Institut für Visualisierung und Interaktive Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Studienarbeit Nr. 2415

Entwicklung eines Konzepts zur Annotation von grafischen Elementen in Visualisierungen

Stefan Strohmaier

Studiengang:	Informatik
Prüfer:	Prof. Dr. Thomas Ertl
Betreuer:	Dipl.-Phys. Michael Raschke, Dipl.-Inf. Tanja Blascheck
begonnen am:	21.12.2012
beendet am:	21.06.2013
CR-Klassifikation:	I.3.6, H.5.4, I.2.4

Kurzfassung

Einer der zeitaufwendigsten Aspekte bei der Durchführung von Eye-Tracking Studien im Bereich der Visualisierung ist, neben dem Entwurf und dem Design der Studie, vor allem die Auswertung der Blickdaten. Diese Blickdaten werden in Form von einzelnen Positionen mit Hilfe eines Eye-Trackers aufgezeichnet. Anschließend müssen von Hand diejenigen Elemente einer Visualisierung identifiziert werden, die sich an den gemessenen Positionen befinden. Durch diese Abbildung von Positionen auf visuelle Elemente wird das Ziehen von Schlüssen über die Vorgehensweise eines Probanden und dessen kognitive Prozesse beim Betrachten von Visualisierungen möglich.
Das Ziel dieser Studienarbeit ist, ein Konzept zu entwickeln, welches die Annotation von grafischen Elementen in Visualisierungen durch semantische Informationen aus einer Visualisierungs-Ontologie ermöglicht. Anhand eines Fixationspunkts eines Probanden kann aufgrund der Annotierung dann automatisch bestimmt werden, welche visuellen Elemente sich an dieser Position befinden, sowie welche Metainformationen diese enthalten. Der implementierte Prototyp demonstriert, dass das Konzept in bestehende Visualisierungs-Frameworks integriert werden kann. Dazu wurden außerdem drei verschiedene Visualisierungstechniken beispielhaft semantisch annotiert.

Abstract

One of the most time-consuming aspects when carrying out eye tracking studies in the field of visualization is, besides the outline and the design of the study, the analysis of the gaze data. The gaze data is collected as several positions by using an eye-tracker. Afterwards the elements of a visualization, which are located at the measured positions, have to be identified manually. This mapping from positions to visual elements enhances the user's ability to draw conclusions about the approach of a subject and its cognitive processes while viewing visualizations.
The aim of this thesis is to develop a concept, which allows the annotation of graphical elements in visualizations by using semantic information from a visualization ontology. Due to the annotation, it is possible to automatically determine which visual elements are located at a fixation point of a subject and which meta information the perceived elements contain. The implemented prototype demonstrates that the concept can be integrated into existing visualization frameworks. Furthermore three different visualization techniques have been semantically annotated as an example.

Inhaltsverzeichnis

1 Einleitung

Die menschliche Wahrnehmung erfolgt durch die Aufnahme unterschiedlichster Reize aus der Umwelt unter Verwendung der entsprechenden Sinnesorgane. Hierbei spielt der Bereich der visuellen Wahrnehmung eine wichtige Rolle, da, mit ca. 80% der größte Anteil an Informationen über die Umgebung mit den Augen aufgenommen wird. Das Forschungsgebiet der Visualisierung beschäftigt sich daher mit der grafischen Aufbereitung und Darstellung abstrakter Daten, mit dem Ziel besondere Muster sichtbar zu machen, um daraus letztendlich Schlüsse über die zugrunde liegenden Sachverhalte ziehen zu können. Ein frühes Beispiel für die Visualisierung eines vielschichtigen Datensatzes stellt die sogenannte „Minard Map“ [17] von Charles Joseph Minard aus dem Jahr 1869 dar. Sie zeigt in kompakter Form den Rußlandfeldzug Napoleons, wobei eine Vielzahl an Variablen, wie beispielsweise die Truppenstärke, die Marschrichtung der Armee, die Temperatur während des Feldzugs usw., berücksichtigt wurde ohne die Lesbarkeit zu beeinträchtigen. Durch die voranschreitende Entwicklung von Computern und das damit zusammenhängende Anwachsen der Datenmengen, wie beispielsweise Messdaten von Sensoren, Texte, Bilder oder Videos, entwickelten sich schließlich rechnergestütze Visualisierungstechniken. Diese zeichnen sich besonders durch die Möglichkeit aus, die gerenderte Visualisierung mit Hilfe von Interaktionskonzepten weiter zu manipulieren. So kann unter anderem durch den Einsatz von Zooming & Panning [3] der auf dem Bildschirm dargestellte Ausschnitt angepasst werden, was dem Benutzer den Eindruck vermittelt, er könne sich dadurch im Datensatz bewegen. Da der Entwicklungsprozess solcher Visualisierungen häufig zeitlich aufwendig ist und Kenntnisse aus verschiedenen Bereichen der Informatik erfordert, entstand die Idee Visualisierungs-Frameworks zu entwerfen, welche häufig benötigte Komponenten und Schnittstellen (Graph-Layout Algorithmen, verschiedene Renderingmechanismen, etc.) bereitstellen ohne dabei die Erweiterbarkeit oder das Anwendungsgebiet allzu sehr einzuschränken.

Eine in den letzten Jahren immer mehr in den Vordergrund gerückte Technik, welche unter anderem zur Evaluation von Benutzeroberflächen oder Webseiten angewandt wird, ist das sogenannte Eye-Tracking Experiment. Hierbei werden die Bewegungen der Augen eines Probanden in einer Studie mit Hilfe technischer Geräte erfasst und aufgezeichnet. Bei den ersten Experimenten dieser Art standen jedoch noch kaum verwertbare Hilfsmittel zur Verfügung, weshalb die Augenbewegungen lediglich von einer anwesenden Person durch einen Spiegel betrachtet werden konnten. Louis Émile Javal fand dabei bereits 1879

heraus, dass sich das menschliche Auge beim Lesen von Texten nicht gleichmäßig über die einzelnen Zeichen hinwegbewegt, sondern eine Reihe schneller, kurzer Bewegungen ausführt (Sakkaden), die durch ebenfalls kurze Pausen (Fixationen) voneinander getrennt sind. Moderne Eye-Tracking Systeme erfassen Blickpositionen unter Verwendung eines Infrarot-Emitters mittlerweile mit einer Frequenz von ca. 60 Hz pro Auge [29] und liefern damit die Grundlage für eine umfangreiche Erforschung der kognitiven und perzeptiven Vorgänge beim Betrachten verschiedener Stimuli. Bevor allerdings mit der Interpretation der gemessenen Daten begonnen werden kann, muss zunächst eine visuelle Repräsentation gewählt werden, welche es ermöglicht tausende von Messpunkten in geeigneter Form darzustellen. Hierfür existieren mehrere verschiedene Ansätze. Ist ein grober Überblick darüber gewünscht, welche Stellen des Bildschirms von einem Proband am meisten betrachtet wurden, empfiehlt sich eine Heatmap Visualisierung. Gaze Plots zeigen zudem neben den Fixationen zusätzlich auch die Sakkaden der Augenbewegungen an. Dies geschieht indem die einzelnen Fixationen, welche üblicherweise als Kreise repräsentiert sind, durch Linien verbunden werden. Die Dauer einer Fixation kann hierbei auf den Radius des entsprechenden Kreises abgebildet werden. Eine neuartige Methode die Blickdaten mehrerer Probanden in einer übersichtlichen Form zu visualisieren bieten Parallel Scan-Paths [23]. Nachdem der Benutzer einige wichtige Bereiche (Areas of Interest, AOIs) auf dem Bildschirm markiert hat, werden hierbei in einem parallelen Gaze Plot nur Augenbewegungen, die zwischen diesen Bereichen stattgefunden haben, eingezeichnet.

Bei der Betrachtung von gerenderten Visualisierungen durch einen Mensch, lassen sich unterschiedliche Objekte erkennen und dadurch schließlich Aussagen treffen. Aus Sicht der Maschine handelt es sich bei diesen Objekten allerdings lediglich um eine Anzahl an Pixeln, die keinerlei fest definierte Bedeutung besitzt. Um einen für Menschen und Maschinen verständlichen Begriff der Semantik zu formen, wurde infolgedessen das Konzept entwickelt, Wissen in Ontologien zu formulieren. Eine Ontologie stellt dabei einen gerichteten Graph dar, der als Knoten verschiedene Konzepte (Klassen) und die konkreten Ausprägungen dieser Konzepte (Instanzen) enthält, während die Kanten die Beziehungen (Relationen) zwischen diesen Knoten ausdrücken. Auf dieser Grundlage lassen sich nun Bilder, Videos, und andere Daten semantisch annotieren. Der Prozess der Annotation lässt sich entweder automatisiert [27] oder manuell von Benutzern durchführen [26], was stark abhängig davon ist, wie die Daten beschaffen sind und in welcher Qualität die Annotation benötigt wird.

In den bisher existierenden Visualisierungs-Frameworks ist es allerdings nicht möglich, Semantik in Form von Ontologien mit den Visualisierungen zu verknüpfen. Beim Durchführen von Eye-Tracking Studien zur Evaluation von Visualisierungstechniken fehlen daher Informationen über die semantische Bedeutung der einzelnen grafischen Elemente. Solche Informationen können in Zukunft dazu verwendet werden, die Auswertung

von Eye-Tracking Studien zu vereinfachen. In dieser Arbeit soll daher ein Konzept zur automatischen Annotation von Visualisierungs-Elementen entwickelt werden, welches vorsieht die Semantik in den Rendering-Prozess zu integrieren und somit die grafische Repräsentation direkt mit dem semantischen Wissen zu koppeln. Neben der Entwicklung eines Prototyps soll untersucht werden, inwiefern sich das Konzept in bereits bestehende Frameworks einbetten lässt.

1.1 Aufbau der Arbeit

Dieses Dokument wurde in insgesamt sieben Kapitel unterteilt. Nach dem Einleitungskapitel, welches eine Einführung in das bearbeitete Thema sowie eine kurze Motivation enthält, folgen die zum Verständnis der Arbeit erforderlichen Grundlagen. Kapitel 3 behandelt Ansätze, die sich mit ähnlichen Teilproblemen wie diese Studienarbeit befassen. Hierbei werden bereits existierende Visualisierungs-Frameworks genauer untersucht. Nach der Formulierung der konkreten Aufgabenstellung wird daraufhin anschließend die Herleitung des Lösungsansatzes in Kapitel 4 beschrieben. Auf Grundlage dieses Ansatzes wird in Kapitel 5 unter verschiedenen Gesichtspunkten ein detailliertes Lösungskonzept hergeleitet. Der Implementierungsteil beschreibt einige der zentralen Aspekte der technischen Umsetzung des Lösungskonzepts, worauf schließlich in Kapitel 7 die Ergebnisse der Arbeit präsentiert und diskutiert werden.

2 Grundlagen

Die Grundlagen dieser Arbeit setzen sich aus den beiden Teilgebieten Web Technologien und Visualisierung zusammen. Die einzelnen Begriffe werden hier zunächst definiert und anschließend genauer ausgeführt. Dadurch soll das erforderliche Wissen bereitgestellt werden, welches zum Verständnis der folgenden Kapitel erforderlich ist.

2.1 Semantisches Web und Web Technologien

In diesem Abschnitt sollen einige wichtige Technologien des Semantischen Web sowie des Web 2.0 vorgestellt werden. So können Ontologien mit Hilfe des Resource Description Frameworks (RDF) oder der Web Ontology Language (OWL) realisiert werden und anschließend zur semantischen Annotation von Inhalten wie z. B. Text, Grafiken oder Videos verwendet werden. Scalable Vector Graphics (SVG) stellen eine Form solcher Inhalte dar. Diese werden im Webbrowser unter Verwendung des Document Object Model (DOM) repräsentiert.

