachim Baumeister von der in Würzburg ansässigen Firma *denkbares GmbH*, welche sich im Bereich Informationsmanagement und Wissensmanagement bewegt, und Professor Alexander Wolff vom Lehrstuhl für Informatik I an der Julius-Maximilians-Universität durchgeführt. Ziel ist es, industrielle Daten eines *RDF-Graphen* (Resource Description Framework) übersichtlich zu visualisieren. Ein RDF-Graph ist ein gerichteter Graph mit der besonderen Eigenschaft, dass jede Kante und jeder Knoten im Graphen durch einen *URI* (Uniform Resource Identifier) identifiziert und mithilfe eines Labels benannt werden kann. Die Label der Kanten geben beispielsweise mehr Auskunft über die Relation zwischen den verbundenen Knoten. Die Semantik der URIs und Labels kann in einem RDF-Datenmodell namens *RDFS* (Resource Description Framework Schema) explizit definiert werden.

1.1 Datenaufbereitung

Um den Graphen letztendlich zu visualisieren, müssen die Daten zuerst aufbereitet werden. Ein wichtiger Aspekt der Aufbereitung ist es, die Daten in ein hierarchisches Format zu überführen, sodass der RDF-Graph in Clustern dargestellt werden kann. Die Aufbereitung wurde in diesem Praktikum mithilfe einer eigenständigen Java-Klasse realisiert, welche bei Ausführung aus einer *ttl*-Datei (Terse RDF Triple Language) eine *csv*-Datei (Comma Separated Values) generiert. Dafür wird die der W3C spezifisch für RDF-Graphen standardisierte Abfragesprache *SPARQL* (SPARQL Protocol And RDF Query Language) innerhalb des *Jena Semantic Web Frameworks* namens *ARQ* verwendet. Nach der Aufbereitung der Daten wird die in diesem Praktikum verwendete Baumstruktur knapp 200.000 Knoten umfassen.

Des Weiteren soll der geclusterte Graph mithilfe der JavaScript-Bibliothek *d3.js* (Data-Driven Documents) visualisiert werden. Diese Bibliothek haben Bostock, Ogievetsky und Heer [BOH11] entwickelt. Sie bietet einige Funktionen zur Visualisierung von (unter anderem) Graphen im Browser an. Ein großer Vorteil der Bibliothek ist die Unterstützung des *SVG*-Formats (Scalable Vector Graphics), welches auch in diesem Projekt verwendet wurde. In diesem Grafikformat ist es möglich, Bilder beliebig zu skalieren, ohne dass sie verpixeln oder sich andere unerwünschte Artefakte bilden.

Das primäre Ziel der Visualisierung ist, zum einen die Dichte des Graphen so stark wie möglich zu reduzieren, ohne wichtige Relationen zu verlieren und zum anderen die Größe eines Clusters proportional zur Anzahl der enthaltenden Knoten darzustellen. Im Zuge des Praktikums wurden zwei Darstellungsmöglichkeiten implementiert, welche verschiedene Ansätze in Bezug auf die Übersichtlichkeit bieten.

1.2 Treemap

Als erstes wurde eine *Treemap* gewählt. Die Darstellung von hierarchischen Daten und somit auch von Bäumen als Treemap wurde erstmals von Johnson und Shneiderman [JS99] beschrieben. In einer Treemap wird jeder Knoten eines Baumes als eine rechteckige

Fläche dargestellt. Ein Beispiel eines Baumes und der dazugehörigen Treemap kann in Abbildung 1 betrachtet werden. Das Ausschlaggebende bei dieser Darstellung ist, dass die rechteckigen Bereiche genau wie im Baum ineinander verschachtelt sind. Somit befindet sich der Bereich eines Knotens in allen Bereichen der Knoten des Pfades von sich bis zur Wurzel des Baumes. Lediglich der Bereich der Wurzel umfasst die ganze Zeichenfläche. Die Größe der jeweiligen Bereiche hängt dabei von einer Gewichtung der jeweiligen Teilbäume ab. Eine solche Gewichtung ist zulässig, falls für jeden Knoten gilt, dass das eigene Gewicht mindestens so groß ist wie die Summe der Gewichte der Kinder des Knotens. Eine zulässige Gewichtung der Knoten ist beispielsweise die Anzahl der Knoten, die sich in dem Teilbaum befinden, dessen Wurzel der jeweilige Knoten ist. In einer solchen Treemap entspricht die Tiefe der Verschachtlungen genau der Tiefe der Hierarchie.

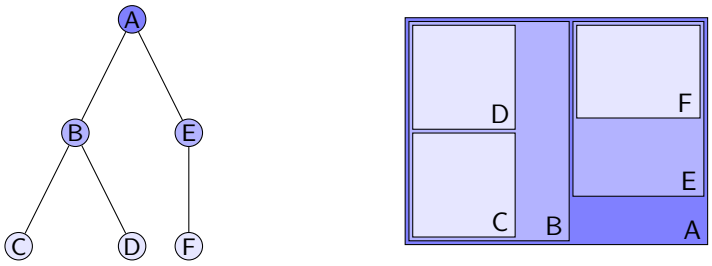

Abb. 1: Darstellung eines Baumes (a) und seiner Treemap (b), wobei die Gewichtung eines Knotens die Anzahl der Knoten im jeweiligen Teilbaum inklusive dem Knoten selbst ist.