2.1.1 Defintion Ontologie

Anders als in der Philosophie wird der Ontologiebegriff in der Informatik im Zusammenhang mit der formalen Beschreibung von Wissen verwendet. Thomas Gruber definiert eine Ontologie als „eine explizite formale Spezifikation einer Konzeptualisierung“ [13]. Eine Konzeptualisierung stellt in diesem Fall ein vereinfachtes theoretisches Modell der Welt dar, welches durch eindeutig identifizierbare Objekte (Entitäten), und Verbindungen (Relationen) zwischen diesen Objekten beschrieben werden kann. Das Ziel ist es, einzelne Wissensbereiche (Domänen) in einer sowohl für Menschen als auch für Maschinen verständlichen, einheitlichen Darstellung zu modellieren, um dadurch den Wissensaustausch sowie das automatische Ableiten (Inferenz) von neuem Wissen zu ermöglichen. Der Inferenzprozess wird dabei von einem Reasoning-Algorithmus übernommen, der die Ontologie unter anderem um transitive Eigenschaften erweitert und somit die Voraussetzung für die anschließende Weiterverarbeitung durch Anfragesprachen wie SPARQL schaffen. Ein Anwendungsbereich von Ontologien ist beispielsweise das Semantische Web [1]. Das Semantische Web bezeichnet ein aus unterschiedlichen Technologien bestehendes Konzept

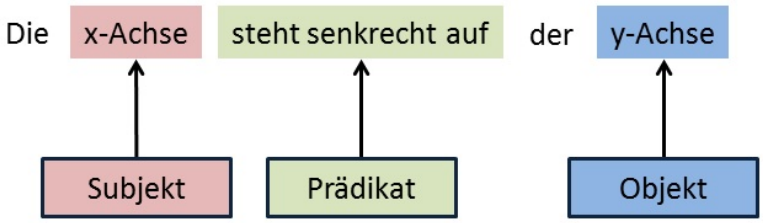

Abbildung 2.1: Beispiel für den Zusammenhang zwischen einem Satz in einer natürlichen Sprache und einer Aussage in RDF. Die drei Bestandteile Subjekt, Prädikat und Objekt sind jeweils entsprechend ihrem zugehörigen Satzteil farblich markiert.

des World Wide Web Consortium (W3C), welches den Wandel des Web 2.0 hin zu einem Internet der Dinge skizziert. Dabei sollen Texte, Bilder, Videos sowie sämtliche anderen Inhalte des Internets Schritt für Schritt semantisch annotiert und somit in eine geordnete Form gebracht werden (siehe Abschnitt 2.1.4).

2.1.2 Resource Description Framework (RDF)

Das Resource Description Framework (RDF) ist ein Datenmodell, mit dessen Hilfe es möglich ist, einfache Aussagen (Statements) über Daten in Form von Tripeln zu formulieren. Die vom W3C veröffentlichte Spezifikation [7] legt fest, dass ein solches Statement aus den folgenden drei Bestandteilen besteht: Subjekt, Prädikat und Objekt. Diese Bestandteile werden als sogenannte Ressourcen angelegt und sind durch Uniform Resource Identifier (URIs) global eindeutig identifizierbar. Neben existierenden Ressourcen können Objekte außerdem auch durch Literale, wie beispielsweise Zahlen oder Strings, angegeben werden. Analog zu einem Satz in einer natürlichen Sprache, bezeichnet das Subjekt eines Statements über welche Ressource eine Aussage gemacht werden soll, während das Objekt angibt mit welcher Ressource das Subjekt in Verbindung steht. Das Prädikat beschreibt schließlich die Art dieser Verbindung zwischen den beiden Entitäten. Ein Beispiel hierfür ist in Abbildung 2.1 dargestellt.
Mit dem Ziel auch komplexere Phänomene der realen Welt abbilden zu können, wurde RDF im Jahr 2004 um einige Bestandteile erweitert und unter dem Namen RDF-S (RDF-Schema) standardisiert. Die wichtigste Neuerung war hierbei die Einführung eines Klassenkonzepts wie es in objektorientierten Programmiersprachen üblich ist. Dies eröffnet die Möglichkeit Vererbungshierarchien von Klassen aufzubauen und Subjekte sowie Objekte eindeutig als Instanz oder Klasse zu deklarieren. Auch eine Einschränkung der Prädikate auf bestimmte Klassen von Subjekten (Domain) und auf bestimmte Klassen von Objekten (Range) lässt sich dadurch realisieren.

```
1  <?xml version="1.0"?>
2  <!-- RDF namespace -->
3  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
4      <!-- Subjekt des Statements -->
5      <rdf:Description rdf:about="http://www.example.org/XAchse">
6          <!-- Prädikat des Statements -->
7            <ex:stehtSenkrechtAuf>
8              <!-- Objekt des Statements -->
9              <rdf:Description rdf:about="http://www.example.org/YAchse"/>
10         </ex:stehtSenkrechtAuf>
11     </rdf:Description>
12 </rdf:RDF>
```

Quellcode 2.1: Beispiel eines RDF Tripels in XML-Schreibweise. Subjekt und Objekt werden hierbei durch ein `rdf:Description` Tag realisiert. Entsprechende Prädikate befinden sich geschachtelt innerhalb des zugehörigen Subjekt Tags.

XML-basierte Darstellung

Da die RDF Spezifikation keine bestimmte Repräsentation oder Schreibweise vorgibt, existieren verschiedene Möglichkeiten die einzelnen Tripel der Statements darzustellen. Am weitesten verbreitet ist dabei die Verwendung von Extensible Markup Language (XML) Syntax. Hierfür muss lediglich ein XML Dokument angelegt werden, in welches die entsprechenden RDF Namespaces (Quellcode 2.1, Zeile 3) anschließend eingebettet werden können.
Alle Subjekte und diejenigen Objekte, die nicht als Literale realisiert sind, werden durch ein `rdf:Description` oder ein `rdfs:Class` Tag gekennzeichnet, dessen `rdf:about` Attribut die entsprechende URI der Ressource enthält (Quellcode 2.1, Zeile 5). Geschachtelt innerhalb des Subjekt Tags befinden sich sämtliche Relationen, die sich auf diese Entität beziehen (Quellcode 2.1, Zeile 7).

Graph-basierte Darstellung

XML kann mit Hilfe einer Vielzahl an existierenden Werkzeugen effizient geparst, maschinell verarbeitet und in Dateien geschrieben werden. Allerdings ist die Syntax für den Menschen aufgrund des verwendeten Schachtelungsprinzips schwer nachvollziehbar. Interpretiert man die RDF Tripel der formulierten Statements als einen gerichteten Graph, legt dies eine graphische Repräsentation als Knoten-Kanten Diagramm nahe (siehe Abbildung 2.2). Subjekte und Objekte werden in einem solchen Diagramm durch die visuelle Metapher eines Knotens dargestellt, während die verschiedenen Relationen als Kanten zwischen den Knoten im Graph eingezeichnet werden.

Abbildung 2.2: Darstellung einer Aussage in RDF als Graph. Subjekt und Prädikat werden hierbei jeweils als Knoten, das Prädikat als Kante gezeichnet.

```
1 <!-- Subjekt des Statements -->
2 <div typeof="Person">
3    <!-- Prädikat und Objekt des Statements -->
4    <span property="hatName">Peter Schmidt</span>
5 </div>
```

Quellcode 2.2: Eine mit RDFa vorgenommene Annotation eines HTML Elements. Das Attribut `typeof` bezeichnet hierbei die Instanz einer Klasse und definiert somit das Subjekt der Aussage. Mit Hilfe des `property` Attributs wird eine Relation definiert, deren Objekt das Literal „Peter Schmidt“ darstellt.

RDF in Attributes (RDFa) und Mikroformate

Eine weitere Möglichkeit Dokumente mit semantischen Informationen zu versehen bietet RDF in Attributes (RDFa) [20]. Durch die im Juni 2012 vom W3C veröffentlichte Empfehlung können RDF-Statements, anders als bei reinem RDF, direkt in die entsprechenden HTML- oder XML-Attribute der Dokumente eingebettet werden. Dies hat den Vorteil, dass der Inhalt und die Semantik eines Dokuments fest miteinander gekoppelt sind und nicht räumlich voneinander getrennt in unterschiedlichen Abschnitten oder gar Dokumenten abgelegt werden müssen. Hierfür wurde zusätzlich zu den in (X)HTML verwendeten Attributen `content`, `href`, `rel`, `ref` und `src` eine Reihe neuer Attribute definiert, deren Namensgebung zum Großteil an die in RDF spezifizierte Terminologie angelehnt ist. Ein Subjekt-Prädikat-Objekt Tripel lässt sich so beispielsweise durch die Attribute `typeof` (Instanz einer Klasse), `property` (Prädikat) und einem Literal (Objekt) erstellen (siehe Quellcode 2.2).
Neben RDFa existiert außerdem eine ähnliche Technologie, die eine direkte Annotation von Dokumenten durch Metadaten erlaubt. Sogenannte Mikroformate stellen eine Erweiterung von HTML dar und verwenden ausschließlich die in (X)HTML gebräuchlichen Attribute `class`, `rel` sowie `rev`. Für unterschiedliche Domänen wie z. B. Personen, Kontaktinformationen und Reviews müssen hierbei allerdings jeweils eigene Mikroformate mit spezifischem Vokabular entwickelt werden. Im Gegensatz zu RDF/RDFa, welches unabhängig von der verwendeten Ontologie geparst werden kann, ist der Parse-Vorgang bei Mikroformaten vom entsprechenden Vokabular abhängig. Aus diesen Gründen ist es erheblich schwieriger neue Domänen und Anwendungsgebiete mit diesem Konzept zu verknüpfen.

2.1.3 Web Ontology Language (OWL)

Eine weitere Technologie des Semantischen Web, welche ebenfalls durch eine W3C Spezifikation definiert wird, ist die sogenannte Web Ontology Language (OWL) [25]. OWL baut aus technischer Sicht auf RDF auf und beinhaltet einige neue Konzepte, um semantische Informationen korrekt zu formalisieren. Dabei ist es unter anderem nicht nur möglich, spezielle Relationen zwischen einzelnen Instanzen der Klassen auszudrücken, sondern auch Mengenoperationen auf Klassen zu definieren. Die in Abschnitt 2.1.2 beschriebenen Relationen werden zusätzlich in Data Properties, also Relationen mit einem Literal als Objekt, und Object Properties, also Relationen zwischen zwei Instanzen, aufgeteilt. Außerdem lassen sich, neben inversen Relationen, auch Einschränkungen bezüglich der Kardinalität definieren, d.h. Einschränkung der Anzahl der Objekte, die diese Relation beinhalten kann.
OWL existiert in drei unterschiedlichen Ausprägungen, welche sich jeweils in ihrer Mächtigkeit unterscheiden: OWL Full, OWL DL sowie OWL Lite. OWL Full besitzt hierbei keinerlei Einschränkungen, was die Verwendung der vorhandenen Sprachkonstrukte angeht, hat aber den Nachteil, dass die Sprache nicht entscheidbar ist. Eine Ebene darunter befindet sich OWL DL, welches sich vollständig auf Prädikatenlogik erster Stufe abbilden lässt und aus diesem Grund besonders häufig Anwendung findet. Eine noch weiter vereinfachte Variante für die Darstellung von weniger komplexen Wissenbasen stellt schließlich OWL Lite dar.

2.1.4 Semantische Annotation

Semantische Annotation bezeichnet den Prozess der Anreicherung von unterschiedlichen Medien wie Dokumenten, Bildern, Videos usw. durch Metadaten. So können diese sowohl von Computern als auch von Menschen verarbeitet werden. Um solche Annotationen zu erhalten, müssen die entsprechenden relevanten Inhalte zunächst entweder maschinell oder von Hand extrahiert und anschließend mit Begriffen aus einer Wissensbasis, wie beispielsweise einer Ontologie, versehen werden.

2.1.5 Automatische semantische Annotation

Die Vision des Semantischen Web sieht vor, das Internet als Ganzes, inklusive aller vorhandenen Webseiten, mit semantischen Informationen zu versehen. Da dieser Vorgang unmöglich ohne Hilfsmittel zu bewerkstelligen ist, werden für unterschiedliche Datentypen jeweils Werkzeuge entwickelt, die halb-automatisch oder voll-automatisch semantische Annotationen erzeugen können. Während halb-automatische Verfahren nur vorverarbeitende Schritte ausführen, welche durch einen Benutzer vervollständigt werden müssen, können voll-automatische Verfahren selbstständig komplette Annotationen erzeugen.

In dieser Arbeit soll jedoch der Begriff der automatischen Annotation unter einem anderen Schwerpunkt betrachtet werden. Anstatt, wie im vorherigen Absatz beschrieben, bereits vollständige Bilddaten nachträglich zu annotieren, soll die Annotation bereits automatisch während dem Rendering-Prozess einer Visualisierung stattfinden. Aus diesem Grund sind konzeptionell keinerlei Techniken aus dem Gebiet Bildverstehen oder maschinelles Sehen nötig, um eine solche Annotation zu generieren.

2.1.6 Scalable Vector Graphics (SVG)

Unter dem Begriff der Bilddaten versteht man visuelle, für das Auge wahrnehmbare Daten, welche auf zwei unterschiedliche Arten digital repräsentiert werden können: als Rastergrafik oder als Vektorgrafik. Rastergrafiken bestehen aus einer Ansammlung von Pixeln (picture elements), die räumlich zueinander in einem Gitter angeordnet sind. Die Anzahl der Pixel eines Bildes bestimmt wie hoch dessen Auflösung ist und somit wie viele Informationen es enthält. Währenddessen charakterisiert die Menge an möglichen Farbwerten, die einzelne Pixel annehmen können, die entsprechende Farbtiefe einer solchen Grafik.
Vektorgrafiken hingegen sind nicht aus Pixeln, sondern aus einzelnen grafischen Primitiven wie z. B. Linien, Pfaden usw. aufgebaut. Innerhalb eines zuvor festgelegten Koordinatensystems werden, durch die Angabe einer oder mehreren Positionen sowie von zusätzlichen Attributen, die einzelnen grafischen Elemente definiert. Dies hat gegenüber Rastergrafiken den Vorteil, dass solche Grafiken beliebig skaliert werden können, ohne dass dabei Qualitätsverluste oder Artefakte auftreten. Ein vom W3C publizierte Standard für Scalable Vector Graphics 1.1 (SVG) [10] aus dem Jahr 2011 spezifiziert eine für das Internet geeignete Darstellung von Vektorgrafiken auf Grundlage von XML. Der dadurch festgelegte Sprachumfang von SVG kann von modernen Browsern heutzutage größtenteils vollständig ohne Zuhilfenahme von Plug-ins interpretiert werden kann. Grafische Primitive werden hierbei durch geschachtelte Tags realisiert, die mit verschiedenen Attributen für affine Transformationen oder für Style-Eigenschaften versehen werden können. Außerdem unterstützt wird die Gruppierung von Elementen, die Anwendung von Filtermatrizen sowie Animation mittels Skriptsprachen oder XML Tags.

2.1.7 Document Object Model (DOM)

(X)HTML und XML Dokumente werden grundlegend zunächst aus einzelnen geschachtelten Tags aufgebaut und anschließend von einem Webbrowser interpretiert. Um Skriptsprachen den dynamischen Zugriff auf die einzelnen Elemente dieser Dokumente zu ermöglichen, wurde vom W3C das sogenannte Document Object Model (DOM) [19] definiert. So wird ein Dokument im DOM als eine Baumstruktur, bestehend aus verschiedenen Knoten- und Beziehungstypen, repräsentiert. Es existieren neben Dokumentknoten

und Dokumentfragmentknoten, welche das gesamte Dokument oder Teile des Dokuments darstellen, auch Knoten für einzelne Elemente sowie deren Attribute und Inhalt. Die unterschiedlichen Beziehungstypen beschreiben Verbindungen zwischen Eltern- und Kindknoten sowie zwischen Geschwisterknoten.

2.2 Visualisierung

Im Folgenden sollen diejenigen für das in Kapitel 5 vorgestellte Lösungskonzept relevanten Begriffe behandelt werden, welche sich mit dem Themenkomplex Visualisierung befassen. Unter diesen Begriffen befinden sich unter anderem auch drei unterschiedliche Visualisierungstechniken, die im Zusammenhang mit der Implementierung in Kapitel 6 stehen. Nach einer Definition des Begriffs der Visualisierung, wird hierbei zunächst die Visualisierungs-Pipeline erläutert. Daraufhin wird ausgeführt, warum es hilfreich ist Visualisierungs-Frameworks zur Erstellung von Visualisierungen zu verwenden und aus welchen Bestandteilen solche Frameworks bestehen. Schließlich folgt eine kurze Beschreibung von Säulen- und Balkendiagrammen, Knoten-Kanten Diagrammen sowie Scatter Plots.

2.2.1 Definition Visualisierung

Der Begriff der Visualisierung bezeichnet eine Disziplin der Computergrafik, die sich damit beschäftigt, unterschiedliche Typen von Daten durch Abbildung auf eine visuelle Repräsentation als Grafiken oder Animationen sichtbar zu machen. Anhand ihres Anwendungsgebiets haben sich im Laufe der Jahre die beiden Teildisziplinen Informationsvisualisierung (InfoVis) und wissenschaftliche Visualisierung (SciVis) herausgebildet. Der signifikante Unterschied der beiden Teildisziplinen liegt hierbei bei den Datentypen, die visualisiert werden sollen. Während sich InfoVis mit rein abstrakten Daten beschäftigt, verarbeiten die meisten SciVis Anwendungen höherdimensionale oder mit geometrischen Informationen versehene Daten.

2.2.2 Visualisierungs-Pipeline

Die in Abbildung 2.3 dargestellte Grafik zeigt die Visualisierungs-Pipeline nach dos Santos und Brodlie [24]. Ihr vierstufiger Aufbau erläutert die einzelnen Schritte, die notwendig sind, um aus gegebenen Rohdaten visuelle Repräsentationen zu erstellen. Die grau eingefärbten Blöcke beschreiben hierbei die unterschiedlichen Operationen, die beim Durchlaufen der Pipeline nacheinander auf den Daten (blaue Container) ausgeführt werden. Im ersten Schritt wird zunächst die Datenanalyse durchgeführt, bei der, durch Anwendung von beispielsweise Glättungs-Operationen oder Interpolation, die Daten

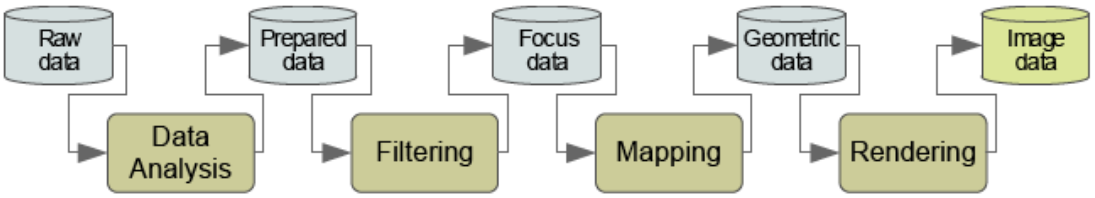

Abbildung 2.3: Die Visualisierungs-Pipeline nach dos Santos und Brodlie [24] beschreibt schematisch den Weg von Rohdaten bis hin zu einer Visualisierung. Hierfür werden in vier Schritten die Daten zunächst nach Fehlern analysiert (Data Analysis), gefiltert (Filtering), auf visuelle Elemente abgebildet (Mapping) und schließlich als Grafik gezeichnet (Rendering).

aufbereitet werden. Daraufhin folgt das Filtern, d.h. die Auswahl der für die Visualisierung relevanten Datenpunkte. Diese Datenpunkte werden anschließend auf grafische Primitive, wie Punkte oder Linien, abgebildet und mit den entsprechenden geometrischen Attributen versehen (Mapping). Im letzten Schritt werden schließlich, unter Verwendung grafischer Schnittstellen, die Bilddaten erzeugt. Dies geschieht indem die Primitive entsprechend ihrer zugewiesenen Attribute in eine Rastergrafik oder Vektorgrafik umgewandelt werden (Rendering).

2.2.3 Visualisierungs-Frameworks

Die Implementierung einer Visualisierungs-Anwendung zeichnet sich durch einen hohen Arbeitsaufwand aus und benötigt zusätzlich fortgeschrittene Kenntnisse in verschiedenen Disziplinen der Informatik, wie beispielsweise Computergrafik, Algorithmik, Mensch-Computer-Interaktion usw. Zwar wurde bereits eine Vielzahl unterschiedlichster Visualisierungs-Anwendungen entwickelt, jedoch sind deren Einsatzgebiete oft sehr auf die entsprechende Domäne beschränkt, für die die jeweilige Anwendung erstellt wurde. Frameworks dienen daher dazu den Entwicklungsprozess durch die Vorgabe von einem gewissen Maß an Rahmenfunktionalität zu vereinfachen und zu beschleunigen. Dazu werden beispielsweise Komponenten zur Ein- und Ausgabe von Daten, Datenspeicherung, Datentransformationen, Benutzerinteraktion etc., mit Hilfe von abstrakten Klassen und Interfaces in einer modularen Form ausprogrammiert und üblicherweise außerdem in einem Dokumentationsteil ausführlich beschrieben. Für den Entwickler der Anwendung beschränkt sich der Programmieraufwand daher größtenteils auf das Verbinden und Erweitern der vorgegebenen Module. Kapitel 3 führt dieses Thema weiter aus und zeigt den Aufbau solcher Frameworks anhand von vier Beispielen aus den Bereichen Informationsvisualisierung sowie wissenschaftlicher Visualisierung.

2.2.4 Säulen- und Balkendiagramme

Ein Säulen- bzw. Balkendiagramm ist eine weit verbreitete Visualisierungstechnik, welche häufig zum Vergleich verschiedener Variablen eingesetzt wird. Diese Art von Diagramm besteht aus entweder einer Achse oder zwei orthogonalen Achsen sowie mehreren Säulen oder Balken. Man verwendet den Begriff Säulendiagramm, falls für die einzelnen dargestellten Variablen vertikal angeordnete Rechtecke verwendet werden. Bei einer horizontalen Orientierung der Rechtecke wird die Grafik als Balkendiagramm bezeichnet. Die Höhe einer Säule bzw. die Breite eines Balkens kodiert dabei den Wert einer der Variable, abhängig von der gewählten Skala der Achsen.

2.2.5 Knoten-Kanten Diagramme

In der Graphentheorie bezeichnet ein Graph $G = (V,E)$ ein Tupel bestehend aus einer Menge an Knoten V und einer Menge an Kanten E zwischen den Knoten. An diese formale Definition ist keine feste visuelle Repräsentation gebunden. Es ist jedoch naheliegend, die Knoten als geometrische Formen wie Kreise, Ellipsen oder Rechtecke und die Kanten als Verbindungslinien darzustellen. Die dadurch entstehenden Diagramme werden Knoten-Kanten Diagramme oder auch Node-Link Diagramme genannt. Diese Darstellung eignet sich besonders für Graphen mit geringer Dichte und einer Anzahl von maximal 20 Knoten, da ansonsten vermehrt Kantenkreuzungen und Verdeckungen auftreten [12].
Die Lesbarkeit hängt außerdem stark von der Position der einzelnen Knoten im Diagramm, dem sogenannten Graph Layout, ab. Neben orthogonalen und kreisförmigen Layouts existieren hierfür außerdem Kräfte basierte Algorithmen, bei denen die Anordnung der Knoten durch eine physikalische Simulation erfolgt [11][15]. Die Knoten werden dabei als geladene Partikel modelliert und bilden dadurch eine abstoßende Kraft, welche auf die umliegenden Knoten wirkt. Die dem entgegengesetzte Kraft entsteht durch die Kanten des Graphs, die wie eine Art Feder betrachtet werden können und die Knoten aufeinander zu bewegen. Angefangen mit zufälligen Knotenpositionen stellt sich so nach einigen Iterationsschritten ein Kräftegleichgewicht und somit eine Knotenanordnung ein, welche Symmetrien im Graph sichtbar macht und versucht Überdeckungen von Knoten zu vermeiden.