1.3 Ein kräftebasiertes Layoutverfahren

Die zweite gewählte Darstellungsmethode ist ein kräftebasiertes Layoutverfahren für gewöhnliche Graphen. Das kräftebasierte Layoutverfahren basiert auf der Idee von Fruchterman und Reingold [FR91], dass sich Knoten in der Grafik abstoßen und Kanten sich wie Federn verlängern beziehungsweise verkürzen, um zu einer festgelegten Länge zu gelangen. Um die flüssige Benutzbarkeit des Applets gewährleisten zu können, können nicht alle Knoten gleichzeitig dargestellt werden, da es sich oft um sehr große Graphen handelt. Daher werden bei dieser Methode verschiedene Anzeigeoptionen zur Verfügung stehen. Die standardmäßige Darstellung zeigt lediglich einen Knoten und seine unmittelbaren Kinder an. Durch Klicken auf die Kinderknoten kann man innerhalb des Baumes absteigen, und durch Klicken des momentanen Wurzelknotens kann man in der Baumstruktur aufsteigen. Eine weitere Darstellungsoption zeigt für die momentane Wurzel mehr als nur seine unmittelbaren Kinder an. Es werden solange rekursiv auch die Kinder der Kinder angezeigt, bis die Anzahl der dargestellten Knoten die Grenze von 200 übersteigt. Zusätzlich gibt es auch eine Anzeigeoption, die für die momentane Wurzel den gesamten Teilbaum darstellt. Diese Option sollte jedoch nicht für die Wurzel des ge-

samten Baumes gewählt werden, da der Browser nach einigen Minuten meist abstürzt.

Die Ergebnisse der beiden Darstellungen können in Abbildung 2 und 3 betrachtet werden. Die Implementierung der Applets und des Parsers befinden sich im GitLab der Uni Würzburg.

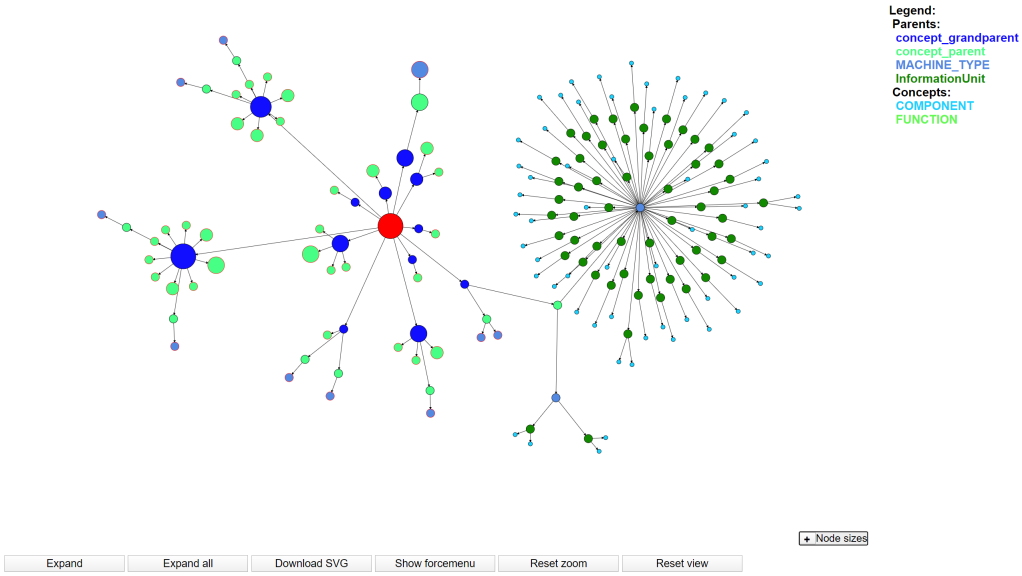

Abb. 2: Screenshot des kräftebasierten Ansatzes mit der erweiterten Anzeigeoption

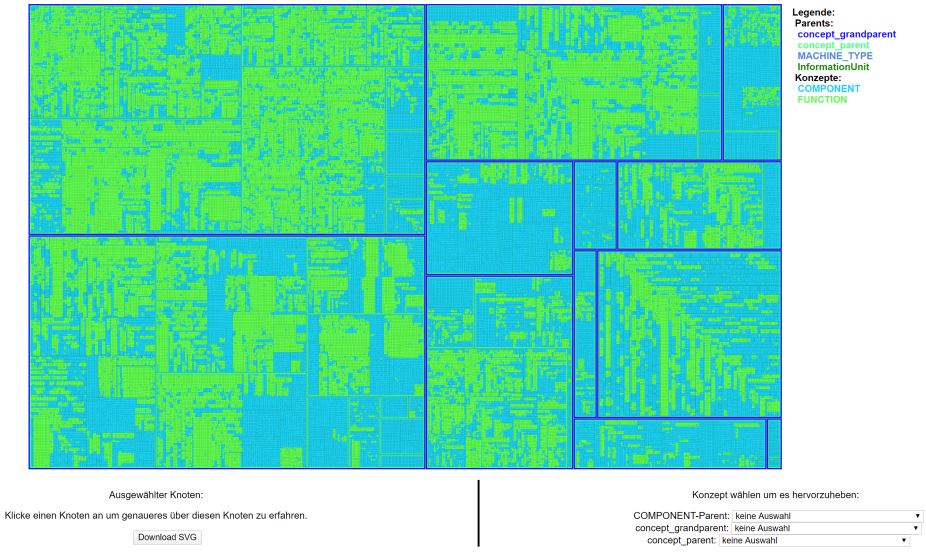

Abb. 3: Screenshot der Treemap-Visualisierung

2 Format der Daten

Die von denkbares GmbH erhaltenen Daten befinden sich im *ttl*-Format (Terse RDF Triple Language) und beinhalten die technische Dokumentation komplexer Fahrzeuge aus der Landtechnik. Im Nachfolgenden werden die für diese Visualisierung wichtigen Komponenten der Daten und ihr Aufbau erklärt. Darin beinhaltet eine Informationseinheit namens „InformationUnit“ genau einen Eintrag für „MACHINE_TYPE“ und jeweils beliebig viele „COMPONENT“- und „FUNCTION“-Einträge. In Abbildung 4 wird eine solche InformationUnit grafisch in einem UML-Diagramm dargestellt. Eine InformationUnit repräsentiert in diesem Datensatz ein Kapitel, in dem beschrieben wird, wie beispielsweise das Getriebe einer Landmaschine repariert werden kann. Der MACHINE_TYPE-Eintrag beschreibt eine bestimmte Ausführung einer Landmaschine. In einem COMPONENT-Eintrag befindet sich, falls in der dazugehörigen InformationUnit-Instanz und somit in einem gewissen Kapitel die Rede von einem gewissen Getriebe ist, die Beschreibung des Getriebes. Der FUNCTION-Eintrag beinhaltet Informationen über eine Funktion, die in dem jeweiligen Kapitel erwähnt wird, wie beispielsweise das Schalten in den 1. Gang. Der MACHINE_TYPE- und COMPONENT-Eintrag tragen eine Information zu ihrem jeweiligen „concept_parent“, zudem besitzt der MACHINE_TYPE-Eintrag noch ein „concept_grandparent“. Das concept_grandparent einer MACHINE_TYPE-Instanz sagt aus, um welche Art von Landmaschine es sich handelt. Zum Beispiel, ob es ein Traktor oder ein Mähdrescher ist. Das concept_parent einer MACHINE_TYPE-Instanz gibt Auskunft darüber, um welche Produktlinie es sich bei dieser Maschine handelt. Bei dem concept_parent eines COMPONENT-Eintrags handelt es sich hingegen um eine übergeordnete Komponente. Wäre beispielsweise das COMPONENT eine Schraube im Motorblock, wäre die übergeordnete Komponente der Motor. Innerhalb des Beispieldatensatzes gibt es mehrere Zehntausende solcher InformationUnit-Instanzen.

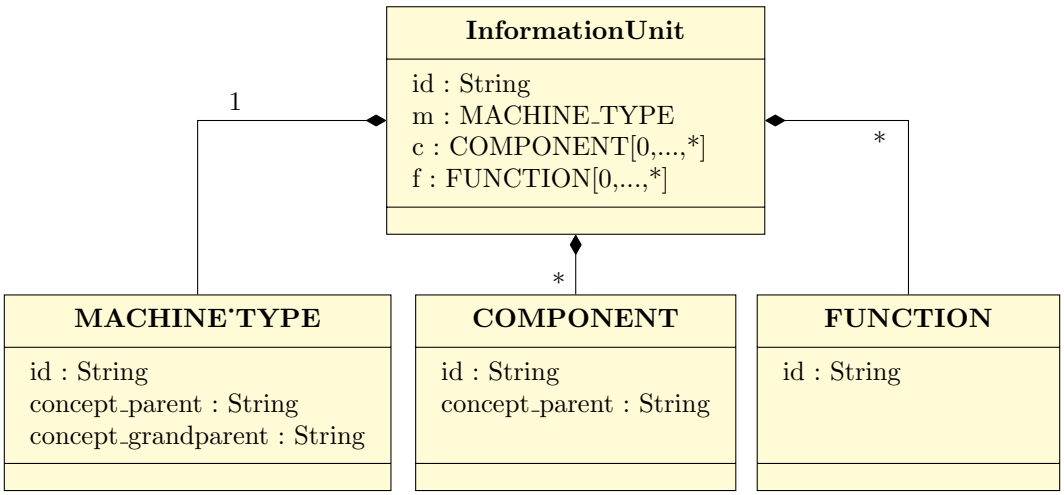

Abb. 4: UML-Diagramm der Daten vor der Aufbereitung des Parsers

3 Umsetzung

Der Parser wurde in Java implementiert und ist zuständig für die Informationsextraktion aus dem RDF-Graphen und die Überführung in ein hierarchisches Format dieser Daten. Abgespeichert werden die Daten danach als csv-Datei. Die beiden Applets wurden in JavaScript und mithilfe der d3.js Bibliothek implementiert. Die Treemap[1] und das kräftebasierte Layoutverfahren[2] können auch online mit maskiertem Datensatz getestet werden[3].