2.2.6 Streudiagramme

Soll der Zusammenhang von den Werten zweier Variablen untersucht werden, so gibt es die Möglichkeit diese Werte in ein zweidimensionales Streudiagramm (engl. Scatter Plot) einzutragen. Streudiagramme bestehen aus jeweils einer horizontalen und einer vertikalen Achse, sowie pro Datenpunkt einem grafischen Element (Kreis, Quadrat, usw.) zur Repräsentation der Werte der Variablen.

3 Themenbezogene Arbeiten

In diesem Kapitel werden vier existierende Publikationen über Visualisierungs-Frameworks vorgestellt. Neben den beiden Frameworks InfoVis Toolkit (IVTK) und prefuse für den Bereich der Informationsvisualisierung, wird außerdem das für wissenschaftliche Visualisierung entwickelte Visualization Toolkit (VTK) sowie das webbasierte Framework Data-Driven Documents (D3) beschrieben. Im Vordergrund steht hierbei jeweils die Analyse des konzeptionellen Aufbaus der Frameworks, auf deren Grundlage in Kapitel 4 schließlich die Herleitung des Lösungskonzepts erfolgt.

3.1 Das InfoVis Toolkit

Fekete et al. [9] stellten 2004 ein von ihnen entwickeltes Toolkit vor, welches das zeitaufwändige und mühsame Programmieren von Visualisierungen im Bereich der Informationsvisualisierung erleichtern soll. Basierend auf Agile2D, einer performanten OpenGL Implementierung von Java2D, wurden vielseitig einsetzbare Komponenten, von generischen Datenstrukturen und Algorithmen bis hin zu kompletten Visualisierungen entwickelt. In ihrer Arbeit zeigen Fekete et al. neben verschiedenen Anwendungsgebieten auch mehrere Möglichkeiten, ihr Toolkit um zusätzliche Funktionalität, wie beispielsweise ein Werkzeug zur geometrischen Verzerrung, zu erweitern.

Die Architektur setzt sich, wie in Abbildung 3.1 dargestellt, aus vier miteinander verknüpften Blöcken zusammen: der Ein- und Ausgabe (a), der Visualisierung (b), der Interaktion (c) und der Ausgabe (d). Beim Erstellen einer Visualisierung werden zu Beginn die gewünschten Daten aus einer Datei geladen und in ein internes Tabellenformat abgespeichert. Auf dieser sehr performanten, zugrunde liegenden Datenstruktur sind spezifischere Klassen für Tabellen, Graphen und Bäume implementiert. Im Visualisierungs-Block geschieht anschließend das Mapping der Daten auf visuelle Attribute, sowie das Rendering dieser Attribute. Die in Abbildung 3.1 c) verwendete Bezeichnung „Components“ ist etwas ungenau, da diese Komponente Interaktionsmechanismen für Visualisierungen, wie beispielsweise dynamische Abfragen, bereitstellt.

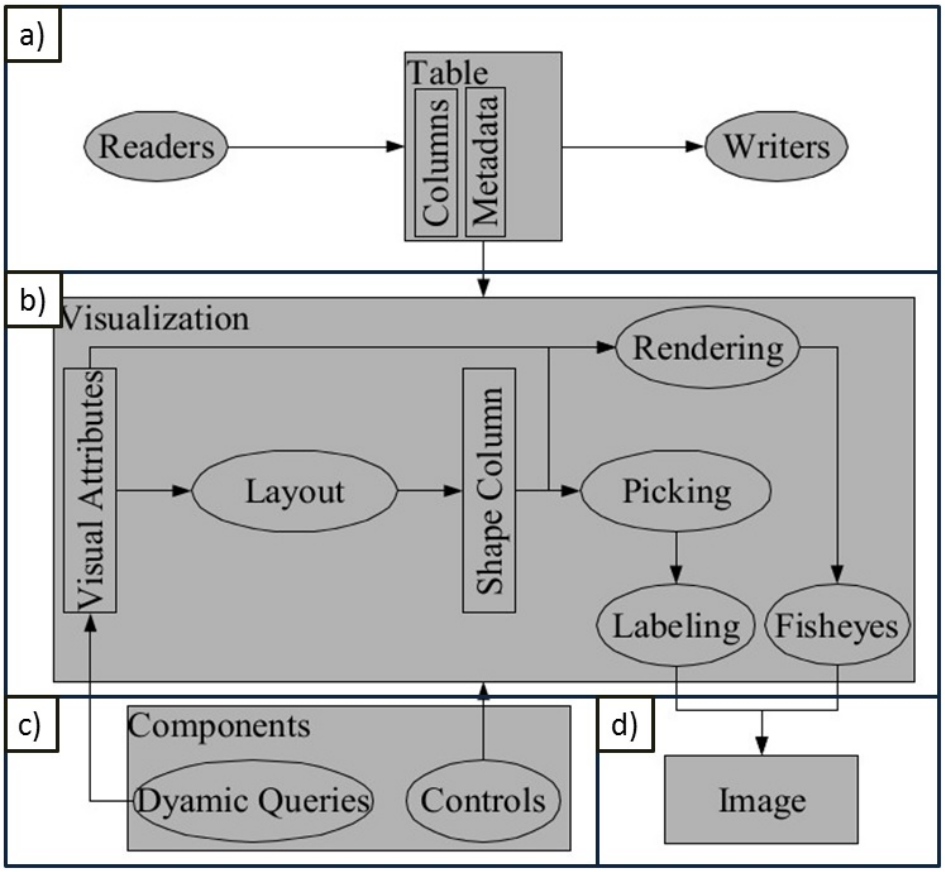

Abbildung 3.1: Vierteiliger Aufbau des InfoVis Toolkit von Fekete et al. Die Daten werden hierbei zunächst geladen (a) und anschließend an die Visualisierungs-Komponente übergeben (b). Diese bildet die Daten auf grafische Primitive ab, rendert die Visualisierung und gibt anschließend die resultierende Grafik (d). Durch verschiedene Mechanismen, kann mit der Visualisierung interagiert werden (c). Grafik aus [9].

3.2 prefuse

Viele der bekannten und weit verbreiteten Frameworks wurden ursprünglich konzipiert, um von den Endnutzern wie eine Art Widget-Toolkit verwendet werden zu können. Dabei stehen vor allem die Kriterien Einfachheit und Bedienbarkeit im Vordergrund. Heer et al. [14] haben sich hingegen dafür entschieden auf der Basis von Java2D ein möglichst flexibles Informationsvisualisierungs-Framework zu entwickeln, welches keine größtenteils vollständigen und starren Konstrukte, sondern kleine, erweiterbare Bausteine enthält. Neben verschiedenen Graph-Layout Algorithmen, geometrischer Verzerrung sowie Animation, werden von prefuse unter anderem auch gängige Interaktionskonzepte wie Zooming und Panning, Drag & Drop usw. unterstützt. Dadurch ist es möglich mit nur verhältnismäßig geringem Programmieraufwand bereits einfache interaktive Visualisierungen zu erstellen.

Angelehnt an die Visualisierungs-Pipeline (siehe Abschnitt 2.2.2) gliedert sich prefuse in eine dreiteilige Struktur. Die Klassen und Interfaces, welche die Implementierung der Pipeline übernehmen, sind umfangreich dokumentiert und zeichnen sich vor allem durch ihre Übersichtlichkeit aus. Um die Erweiterung oder das Austauschen einzelner Komponenten zu ermöglichen, entschieden sich Heer et al. für einen stark modularen Aufbau mit kurzen, in Packages gruppierten Codeblöcken.

Wie in Abbildung 3.2 dargestellt, beschäftigt sich die erste Stufe der prefuse-Pipeline mit der Verarbeitung von abstrakten Daten. Dazu gehört neben verschiedenen Bibliotheken zur Ein- und Ausgabe auch das Ablegen der Daten in den geeigneten Tabellen-, Graph- oder Baum-Datenstrukturen. Der Zwischenschritt des Filtering beschreibt die Überführung dieser Daten in eine visuelle Form. Dabei wird zunächst der Ausschnitt der geladenen Datenpunkte gewählt, welcher später in der Visualisierung sichtbar sein soll. Um das anschließende Rendering zu ermöglichen, werden die Daten mit einigen visuellen Attributen wie Position, Größe und Farbe versehen, woraus sogenannte VisualItems entstehen. Diese VisualItems können daraufhin mit Hilfe verschiedener Renderer auf den Bildschirm gezeichnet werden und sind damit für den Benutzer sichtbar (View). Sämtliche Formen der Interaktion werden vom Anwender durch einen Eingriff in die entsprechende Stufe der Pipeline realisiert. Wie im unteren Bereich von Abbildung 3.2 dargestellt, kann so auf die Bibliotheken zur Ein- und Ausgabe, den durch ActionLists kontrollierten Mapping-Prozess sowie auf das Rendering Einfluss genommen worden.

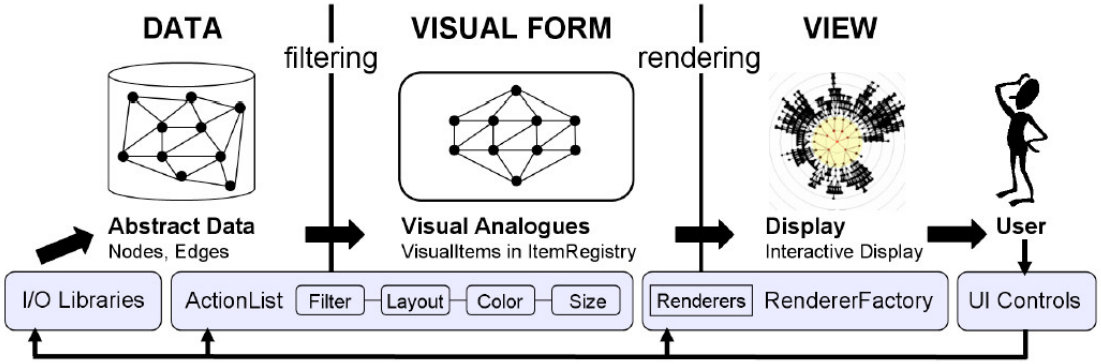

Abbildung 3.2: Die Architektur des prefuse Informationsvisualisierungs-Framework erinnert stark an die Visualiserungs-Pipeline und unterscheidet sich lediglich durch veränderte Bezeichnungen der einzelnen Schritte des Visualisierungsprozesses. So fassen Heer et al. mit dem Begriff Filtering die beiden Schritte Filtering und Mapping zusammen, was dazu führt, dass die hier gezeigte Pipeline kompakter erscheint (Grafik aus [14]).

3.3 Visualization Toolkit

Die bisher in diesem Kapitel aufgeführten Frameworks sind jeweils Beispiele für etablierte Lösungen im Bereich der Informationsvisualisierung. Die verwendeten Renderingmechanismen sowie Datenstrukturen und Algorithmen sind jedoch ausschließlich für Visualisierung im zweidimensionalen Raum entwickelt worden und lassen somit keine Anwendung auf höherdimensionale Daten zu, wie sie üblicherweise in der wissenschaftlichen Visualisierung verwendet werden. Das Visualization Toolkit (VTK) ist ein OpenSource Softwareprodukt von Kitware, welches in C++ und OpenGL implementiert wurde und eine Vielzahl an Komponenten zu den Gebieten 3D-Computergrafik, Bildverarbeitung und Visualisierung enthält. Darunter fallen neben Volume- und Surface-Rendering Mechanismen auch einige der oben erwähnten Techniken für die Informationsvisualisierung. Bindings für die Sprachen Java, Python und Tcl, sowie die Möglichkeit VTK auf den Plattformen Windows, Linux, Mac oder Unix zu verwenden, tragen weiter zur Verbreitung des Toolkits bei.