3.1 Der Parser

Bei dem Parser handelt es sich um eine alleinstehende Java-Klasse, welche bei Ausführung aus einer ttl-Datei eine csv-Datei generiert. Mithilfe der Abfragesprache SPARQL werden die einzelnen InformationUnits (siehe Kapitel 2) aus dem Datensatz extrahiert. Die SPARQL-Anfrage kann in Abbildung 5 betrachtet werden. Dabei werden für jede InformationUnit die in Kapitel 2 beschriebenen Daten extrahiert. Überflüssige Segmente der extrahierten Strings werden darauf hin sofort entfernt, um unnötig große Datenmengen zu vermeiden. Anschließend werden die Daten in das gewünschte csv-Format überführt. Dabei soll eine hierarchische Baumstruktur erstellt werden, die sich wie folgt zusammensetzt. Die Wurzel des Baumes wird durch einen Dummy-Knoten namens „Origin“ realisiert. Da jede InformationUnit-Instanz genau einen MACHINE_TYPE-Eintrag beinhaltet und verschiedene InformationUnit-Instanzen den gleichen MACHINE_TYPE-Eintrag besitzen können, kann für die hierarchische Struktur diese Beziehung umgedreht werden, sodass eine MACHINE_TYPE-Instanz mehrere InformationUnit-Einträge beinhaltet. Diese InformationUnit-Instanzen tragen dann nur noch die Informationen über ihre COMPONENTs und ihre FUNCTIONs. Dementsprechend besteht das erste Level der Hierarchie aus den verschiedenen concept_grandparent-Instanzen, gefolgt von den jeweiligen concept_parent-Instanzen der MACHINE_TYPEs. Das dritte Level belegen die MACHINE_TYPE-Instanzen selbst. Die Kinder der MACHINE_TYPEs sind die InformationUnits, welche im ursprünglichen Format den jeweiligen MACHINE_TYPE-Eintrag trugen. Das letzte Level teilen sich alle COMPONENT- und FUNCTION-Instanzen. Dabei kann es vorkommen, dass mehrere InformationUnits die gleiche FUNCTION oder das gleiche COMPONENT beinhalten. Um die gewünschte Baumstruktur zu erreichen, werden aus dem einen Element an dieser Stelle mehrere Elemente, sodass jede InformationUnit seine eigene Instanz des Elements besitzt. Zum besseren Verständnis ist die Hierarchie auch in Abbildung 6 dargestellt.

In der csv-Datei wird eine Relation durch eine Zeile repräsentiert, in der es insgesamt fünf Einträge gibt. Der erste Wert einer Zeile ist die ID des momentanen Elements, gefolgt von der ID des Elternknotens. Der einzige Knoten, der keinen Elternknoten besitzt, ist, wie es bei einem Baum üblich ist, die Wurzel beziehungsweise der Dummy-Knoten.

[1] Treemap unter `https://www.domi-bau.de/prakt-test/treemap/`

[2] Der kräftebasierte Ansatz unter `https://www.domi-bau.de/prakt-test/force/`

[3] Das Laden der Seiten kann je nach Internetverbindung einige Zeit in Anspruch nehmen, da es sich um eine große Datenmenge handelt. Besonders die Treemap benötigt einige Zeit zum Laden.

```
SELECT ?informationUnit_uri ?concept_uri ?concept_type2 ?concept_parent
    ?concept_grandparent
 WHERE {
  ?informationUnit_uri a ssc:InformationUnit ;
  ssc:hasRelevantConcept/ssc:hasConcept ?concept_uri .
  ?concept_uri rdf:type ?concept_type .
  OPTIONAL {
   ?concept_uri <http://www.claas.com/casis#belongsToProductLine>
       ?concept_parent
  }
  OPTIONAL {
   ?concept_uri <http://www.claas.com/casis#belongsToProductLine>/
   <http://www.claas.com/casis#belongsToProductFamily> ?concept_grandparent
  }
  OPTIONAL {
   ?concept_uri <http://www.claas.com/casis#subBgzOf> ?concept_parent
  }
  BIND(
   IF(?concept_type = casis:BGZ, \"COMPONENT\",
   IF(?concept_type = casis:CCN, \"FUNCTION\",
   IF(?concept_type = casis:MachineType, \"MACHINE_TYPE\",
   \"NO TYPE\")))
   AS ?concept_type2)
 }
```

Abb. 5: Die SPARQL-Anfrage zum Extrahieren der Daten aus der ttl-Datei.

Der dritte und vierte Eintrag sind der Typ des momentanen Elements (z.B. MACHINE_TYPE oder InformationUnit) und die ID des concept_parents (falls vorhanden, sonst `null`). Der letzte Wert ist eine Zahl, die letztendlich bestimmt, wie groß das Feld des momentanen Knotens in der Treemap werden soll. In der momentanen Implementierung handelt es sich dabei um den Wert 20, wenn es sich um ein Blatt des Baumes handelt (FUNCTION oder COMPONENT). Ansonsten beträgt der Wert standardmäßig 1. Mehr zu diesem Wert in Kapitel 3.2.

3.2 Die Treemap

Um die Treemap zu realisieren, wird zunächst die `stratify()`-Funktion der d3.js Bibliothek benutzt. Diese Funktion fasst einzelne Objekte und ihre Informationen zusammen und gibt den Wurzelknoten des Baumes zurück, welcher auch alle direkten Kinder als Einträge besitzt. Diese Kinder haben wiederum ihre Kinder als Einträge und so weiter. Außerdem werden in diesem Zuge die Zahlenwerte der Elemente in der csv-Datei (20 falls es ein Blatt ist und sonst 1) für jeden Teilbaum aufsummiert und das Ergebnis in einem neuen Eintrag der jeweiligen Wurzel des Teilbaums gespeichert. Somit hat die Wurzel des gesamten Baumes zuletzt die Summe aller Knotenwerte als neuen Eintrag.

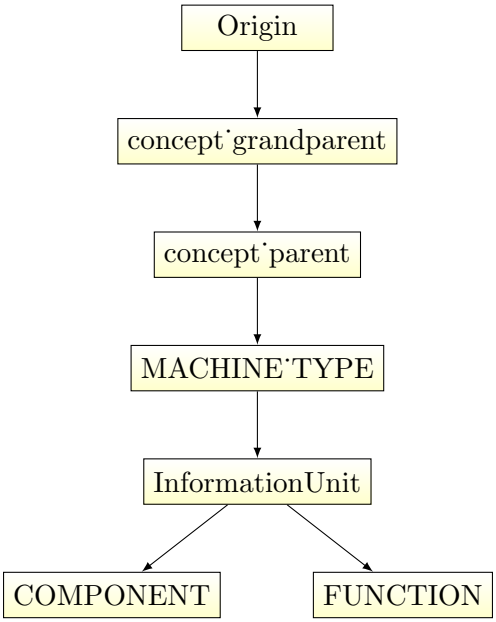

Abb. 6: Darstellung der Hierarchie der Daten nach der Überarbeitung des Parsers. Ein Pfeil von A zu B bedeutet, dass B in der Hierarchie unmittelbar nach A kommt.

Wenn im Folgenden von einem Knotenwert die Rede ist, handelt es sich stets um diesen neuen aufsummierten Wert.

Die Funktion `treemap()`, welche ebenfalls aus der d3.js Bibliothek stammt, weist nun jedem Teilbaum und somit auch letztendlich jedem Knoten einen Platz in der Grafik zu. Wobei die Größe des Platzes abhängig von dem Knotenwert ist. Die Wurzel im Besonderen bekommt den gesamten Platz der Grafik zugewiesen. Für jeden nachfolgenden Knoten gilt, dass dieser prozentual genau soviel Platz des Elternknotens bekommt, sodass diese Relation dem Verhältnis der beiden Knotenwerte entspricht. Demnach werden die Knoten mit zunehmendem Level in der Hierarchie immer weiter ineinander verschachtelt. Ein Beispiel für eine solche Treemap kann in Abbildung 1 betrachtet werden. Dadurch, dass die inneren Knoten des Baumes in der csv-Datei einen Wert von 1 besitzen, ist der aufsummierte Wert eines inneren Knotens um 1 größer als die Summe seiner Kinder. Dies bedeutet, dass nicht der gesamte Platz des Knotens von seinen Kindern übernommen wird, wodurch jeweils der Rand der Fläche noch zu diesem Knoten gehört. Im SVG werden dementsprechend Rechtecke gezeichnet, welche mit den dazugehörigen Daten durch d3.js verknüpft werden. Ändert man etwas an den Daten, findet d3.js das entsprechende Rechteck und passt es auf Befehl gemäß der Änderung an.

Eine Funktionalität der Treemap über die Bibliotheksfunktionen von d3.js hinaus ist das Markieren von einzelnen Knoten in der Treemap durch einen Linksklick auf den Knoten. Dadurch werden alle Knoten die dieselbe ID haben[4] farblich hervorgehoben.

[4]Blätter können dieselbe ID besitzen, da vom Parser von manchen Blättern mehrere Instanzen erstellt werden. Siehe Kapitel 3.1

Außerdem werden die ID des angeklickten Knotens, dessen Typ und die ID des Elternknotens textuell unter der Grafik festgehalten. Durch einen weiteren Linksklick in die Grafik wird die Auswahl wieder aufgehoben und alle Knoten erhalten wieder die ursprünglichen Farben. In Abbildung 7 wurde ein Knoten angeklickt. Die Anzeige seiner ID, seines Typs und der ID des Elternknotens wird durch die Nummer 1 gekennzeichnet.