Die einzelnen Schritte, die nötig sind, um mit Hilfe von VTK aus Rohdaten eine Visualisierung zu erstellen, können durch eine Pipeline beschrieben werden (siehe Abbildung 3.3). Nach dem Laden der entsprechenden Daten durch geeignete Bibliotheken, werden diese durch eine Anordnung verschiedener Filtertypen manipuliert. So ist es beispielsweise nicht nur möglich zwischen Datenpunkten zu interpolieren oder diese zu samplen, sondern auch den Datenfluss auf mehrere separate Flüsse aufzuteilen bzw. diese wieder zusammenzufügen. Im nächsten Schritt folgt die Abbildung von Daten auf

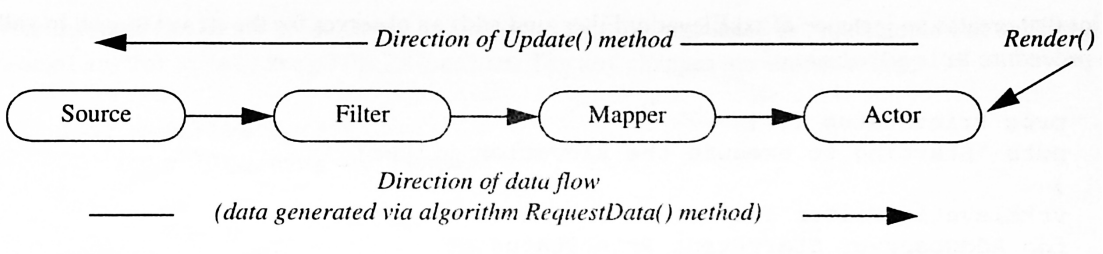

Abbildung 3.3: Generelle Pipeline-Architektur des Visualization Toolkit. Die Daten (Source) durchlaufen einen Filter-Schritt, in dem die Daten transformiert werden. Daraufhin erstellt der Mapper daraus grafische Primitive. Durch verschiedene Actors können schließlich Attribute wie Transparenz oder Farbe gesetzt werden, bevor die Visualisierung schließlich gerendert wird (Grafik aus [16]).

grafische Primitive. Abhängig von der Visualisierungstechnik, die entwickelt werden soll, stellt VTK hierfür verschiedene Mapper, wie z. B. Glyph-Mapper, Molekül-Mapper oder Graph-Mapper, zur Verfügung. Anschließend werden den grafischen Primitiven Actors zugewiesen, die die Darstellung der Objekte bestimmen. An dieser Stelle können jegliche Einstellungen zu Farbe, Transparenz, Level of Detail usw. vorgenommen werden. Durch die Verwendung eines oder mehrerer Renderer, wird die Visualisierung schließlich auf dem Bildschirm sichtbar gemacht. Durch Interaktion hervorgerufene Änderungen der Daten durchlaufen die Pipeline entsprechend in entgegengesetzter Richtung.

3.4 $\mathbb{D}^3$: Data-Driven Documents

Eine komplett andere Herangehensweise an das Thema Visualisierung, wird in der Arbeit von Bostock et al. [5] beschrieben. Das von ihm entwickelte Data-Driven Documents (D3) ist ein, auf den Web-Technologien HTML5 und JavaScript basierendes Framework, welches zum Erstellen dynamischer Visualisierungen in Form von SVG Grafiken (siehe Abschnitt 2.1.6) dient. Dabei wurde großen Wert darauf gelegt, die Datenrepräsentation eng mit dem tatsächlichen Visualisierungs-Vorgang zu verknüpfen. Entwickler binden zuerst die gewünschten Daten mit Hilfe des sogenannten `data` Operators und können anschließend nach einer Datentransformation durch Manipulieren des DOM (siehe Abschnitt 2.1.7) direkten Einfluss auf nicht nur die Darstellung der Visualisierung, sondern auch auf die zugrunde liegenden Daten nehmen.

Vergleichbar mit dem Protovis Framework [4], welches den ebenfalls webbasierten Vorgänger von D3 darstellt, besteht die Implementierung von Bostock et al. aus modularen

JavaScript Komponenten. Diese werden mit Hilfe eines Makefiles zu einer einzigen Datei zusammengefügt und können anschließend zusätzlich verkleinert (minifiziert) werden. Grundsätzlich folgen alle in D3 realisierten Funktionen dem Method Chaining Pattern, bei dem jede Funktion am Ende ihrer Ausführung das aktuell verwendete Objekt zurückgibt und auf diese Weise eine Verkettung von Funktionsaufrufen erlaubt.

Um den Zugriff auf Elemente von HTML Dokumenten und dadurch auch den Zugriff auf das SVG DOM zu vereinfachen, wurden auf Grundlage der W3C Selectors API sogenannte Selektionen eingeführt. So lassen sich durch einen Aufruf von `d3.select` bzw. `d3.selectAll` einzelne Elemente oder Gruppen von Elementen auswählen und anschließend durch entsprechende Operatoren bearbeiten. Basierend auf der W3C DOM API, stellt das Framework neben den Funktionen `append`, `insert` und `remove` zum Hinzufügen und Löschen von DOM Elementen außerdem Operatoren zur Manipulation der Eigenschaften dieser Elemente, wie z. B. der Attribute oder Styles, zur Verfügung. Das Verknüpfen von Daten mit selektierten Knoten im DOM Baum geschieht anschließend mit dem `data` Operator, welcher als einzigen Parameter ein Array bestehend aus Daten von beliebigem Typ enthält. Dieser Vorgang kann automatisch erfolgen oder anhand des Index der entsprechenden Elemente. Außerdem kann alternativ ein geeignetes Mapping vom Entwickler selbst definiert werden. Sind nach der Zuordnung ungebundene Daten oder DOM Elemente übrig, können diese über die speziellen Selektionen `enter` (ungebundene Daten) bzw. `exit` (ungebundene Elemente) ausgewählt werden. Zum Ersetzen der zugrunde liegenden Daten dient zusätzlich die `upgrade` Selektion. Eine Visualisierung wird also letztendlich ausschließlich durch das Erzeugen und Anpassen einzelner SVG Objekte innerhalb eines Dokuments generiert. Zur Verarbeitung von Maus- oder Tastatureingaben greift D3 auf HTML DOM Events zurück, welche sich mit Hilfe des `on` Operators auf jegliche Selektionen anwenden lassen. Beim Erstellen eines solchen Event Handlers wird als Parameter eine Callback-Funktion angegeben, die aufgerufen werden soll, wenn der Benutzer das entsprechende Event auslöst. Des Weiteren können mittels `transition` Operator Animationen mit dynamischen Übergängen definiert werden, indem für Attribute oder Styles ein Start- und Zielwert sowie die gewünschte Art der Interpolation (linear, kubisch etc.) und die Animationsdauer angegeben wird.

Neben den oben genannten Komponenten beinhaltet D3 eine Vielzahl einzelner Module, die den Entwicklern und Designern beim Erstellen von Visualisierungen von Nutzen sein können. So existieren unter anderem verschiedene erweiterte SVG Formen, mehrere Skalen, ein Modul zum Verarbeiten von Kalender- und Zahlenformaten, grafische Layouts, Interaktionskonzepte und ein Parser für Comma-separated-values (CSV).

Seit der Publikation der Arbeit von Bostock et al. im Jahr 2011, wird das Visualisierungs-Framework D3 stetig aktualisiert und weiterentwickelt. Dies hat zur Folge, dass bereits

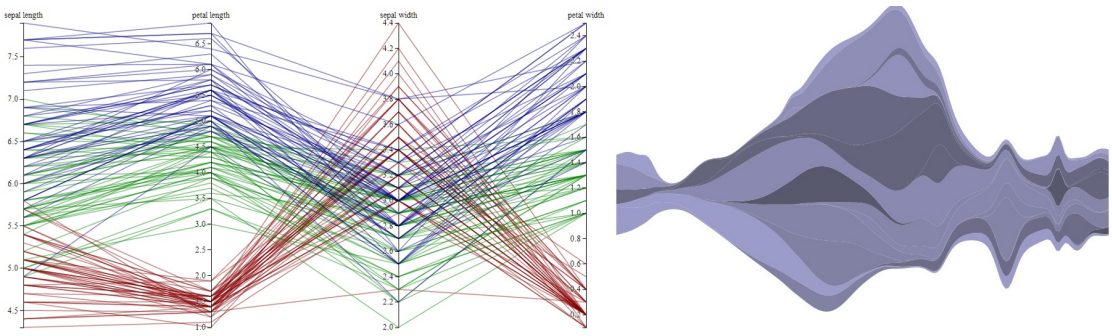

Abbildung 3.4: Beispiele für mit Hilfe des Visualisierungs-Frameworks D3 erstellte Visualisierungen. Auf der linken Seite sind Parallele Koordinaten dargestellt, auf der rechten Seite befindet sich die Abbildung eines Streamgraphs.

eine Vielzahl von Visualisierungstechniken existieren, die mit Hilfe von D3 umgesetzt wurden. Beispielsweise zeigt Abbildung 3.4 auf der linken Seite eine für multivariate Daten geeignete Repräsentation durch Parallele Koordinaten[1] (links) sowie einen Streamgraph[2] (rechts). Michael Bostock arbeitet außerdem unter anderem mit der Tageszeitung „The New York Times“ zusammen, die in ihrer Online-Ausgabe D3 zur Darstellung von verschiedenen Datensätzen verwendet.

3.5 Gemeinsamkeiten und Unterschiede der Frameworks

In diesem Kapitel wurden vier Visualisierungs-Frameworks aus den Bereichen der Informationsvisualisierung und der wissenschaftlichen Visualisierung beschrieben. Obwohl die jeweiligen Implementierungen sich stark voneinander unterscheiden, besitzen alle diese Visualisierungs-Frameworks, mit Ausnahme von D3, als grundlegende Struktur einen an die Visualisierungs-Pipeline (siehe Abschnitt 2.2.2) angelehnten Aufbau. Oftmals wurden hierbei lediglich einige Bezeichnungen oder Details der Verarbeitungsschritte anders definiert. Das webbasierte Framework D3 folgt zwar keiner direkten Pipeline-Architektur, der Visualisierungs-Prozess erfolgt jedoch ebenfalls auf ähnliche Weise: Nach dem Laden von einer Datenquelle, werden die Daten hierbei zunächst auf grafische Primitive abgebildet und schließlich gerendert.

[1] Quellcode: `http://mbostock.github.io/d3/talk/20111116/iris-parallel.html`
[2] Quellcode: `http://bl.ocks.org/mbostock/4060954`

4 Aufgabenstellung und Lösungsansatz

Im Rahmen dieser Studienarbeit soll ein Konzept zur automatischen semantischen Annotation von grafischen Elementen in Visualisierungen entwickelt werden. Das folgende Kapitel beschreibt die einzelnen Teilbereiche der Aufgabenstellung sowie den Lösungsansatz, welcher anschließend in Kapitel 5 genauer ausgeführt wird.

4.1 Aufgabenstellung

Bei der Erforschung von kognitiven Prozessen im Zusammenhang mit Visualisierungen sind vor allem Eye-Tracking Studien von großer Bedeutung. Dabei werden Blickbewegungen von mehreren Probanden aufgezeichnet und daraufhin durch Visualisierungstechniken, wie beispielsweise Heatmaps oder Gaze Plots, sichtbar gemacht. Einer der zeitaufwendigsten Aspekte bei der Durchführung solcher Studien ist, neben dem Entwurf und korrekten Design der Studie, vor allem die Auswertung der gemessenen Blickdaten, welche in der Regel mühsam von Hand erfolgen muss. Dazu müssen manuell wichtige Regionen (Areas of Interest, AOI) in der Visualisierung definiert werden. Um diesen Arbeitsschritt zu vereinfachen und in Zukunft zu automatisieren, besteht das Ziel dieser Studienarbeit darin, ein Konzept zu entwickeln, welches eine semantische Annotation im Rendering-Vorgang von Visualisierungen ermöglicht. Hierfür wird zu Beginn eine Recherche zu bereits veröffentlichten Ansätzen und Techniken inklusive einer Diskussion über deren Gemeinsamkeiten und Unterschiede durchgeführt (siehe Kapitel 3). Diese stellt, zusammen mit Kapitel 2, die wichtigsten Voraussetzungen für das folgende Konzept dar. Das Konzept bildet den Kern der Arbeit und beschreibt ausführlich, welche einzelnen Komponenten und Schritte notwendig sind, um semantische Informationen mit bereits existierenden Informationsvisualisierungs-Anwendungen sowie mit neuen Implementierungen zu verbinden. Des Weiteren wird an dieser Stelle ein kurzer Ausblick darüber gegeben, welche Problematiken sich bei der Abbildung des Konzepts auf den Bereich der wissenschaftliche Visualisierung ergeben und wie sich einige von diesen Problematiken lösen lassen. Teil der Aufgabenstellung ist außerdem die prototypische Implementierung des Ansatzes in einem geeigneten Visualisierungs-Framework anzufertigen und eine Beschreibung des Implementierungs-Vorgangs zu erstellen.