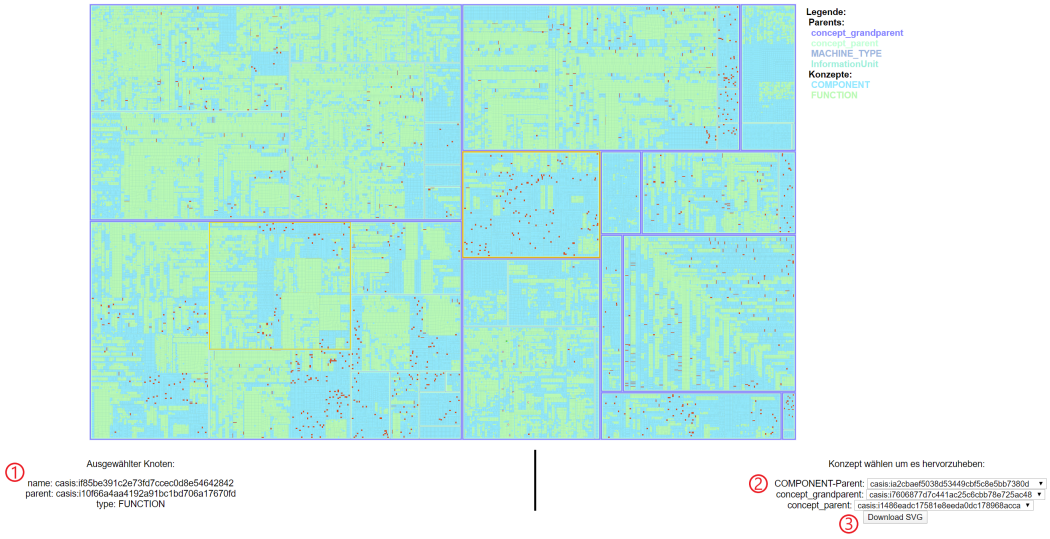

Abb. 7: Die Treemap-Visualisierung mit Nummerierung der Funktionalitäten. Hierbei wurde sowohl ein Knoten angeklickt, als auch in jedem Dropdown-Menü ein Eintrag ausgewählt.

Des Weiteren befinden sich unter der Grafik mehrere Dropdown-Menüs (siehe Abbildung 7 bei Nummer 2) mit dessen Hilfe man beispielsweise alle COMPONENT-Knoten, die ein bestimmtes concept_parent besitzen, hervorheben kann. Die Hervorhebung der Knoten wird erst wieder mit dem Umschalten des Menüs zu „keine Auswahl“ gelöscht. Diese Menüs werden dynamisch mit allen möglichen Einträgen für dieses Feld gefüllt. In Abbildung 7 wurde in jedem Dropdown-Menü ein Eintrag ausgewählt.

Außerdem gibt es einen „Download SVG“-Knopf, mit dem die Grafik separat heruntergeladen werden kann. Dieser befindet sich in Abbildung 7 bei der Nummer 3.

3.3 Der kräftebasierte Ansatz

Auch im kräftebasierten Ansatz wird als erstes die `stratify()`-Funktion verwendet. Jedoch werden hier die Werte aus der csv-Datei ignoriert und jeder Knoten bekommt vor dem Aufsummieren einen Wert von 1. Somit ist der Wert für einen Knoten nach dem Aufsummieren gleich der Anzahl der Knoten des Teilbaums, dessen Wurzel der jeweilige Knoten ist.

Anschließend wird die `forceSimulation()`-Funktion von d3.js verwendet. Diese Funktion baut auf der Idee von Fruchterman und Reingold [FR91] auf, dass sich Knoten in der Grafik abstoßen und Kanten sich wie Federn verlängern beziehungsweise verkürzen, um zu einer festgelegten Länge zu gelangen.

Beim Start des Applets werden in der Grafik nur die Wurzel des Baumes und seine unmittelbaren Kinder dargestellt. Die Größe eines Knotens ist dabei abhängig von der Anzahl der Knoten, die sich in dem Teilbaum unter dem jeweiligen Knoten befinden. Damit verschieden große Knoten auch mit bloßem Auge direkt erkennbar sind, wurden die Größen der Knoten in Stufen eingeteilt, sodass Knoten aus verschiedenen Stufen unterscheidbar sind. Dafür wurde das Intervall von der kleinsten bis zur größten vorkommenden Kinderanzahl in logarithmisch skalierte Subintervalle aufgeteilt. Je nachdem in welchem Subintervall sich die Kinderanzahl eines bestimmten Knoten befindet, bekommt der Knoten die dazugehörige Größe zugeteilt. Die Intervalle und die dazugehörigen Größen werden rechts unten in einer ausklappbaren Tabelle mit der Aufschrift „Node sizes“ dargestellt. Diese Tabelle wird in Abbildung 8 angezeigt, dabei ist zu beachten, dass die Knotengrößen in der Tabelle nur mit denen in der Grafik übereinstimmen, wenn nicht gezoomt wurde.

Innerhalb der Grafik kann per Mausrad rein und raus gezoomt, durch Klicken und Ziehen des Hintergrunds der Bildausschnitt verschoben und durch Klicken und Ziehen eines Knotens ebendieser zu einer danach festen Position verschoben werden. Um nun in der Grafik auch andere Level der Hierarchie zu sehen, kann auf einen beliebigen Knoten gedrückt werden. Sofern der gedrückte Knoten nicht die Wurzel des momentan angezeigten Teilbaumes oder ein Blatt des gesamten Baumes ist, wird dieser Knoten jetzt zur Wurzel und seine unmittelbaren Kinder werden um ihn herum angezeigt. Handelte es sich um die Wurzel des momentan angezeigten Teilbaumes, so wird wenn möglich ein Level in der Hierarchie heraus navigiert. Somit wird der Elternknoten des gedrückten Knotens zur momentanen Wurzel und dazu werden seine unmittelbaren Kinder angezeigt. Ein Klick auf den „Origin“-Knoten oder auf ein Blatt des gesamten Baumes bewirkt nichts. Durch das Halten der SSTRGTaste und dem gleichzeitigen Klicken eines Knotens kann der Pfad des Knotens zur momentanen Wurzel hervorgehoben werden. Dabei werden die Knoten entlang des Pfades gelb gefüllt (siehe Abbildung 8).