Abbildung 4.1: Die erweiterte Pipeline besitzt jeweils einen neuen Eingabe- und Ausgabeparamter: die Visualisierungs-Ontologie bzw. die Annotationsdaten.

4.2 Lösungsansatz

Die Aufgabenstellung lässt sich grob in zwei Bereiche aufteilen. Zum einen muss im Vorfeld eine semantische Wissensbasis entworfen werden, welche die zur Annotation benötigten Elemente und die dazugehörigen Relationen enthält. Zum anderen muss untersucht werden, wie sich diese Wissensbasis in den Rendering-Prozess von Visualisierungen integrieren lässt.
Die ausführliche Recherche über verbreitete Techniken des Semantic Web (siehe Abschnitt 2.1) hat ergeben, dass sich zur Wissensrepräsentation die vom W3C spezifizierte Web Ontology Language (OWL) eignet. OWL bietet durch verschiedene Sprachebenen und Reasoning-Mechanismen einen ausreichenden Umfang für die Domäne der Visualisierung. Welche Version von OWL dafür genau erforderlich sein wird, hängt von den benötigten Konstrukten im Modellierungsvorgang ab und muss im späteren Verlauf der Arbeit entschieden werden. Der Aufbau der Wissensbasis soll, angelehnt an das von Engelhardt [8] und Raschke et al. [22] beschriebene Ontologie-Modell, in drei miteinander verbundenen Ebenen erfolgen: einer Ontologie von grafischen Primitiven, einem Visualisierungsschema und einem Visualisierungsmodell.
Die Entwicklung des Konzepts stützt sich maßgeblich auf die zugrunde liegende Architektur von Visualisierungs-Frameworks, von denen einzelne in Kapitel 3 behandelt wurden. Als durchgehend auftretendes Paradigma, ausgenommen D3, lässt sich dabei die Visualisierungs-Pipeline erkennen. Daher beinhaltet das Lösungskonzept deren Erweiterung um zwei Schritte, damit schließlich neben den Eingabedaten auch die Visualisierungs-Ontologie in die Pipeline einfließt und die Visualisierung als Ausgabe zusätzlich zur gerenderten Grafik entsprechende Annotationsdaten generiert (siehe Abbildung 4.1). Diese Annotationsdaten bestehen aus in OWL formulierten Instanzen bzw. Relationen und beschreiben wie die verschiedenen Elemente in einer konkreten Grafik angeordnet sind sowie welche Beziehungen zwischen diesen Elementen herrschen.
Für die Implementierung des Prototyps wurde aus Kapitel 3 das Framework D3 ausgewählt, da D3 auf Web Technologien aufbaut und daher ohne größeren Aufwand mit Technologien des Semantic Web zu erweitern ist. Auch scheint das Ausgabeformat SVG von D3, welches nicht nur statische Bilder, sondern auch Animationen erlaubt, sich für diesen Zweck eignen. Hierbei können grafische Elemente durch die von D3 bereitgestellten Mechanismen auf direkte Weise erstellt und anschließend manipuliert werden.

5 Lösungskonzept

Dieses Kapitel stellt das erarbeitete Lösungskonzept vor, welches sich unter drei verschiedenen Blickwinkeln mit der Problematik der automatischen semantischen Annotation in Visualisierungen auseinandersetzt. Zuerst wird dabei das Konzept anhand der Visualisierungs-Pipeline erklärt, worauf die Anwendung des Konzepts auf das Visualisierungs-Framework D3 folgt. Schließlich wird gezeigt, inwiefern die Annotation von wissenschaftlichen Visualisierungen auf eine solche Art und Weise denkbar ist. Der einleitende Abschnitt über den grundlegenden Aufbau der Ontologie ist hierbei für alle der drei Ansätze gültig und wird im Konzept jeweils genauer ausgeführt.

5.1 Erstellung einer Visualisierungs-Ontologie

Dieser Abschnitt behandelt den schematischen Aufbau einer Ontologie für Visualisierungen. Hierfür erfolgt zunächst eine kurze Analyse der Visualisierungs-Domäne, um herauszuarbeiten welche semantischen Objekte und Relationen für einen solchen Anwendungsbereich erforderlich sind.

5.1.1 Vom WO-Raum in den WAS-Raum

Die nach der Aufzeichnung folgende Auswertung von Eye-Tracking Studien lässt sich grundsätzlich, wie von Raschke et al. [21] beschrieben, in zwei unterschiedliche Räume aufteilen. Eine Analyse im WO-Raum gibt dabei an, welche Positionen vom Proband fixiert wurden und welche Augenbewegungen zwischen diesen Fixationen erfolgt sind. Die Ausgabe von Eye-Tracking Systemen besteht hierbei in der Regel aus absoluten Pixelwerten, die in einem kartesischen Koordinatensystem aufgetragen sind. Entscheidend für die Auswertung solcher Studien ist jedoch meist nicht welche Pixel ein Proband betrachtet hat, sondern welche wahrgenommenen Objekte durch die entsprechenden Pixel repräsentiert werden. Dies wird mittels einer geeigneten Abbildung vom WO-Raum in den WAS-Raum realisiert. Abbildung 5.1 zeigt eine Gegenüberstellung der beiden Räume. Auf der linken Seite der Grafik befindet sich der WO-Raum, auf der rechten Seite der WAS-Raum. Die blauen Kreise symbolisieren dabei einzelne mögliche Fixationen eines Probanden. An den Kreisen sind beispielhaft die entsprechenden Informationen

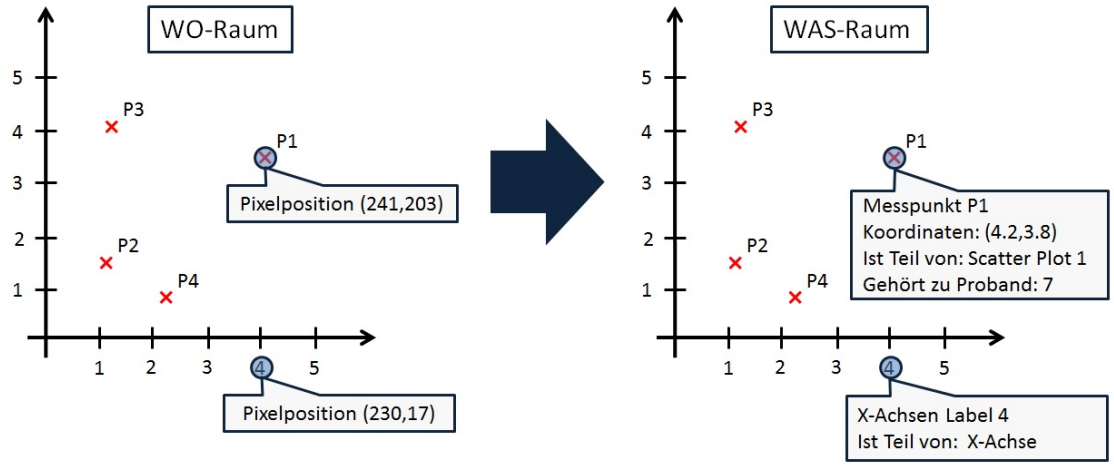

Abbildung 5.1: Darstellung der Abbildung vom WO-Raum (links) in den WAS-Raum (rechts). Im WO-Raum kann, im Gegensatz zum WAS-Raum, nicht automatisch anhand von Fixationspunkten (symbolisiert durch blaue Kreise) ermittelt werden, welche Elemente der Visualisierung vom Proband betrachtet wurden.

eingetragen, die im WO- bzw. WAS-Raum anhand eines Fixationspunkts bestimmt werden können. Die Bedeutung der Objekte sowie die Beziehungen der einzelnen Objekte untereinander lassen sich im WAS-Raum z. B. mit dem Konzept einer Ontologie (siehe Abschnitt 2.1.1) verwirklichen.

5.1.2 Domänenanalyse und Anforderungen an die Ontologie

Es existieren bereits verschiedene Ansätze, welche versuchen den Themenbereich der Visualisierung durch semantische Konzepte wie Ontologien zu beschreiben. So haben Voigt und Polowinski [30] an der Technischen Universität Dresden auf Grundlage von vorangegangenen Publikationen eine Ontologie formuliert, welche die Vorteile der untersuchten Arbeiten vereinen soll und dadurch für eine Vielzahl von Anwendungsgebieten verwendet werden kann. Dabei unterscheiden sie die drei Bereiche Daten (Data), Grafiken (Graphic) und Interaktion (Human Activity) und führen diese anschließend weiter aus. Ähnlich wie Voigt und Polowinski [30] befassen sich Brodlie et al. [6], Wehrend et al. [31] und weitere Autoren meist damit, den gesamten Visualisierungsprozess in einem oder mehreren Konzepten zusammenzufassen. Der Schwerpunkt liegt dabei auf der korrekten und möglichst vollständigen Klassifizierung der auftretenden Begriffe.
In dieser Studienarbeit soll hingegen eine Ontologie entworfen werden, die durch verschiedene Relationen beschreibt, wie einzelne grafische Primitive zu Visualisierungen

zusammengefügt werden können. Bei der Betrachtung eines Visualisierungselements in der annotierten Visualisierung durch einen Proband können dann Rückschlüsse auf die Bedeutung des Elements gezogen werden. Dafür notwendig ist im Folgenden, neben dem Entwurf des Aufbaus der Ontologie, die nähere Betrachtung der verfügbaren Datentypen sowie der zu formalisierenden Visualisierungstechniken.

5.1.3 Aufbau der Ontologie-Ebenen

Wie in den Grundlagen beschrieben, werden Ontologien unter Verwendung der Beschreibungssprachen RDF oder OWL erstellt und setzen sich aus Klassen, Instanzen und Relationen zusammen (siehe Abschnitt 2.1). Um die Erweiterbarkeit und die Übersichtlichkeit einer Ontologie zu gewährleisten, ist es üblich die einzelnen Teilgebiete der Wissensdomäne in modulare Blöcke aufzuteilen. In seiner Diplomarbeit über die Modellierung kognitiver Prozesse in der Visualisierung schlägt Engelhardt eine Zerlegung in drei Ebenen vor: Ontologie, Visualisierungsschema und Visualisierungsmodell [8]. Wie Abbildung 5.2 zeigt, beinhaltet die erste Ebene dabei sämtliche Visualisierungs-Elemente, die in einer Visualisierung auftreten können sowie immer geltende Relationen zwischen den Elementen. Das Visualisierungsschema benutzt diese Elemente und bildet daraus verschiedene Klassen von Visualisierungstechniken. Die konkrete Instanziierung der Visualisierungstechniken werden schließlich im Visualisierungsmodell festgehalten. Dieser dreischichtige Aufbau soll grundsätzlich beibehalten werden, wobei die einzelnen Ebenen entsprechend des Anwendungsgebiets angepasst werden. Dies betrifft insbesondere diejenigen Unterpunkte von Engelhardt, die durch den Zusammenhang mit kognitiven Prozessen motiviert waren und für diese Studienarbeit daher nicht relevant sind.

Die erste Ebene der Visualisierungs-Ontologie, bei Engelhardt als Ontologie bezeichnet, soll in dieser Arbeit keine Visualisierungs-Elemente sondern lediglich grafische Primitive enthalten. Die Visualisierungs-Elemente, sowie die daraus zusammengesetzten Visualisierungstechniken befinden sich hingegen im Visualisierungsschema. Das Visualisierungsmodell entspricht der entsprechenden Definition von Engelhardt und bleibt weitgehend unverändert. Eine detaillierte Beschreibung der einzelnen Komponenten findet sich in Abschnitt 5.2.2.

5.2 Semantische Annotation in der Visualisierungs-Pipeline

An dieser Stelle soll ein Konzept beschrieben werden, welches sich mit der Integration von automatischer semantischer Annotation in Informationsvisualisierungs-Anwendungen beschäftigt. Dabei wird davon ausgegangen, dass deren zugrunde liegende Architektur auf der Visualisierungs-Pipeline beruht.

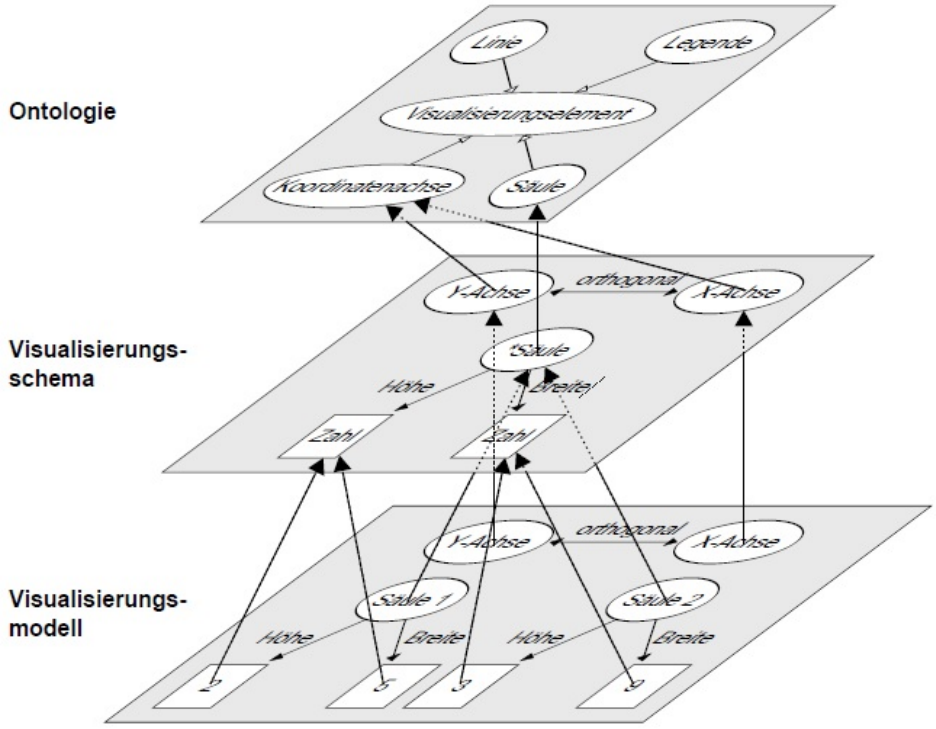

Abbildung 5.2: Aufbau der aus drei Ebenen bestehenden Visualisierungs-Ontologie nach Engelhardt [8]. Die oberste Ebene Ontologie beinhaltet hierbei sämtliche Visualisierungs-Elemente und die jeweiligen Relationen. Das Visualisierungsschema besteht aus einzelnen Klassen von Visualisierungstechniken, die wiederum aus Visualsierungs-Elementen zusammengesetzt sind. Die Instanzen dieser Klassen befinden sich im Visualisierungsmodell.

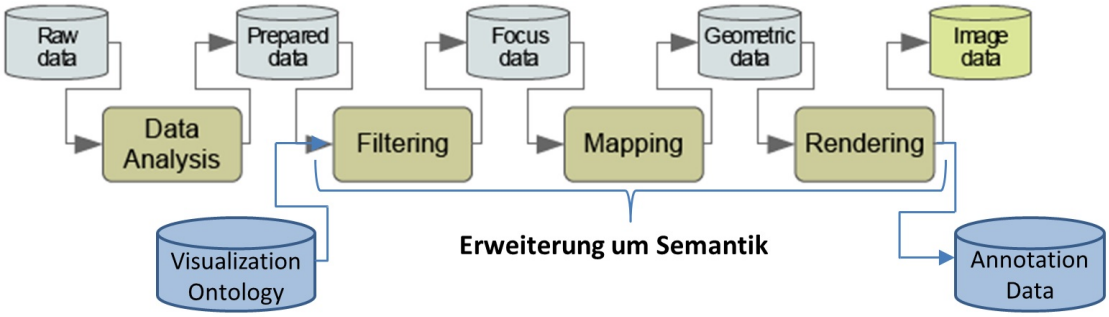

Abbildung 5.3: Die Pipeline-Architektur des Lösungskonzepts. Der Visualisierungs-Pipeline wurden hierbei zwei Komponenten hinzugefügt: eine Visualisierungs-Ontologie (a) sowie durch den semantischen Rendering-Prozess generierte Annotationsdaten (b). Die ursprüngliche Grafik stammt aus [24] und wurde entsprechend modifiziert.

5.2.1 Die semantische Visualisierungs-Pipeline

Im Bereich der Visualisierung gilt die von dos Santos und Brodlie [24] entwickelte Visualisierungs-Pipeline als ein Grundpfeiler, auf den die meisten gebräuchlichen Visualisierungs-Frameworks und oft auch einzelne Anwendungen (siehe Kapitel 3) aufbauen. Aus diesem Grund verwendet das im Rahmen der Studienarbeit entstandene Lösungskonzept diese Pipeline-Struktur als Ausgangspunkt für eine semantische Visualisierungs-Pipeline. Durch diese wird es möglich, automatische semantische Annotation nicht nur in zukünftigen Anwendungen umzusetzen, sondern auch in bereits existierende Werkzeuge einzubetten. Die einzelnen Komponenten, die in der Pipeline hinzugefügt oder abgeändert werden müssen, sind in Abbildung 5.3 eingezeichnet und werden in den nachfolgenden Abschnitten erläutert.

Visualisierungs-Ontologie: Entsprechend der Visualisierungs-Domäne, wird eine Ontologie erstellt, welche die benötigten grafischen Primitive, Klassen von Visualisierungstechniken und die dazugehörigen Instanzen enthält. Diese wird erst vor dem Filtering in die Pipeline eingefügt, das heißt sie wird an dieser Stelle von der Anwendung aus einer Datei oder von einem Webserver geladen. Anschließend kann sie geparst und in einer geeigneten Datenstruktur abgelegt werden. Eine detaillierte Beschreibung des Aufbaus der Visualisierungs-Ontologie findet sich in Abschnitt 5.2.2.

Filtering: Am Ende des Filtering-Schritts kann eine erste Form der Annotation erfolgen. Zu diesem Zeitpunkt ist bereits klar, welche der Datenpunkte im späteren Verlauf zur Visualisierung beitragen werden, auch wenn ihre visuelle Repräsentation noch nicht feststeht. Wichtige Merkmale, die zu einem bestimmten Datenpunkt gehören können

durch eine einfache Referenzierung des Elements in der Visualisierungs-Ontologie gekennzeichnet werden. Beim Durchführen einer Studie könnten solche Merkmale beispielsweise Nummer und Alter eines Probanden oder ähnliche Informationen sein. Für objektorientierte Programmiersprachen bietet es sich hierfür an, falls nicht bereits vorhanden, eine Datenpunkt Klasse zu erstellen, welche neben dem Wert des Datenpunktes zusätzlich einen `semantics` Vektor enthält, in dem sämtliche semantische Informationen über das Objekt aufgelistet werden.

Mapping: Das Mapping beschreibt den Vorgang in der Pipeline, in dem den ausgewählten Datenpunkten ihre visuelle Repräsentation zugewiesen wird. An dieser Stelle steht fest, welcher Datenpunkt welches Element oder welche Elemente der Visualisierung darstellen soll und damit auch auf welche Position oder Positionen er abgebildet wird. In der zuvor angelegten Datenpunkt Klasse können daher zusätzliche Informationen dem `semantics` Vektor hinzugefügt werden: die Position(en) des Punktes in der Visualisierung bzw. alle Pixel, die ein Datenpunkt in der Visualisierung belegt (siehe Unterpunkt Annotationsdaten), das zugeordnete Visualisierungs-Element (beispielsweise Linie, Kreis oder Achse), Orientierung des Elements, benachbarte Elemente, Relationen zu anderen Datenpunkten usw. Dies geschieht in der Regel durch die Instanziierung der Klassen, die zuvor in Ebene 2 der Visualisierungs-Ontologie definiert wurden sowie durch die Benutzung der dort spezifizierten Relationen.

Rendering: In den vorherigen beiden Schritten ist die vollständige Annotation der Visualisierung erfolgt. Der Rendering-Prozess hat nun die Aufgabe neben der gerenderten Visualisierung auch die angegeben semantischen Informationen auszugeben. Dies kann, je nach Ausgabeformat, auf zwei unterschiedlichen Wegen geschehen. Unter der Verwendung eines Rastergrafik Formats muss die im `semantics`-Vektor bereitgestellte Annotation in einem geeigneten Format (siehe Abschnitt 2.1.2) in eine separate Datei geschrieben werden. Alternativ lässt sich für die Visualisierung ein auf XML basierendes Grafikformat, wie z. B. SVG, wählen, bei dem die Annotation direkt in der Grafik als XML Attribute abgelegt werden kann. Eine genauere Beschreibung, wie sich dieser Vorgang umsetzen lässt, findet sich in Abschnitt 5.3.1.

Annotationsdaten: Die Annotationsdaten bestehen aus sämtlichen, zur Visualisierung gehörenden Ontologie-Instanzen. Die gesamten Annotationsdaten aller möglichen Instanzen werden im Zusammenhang der Visualisierungs-Ontologie auch als Visualisierungsmodell bezeichnet. Befinden sich die Annotationsdaten in einer separaten Datei, ist besonders wichtig, dass die Annotation unter anderem für jedes Ontologie-Element diejenigen Pixel enthält, die es in der Ausgabegrafik umfasst. Nur so können später die Daten den visuellen Elementen in der resultierenden Visualisierungs-Grafik zugeordnet werden. Bei der Zuordnung muss beachtet werden, dass sich grafische Elemente in Visualisierungen häufig überdecken können. Das hat zur Folge, dass ein Pixel in einer Rastergrafik beliebig vielen Ontologie-Elementen entsprechen

kann. Diese Tatsache spielt besonders dann eine wichtige Rolle, wenn beispielsweise in Eye-Tracking Studien zu einem bestimmten Blickpunkt eines Probanden die entsprechende semantische Bedeutung gesucht wird.

5.2.2 Ontologie-Ebenen im Detail

Wie genau die einzelnen Ebenen der Visualisierungs-Ontologie aussehen müssen, ergibt sich durch eine Analyse der zugrunde liegenden Programmiersprache bzw. der bestehenden Anwendung und der Visualisierungstechniken, die mit semantischer Information versehen werden sollen.

Ebene 1 - Grafische Primitive: Alle Arten von zweidimensionalen Visualisierungen setzen sich aus grafischen Primitiven zusammen. Welche Primitive dem Entwickler genau zur Verfügung stehen, ist jedoch davon abhängig welche Programmierschnittstelle (auch Application Programming Interface, API) für die Ausgabegrafiken verwendet wird. Beispiele hierfür sind Java 2D sowie Direct2D. Die einzelnen Elemente, die in der entsprechenden API beschrieben sind, werden lediglich in einen hierarchischen Zusammenhang gebracht und bilden somit die erste Ebene der Visualisierungs-Ontologie (siehe Abbildung 5.4, linke Komponente). In einigen Fällen kann es zudem nützlich sein, neben hierarchischen Beziehungen auch weitere Relationen für die grafischen Primitiven bereits in dieser Ebene einzufügen, um z. B. Metainformationen zu speichern.
Es ist außerdem hilfreich, diejenigen Datentypen aufzuführen, die als Rohdaten in der Visualisierungs-Pipeline in Erscheinung treten. So kann im späteren Verlauf bei der Annotation festgelegt werden, welcher Datenpunkt auf welches grafische Element abgebildet wurde.

Ebene 2 - Visualisierungsschema: Das Visualisierungsschema legt fest, wie Visualisierungen aus grafischen Primitiven aufgebaut werden können. Dazu wird für jede Visualisierungstechnik eine Klasse erstellt und dieser bestimmte Relationen zugewiesen, die bestimmen, welche Visualisierungs-Elemente in der Visualisierung vorhanden sind. Visualisierungs-Elemente sind hierbei „Einzelteile“ einer Visualisierung, die aus grafischen Primitiven zusammengefügt oder direkt aus diesen abgeleitet werden. Weitere Informationen wie die Ausrichtung oder die relative Position der Elemente zu anderen Objekten wird anschließend durch zusätzliche Relationen definiert (siehe Abbildung 5.4, mittlere Komponente).