Außerdem gibt es verschiedene Knöpfe unterhalb der Grafik. Diese Knöpfe können auch in Abbildung 8 betrachtet werden.

Der „Expand“-Knopf führt dazu, dass nicht nur die unmittelbaren Kinder der momentanen Wurzel angezeigt werden, sondern dass auch einige weitere Knoten des Baumes dargestellt werden. In Abbildung 8 wurde eben diese Anzeigeoption ausgewählt. In der momentanen Implementierung werden die Kinder von angezeigten Knoten mit nur einem Kind immer angezeigt. Sonst werden so lange die Kinder von bereits angezeigten Knoten ebenfalls angezeigt, bis sich über 200 Knoten in der Grafik befinden. Knoten, die Kinder besitzen, welche jedoch nicht angezeigt werden, haben eine rote Umrandung, wohingegen Knoten, deren Kinder angezeigt werden, eine schwarze Umrandung haben. Die Schranke für die Anzahl der angezeigten Knoten ist willkürlich gesetzt und kann noch durch einen besseren Algorithmus ersetzt werden, ist aber notwendig, da sonst das Applet eine sehr lange Zeit zum Aktualisieren benötigt oder gar abstürzt. Besonders das Erweitern von Knoten im tieferen Bereich des Baumes während ein anderer Teilbaum schon sehr weit oben im Baum nicht weiter dargestellt wird, gibt einen falschen Eindruck über die Größe des jeweiligen Teilbaums. Möchte man dennoch den gesamten Teilbaum unter der momentanen Wurzel angezeigt bekommen, kann der „Expand all“-Knopf gedrückt

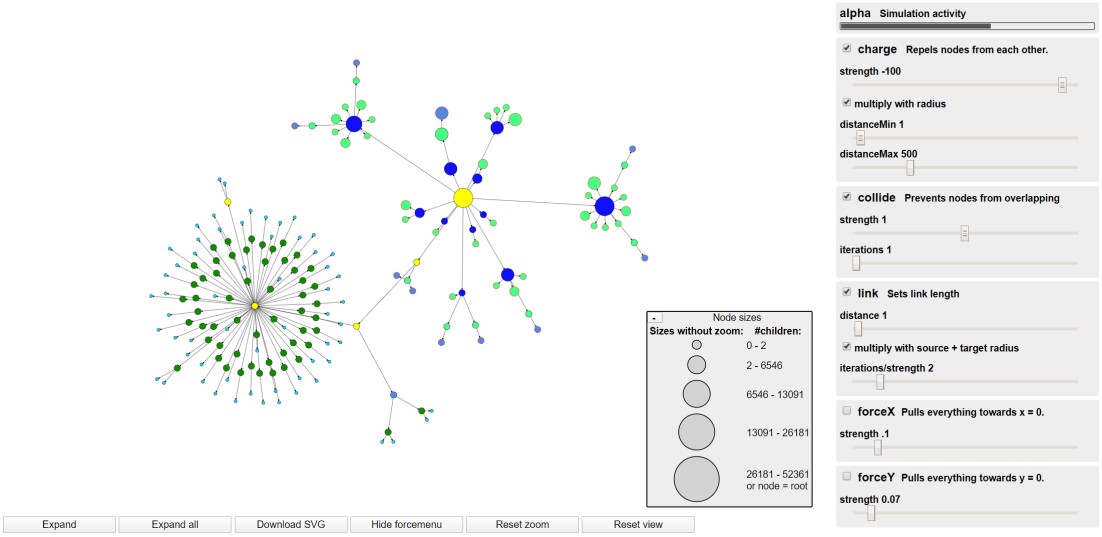

Abb. 8: Die kräftebasierte Visualisierung mit STRG-Klick hervorgehobenen Knoten, ausgeklapptem Forcemenu, ausgeklappter Knotengrößen-Tabelle und erweiterter Anzeige.

werden.

Mit dem „Download SVG“-Knopf kann, wie der Name schon verrät, die momentane Grafik als SVG heruntergeladen werden.

Der „Show forcemenu“-Knopf tauscht die Legende auf der rechten Seite durch ein Menü, in welchem die Kräfte die zur Berechnung der Knotenpositionen verwendet werden, angepasst werden können (siehe Abbildung8). Außerdem wird der „Show forcemenu“-Knopf zu einem „Hide forcemenu“-Knopf, welcher das Menü wieder durch die Legende austauscht. Dabei bleiben jedoch die eingestellten Kräfte erhalten. Im Kräftemenü befindet sich ganz oben eine Leiste mit der Überschrift „alpha“, welche das Potential des kräftebasierten Algorithmus darstellt. Je höher das Potential ist, umso schneller können sich die Knoten in der Grafik noch bewegen. Demnach ist die Animation und damit das Zeichnen des Graphen abgeschlossen, sobald in dieser alpha-Leiste kein schwarzer Balken mehr zu erkennen ist. Um das Potential auf das Maximum zurückzusetzen, genügt es auf die alpha-Leiste zu klicken. Darunter folgen mehrere Kästen, die jeweils eine Überschrift mit einer Checkbox haben. Wenn diese Checkbox keinen Haken hat, wird diese Kraftart in der Berechnung komplett ausgeschlossen und auch Änderungen innerhalb des jeweiligen Kastens bewirken nichts. Mithilfe der Schieberegler können die verschiedenen Attribute der Kräfte angepasst werden. Eine Veränderung bewirkt einen sofortigen Neustart der Animation durch Zurücksetzen des Potentials auf das Maximum.

Der nächste Knopf namens „Reset zoom“ setzt lediglich den Bildausschnitt zurück auf den Ursprung. Der „Reset view“-Knopf hingegen setzt alle Kräfte und alle vom Benutzer veränderten Einstellungen oder Interaktionen mit der Grafik zurück.

4 Anwendungsszenarien

Die Treemap wurde primär implementiert, um einen groben Überblick über einen großen Datensatz zu erhalten. Dabei kann man einen ersten Eindruck von beispielsweise dem Mengenverhältnis der Daten zweier concept_parents bekommen und gleichzeitig herausfinden, ob es innerhalb des jeweiligen concept_parents eher mehr COMPONENTs oder FUNCTIONs gibt.

Das kräftebasierte Layoutverfahren hingegen ist für die Exploration eines solchen Datensatzes erstellt worden. Mit dessen Hilfe kann man auf jedem Level sofort erkennen, wie groß die Teilbäume der momentanen Kinder sind. Dies ist vor allem auf den unteren Levels der Hierarchie übersichtlicher als in der Treemap.

Außerdem sind beide Darstellungen leichter zu verstehen als eine Textdatei. Dies ermöglicht es auch Menschen ohne technischem Hintergrund die Daten zu visualisieren und zu explorieren. Zusätzlich können damit die Daten eines Kunden visualisiert und anhand der Grafik besser beschrieben werden.

5 Verbesserungen

Eine Verbesserung beziehungsweise Erweiterung der Applets, welche auch schon während des Praktikums des Öfteren in Betracht gezogen wurde, wäre eine zoombare Treemap zu implementieren. Beispielsweise könnte man die hier erstellte Treemap benutzen und sobald ein Element angeklickt wird, werden nur noch das Element und seine unmittelbaren Kinder oder Elternknoten angezeigt. Dadurch könnte man eventuell die Vorteile der beiden Applets kombinieren.

Zusätzlich wäre es wünschenswert, die Hierarchie und die Farbgebung mithilfe einer „config“-Datei ändern zu können, sodass auch Daten eines anderen Namensraumes dargestellt werden können und man personalisierte Farben vergeben kann.

Außerdem könnte das Interface um das SVG herum generell überarbeitet werden, um den Gebrauch der Applets intuitiver zu gestalten.

Da das SVG der Treemap auch beim Öffnen der schon fertig erstellten Datei sehr viel Zeit benötigt, könnte eine Änderung beziehungsweise eine Komprimierung des SVGs beim Herunterladen durch den Nutzer die Anzeige beschleunigen. Die normale Dateigröße des Treemap-SVGs beträgt knapp 43 MB. Durch eine Umbenennung der Variablen im SVG auf kurzmöglichste Strings und das Löschen von Daten, die für die letztendliche statische Anzeige irrelevant sind, kann die Dateigröße auf knapp 14 MB verringert werden. Das Öffnen der verbesserten SVG-Datei benötigt dementsprechend auch weniger Zeit.

Wie auch schon in Kapitel 3.3 beschrieben, könnte die erweiterte Darstellung, welche nicht immer alle Knoten darstellt, überarbeitet werden. Besonders sollte darauf geachtet werden, dass der Pfad zur Wurzel von allen Blättern des angezeigten Baumes in etwa gleich lang ist.

6 Fazit

Zusammenfassend kann gesagt werden, dass die ersten Darstellungen der Daten als Treemap und mithilfe eines kräftebasierten Verfahren erfolgreich waren, jedoch noch viel Spielraum für Verbesserungen bieten. Insbesondere ist eine Erkenntnis aus diesem Praktikum, dass ein Browser momentan noch nicht schnell genug ist, um einen so großen Graphen als SVG im Gesamten darzustellen.

Literatur

[BOH11] Michael Bostock, Vadim Ogievetsky und Jeffrey Heer: D^3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

[FR91] Thomas MJ Fruchterman und Edward M Reingold: Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991.

[JS99] Brian Johnson und Ben Shneiderman: Tree-Maps: A Space-Filling Approach to the Visualization of Hierarchical. In: *Readings in Information Visualization: Using Vision to Think*, Seiten 152–159. Morgan Kaufmann, 1999.