Ebene 3 - Visualisierungsmodell: Durch die beiden bisher beschriebenen Ebenen lassen sich die Konzepte von Visualisierungen bereits formal festlegen. Konkrete Ausprägungen, welche durch einen Datensatz generiert werden, sind jedoch noch nicht Bestandteil der Visualisierungs-Ontologie. Diese werden im Visualisierungsmodell instanziiert und gegebenenfalls mit weiteren Relationen versehen (siehe Abbildung

Abbildung 5.4: Schematischer Aufbau der Visualisierungs-Ontologie. Links befinden sich die grafischen Primitive, die über Subklassen-Beziehungen mit dem Visualisiersschema (mitte) verbunden sind. Aus dem Visualisierungsschema ensteht durch Instanzbildung das Visualisierungsmodell (rechts).

5.4, rechte Komponente). So kann beispielsweise zusätzlich in der Ontologie verankert werden, welche Farbe, Linienstärke usw. die einzelnen visuelle Elemente besitzen.

5.3 Semantische Annotation mit dem $\mathbb{D}^3$ Visualisierungs-Framework

Das webbasierte Visualisierungs-Framework D3 liefert, durch die Unterstützung von Web Technologien sowie von einem geeigneten Ausgabeformat, bereits die geeignete Grundfunktionalität, welche eine Erweiterung um einen automatischen Annotationsprozess erheblich erleichtert. Dieser Abschnitt erläutert, welche Schritte konzeptionell nötig sind, um grafische Elemente in Visualisierungen mit Hilfe von D3 zu annotieren und wie genau eine Visualisierungs-Ontologie für D3 aussehen kann.

5.3.1 Integration in das bestehende Framework

Im Gegensatz zu den anderen, in Kapitel 3 vorgestellten Visualisierungs-Frameworks, baut D3 nicht auf einer Pipeline-Architektur auf. Stattdessen steht die direkte Abbildung von Daten auf visuelle Elemente im Vordergrund. Dies hat zur Folge, dass das in Abschnitt 5.2 ausgeführte Konzept nur zum Teil auf dieses Framework angewendet werden kann. Die meisten der dort beschriebenen Schritte sind für eine Integration einer semantischen Annotation in D3 nicht erforderlich. Es genügt hierbei, lediglich an zwei unterschiedlichen Stellen im Visualisierungs-Prozess einige Änderungen vorzunehmen: zu Beginn des Dokuments, welches die Visualisierung enthält, sowie später bei der Abbildung von Datenpunkten zu visuellen Elementen (Mapping).

Einbinden der Visualisierungs-Ontologie: Die Visualisierungs-Ontologie liegt in Form von RDF oder OWL Dateien vor und kann mit Hilfe einer geeigneten Bibliothek geöffnet und verarbeitet werden. Um anschließend innerhalb von D3 Verweise auf die jeweiligen Elemente zu ermöglichen, muss der Ladevorgang der Ontologie bereits vollständig abgeschlossen sein bevor das Visualisierungs-Framework in das Dokument eingebunden wird.

Annotation der Visualisierung: Der Annotations-Prozess selbst lässt sich in zwei Schritte aufteilen. Zum einen können grafische Primitive, die durch Ebene 1 der Visualisierungs-Ontologie definiert sind, direkt innerhalb von D3 automatisch annotiert werden. Dafür muss zunächst festgestellt werden, an welcher Stelle das Framework neue DOM Elemente in den DOM Baum des HTML Dokuments einfügt. Dort kann nun für jedes neu erstellte Objekt überprüft werden, ob es sich dabei um ein SVG Primitiv aus der ersten Ebene der Ontologie handelt oder nicht. Wenn dies der

Fall sein sollte, wird ein neues Attribut im jeweiligen SVG Tag angelegt, dessen Inhalt auf das zugehörige Ontologie-Element verweist. Alle Ontologie-Elemente aus dem Visualisierungsschema (Ebene 2) müssen zum anderen hingegen in der konkreten Ausprägung der Visualisierung instanziiert werden und bilden dadurch die dritte Ebene, das Visualisierungsmodell. Dies geschieht im Dokument nach dem Laden der Datenpunkte aus der Datenquelle. Nur zu jenem Zeitpunkt lässt sich eindeutig feststellen, aus welchem Datenpunkt welche visuellen Elemente generiert werden. Zur Annotation werden hierbei, wie bereits bei den grafischen Primitiven, entsprechende Attribute mit Referenzen auf die Ontologie-Elemente hinzugefügt. Eine separate Ausgabe der Annotationsdaten in eine Datei (siehe Abbildung 5.3) ist nicht nötig, da diese bereits in den Attributen der SVG Datei enthalten sind und daraus bei Bedarf extrahiert werden können.

5.3.2 Anpassung der Ontologie-Ebenen

Wie bereits in Abschnitt 3.4 erwähnt, arbeitet D3 mit dem 2D Vektorgrafik Format SVG, welches auf der Sprache XML aufbaut. Da sich auch das Ontologie-Modell von Engelhardt auf 2D-Diagramme beschränkt, genügen nur geringe Anpassungen, um dieses auf D3 anwenden zu können.

Ebene 1 - Taxonomie grafischer Primitive und Datentypen: Die erste Ebene der entworfenen Visualisierungs-Ontologie stellt streng genommen keine Ontologie dar, sondern lediglich eine Taxonomie, d. h. eine hierarchische Anordnung von Termen. Im ersten Teil der Taxonomie werden, entsprechend der SVG Spezifikation [10], alle beschriebenen grafischen Primitive aufgelistet und außerdem durch Subklassen-Beziehungen wie in Abbildung 5.5 strukturiert. Zusätzliche Relationen für beispielsweise die Position oder eindeutige Bezeichner der Objekte sind nicht notwendig, da der Zugriff auf solche Werte bereits vollständig durch Attribute im DOM bereitgestellt wird.
Die Taxonomie definiert zusätzlich diejenigen Datentypen, die nach dem Laden eines Datensatzes innerhalb von D3 verarbeitet werden können. Hierbei beschränkt sich das Framework auf die Datentypen String, Number, Boolean, Array und Object aus JavaScript.

Ebene 2 - Visualisierungsschema: Für das Visualisierungsschema werden nun, wie in Abschnitt 5.2.2 aufgeführt, die benötigten Visualisierungstechniken untersucht und anschließend aus den einzelnen Elementen aus Ebene 1 mit Hilfe von Relationen in die Visualisierungs-Ontologie eingefügt. Ein Schritt-für-Schritt Beispiel für das Hinzufügen einer neuen Visualisierungstechnik in das Visualisierungsschema findet sich im Implementierungskapitel in Abschnitt 6.2.

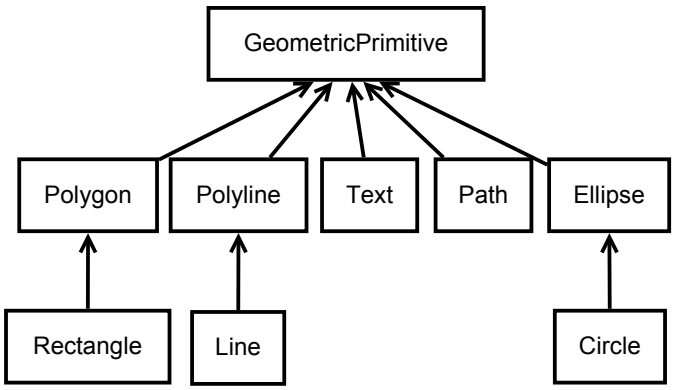

Abbildung 5.5: Hierarchische Anordnung der grafischen Primitive in Anlehnung an das SVG Format. Die Klasse `GeometricPrimitive` dient hierbei als Superklasse für alle vorhandenen Primitive.

Ebene 3 - Visualisierungsmodell: Aus den Klassen des Visualisierungsschemas lassen sich anschließend konkrete Instanzen bilden. Wie in Abschnitt 5.3.1 erläutert, befindet sich das dadurch entstehende Visualisierungsmodell hierbei nicht in einer separaten Datei, sondern in Form von mehreren Attributen direkt in den jeweiligen Tags der visuellen Elemente.

5.3.3 Ausführen von Eye-Tracking auf annotierten Visualisierungen

Nachdem in den vorigen Abschnitten erläutert wurde wie sich Visualisierungen mit D3 annotieren lassen, können Eye-Tracking Experimente auf Basis der gerenderten und annotierten Visualisierungen durchgeführt werden. Hierbei ist es nützlich für die Auswertung weiterhin einen Webbrowser in Verbindung mit Skriptsprachen zu verwenden. Jeder vom Eye-Tracker gemessene Fixationspunkt wird von der Skriptsprache JavaScript verarbeitet, die anschließend alle grafischen Elemente ausgibt, welche sich an einem dieser Position befinden.
Aufgrund von möglicher Transparenz, Überdeckung oder Überlappung von Objekten, ist es nicht möglich automatisch festzustellen, welches der in der ausgegebenen Menge vorhandenen Elemente vom Proband tatsächlich betrachtet wurde. Im Rahmen von Eye-Tracking Studien werden Probanden jedoch außerdem dazu angehalten bestimmte Aufgabenstellungen zu bearbeiten. Anhand von dieser Information lässt sich die Menge der möglichen betrachteten Elementen einschränken. Soll für einen gerichteten Graph beispielsweise die Anzahl von Kantenkreuzungen festgestellt werden (siehe Abbildung 5.6, links), so ist es unwahrscheinlich, dass die Knoten im Graph häufig fixiert werden. Ist das Ziel, einen Pfad von einem Knoten zu einem anderen Knoten zu finden (siehe Abbildung 5.6, rechts), so ist diese Wahrscheinlichkeit wesentlich höher. Visualisierungen

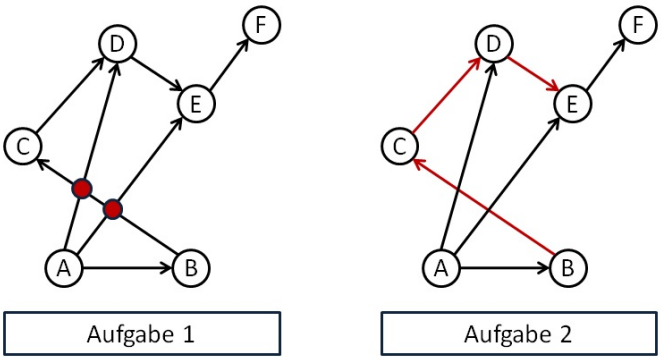

Abbildung 5.6: Beispielhafte Darstellung zweier Aufgabentypen für Eye-Tracking Studien anhand eines gerichteten Graphs. In Aufgabe 1 (links) soll die Anzahl der Kantenkreuzungen im Graph bestimmt werden. Zur Lösung von Aufgabe 2 müssen die Probanden hingegen einen Pfad von Knoten B zu Knoten E finden.

beinhalten zudem oft sehr kleine oder semantisch irrelevante Objekte, die von vornherein ausgeschlossen werden können. Die Kombination dieser Annahmen führt zu einer stark eingeschränkten Menge an möglichen Elementen, die zusammen mit ihren entsprechenden Attributen, wie beispielsweise Form, Farbe und Transparenz, ausgegeben werden können. Unter Verwendung dieser Attribute kann schließlich von Hand das betrachtete Objekt im WAS-Raum gefunden werden.

5.4 Semantische Annotation für wissenschaftliche Visualisierungen

Mit den in den Grundlagen vorgestellten Techniken des Semantischen Web (siehe Abschnitt 2.1) ist es möglich, die unterschiedlichsten Domänen mit Hilfe von Ontologien formal zu beschreiben. So gibt es bereits erste Ansätze, die sich mit semantischer Annotation im dreidimensionalem Raum beschäftigen (siehe Abschnitt 5.4.1). Diese beschränken sich allerdings lediglich auf eine nachträgliche Annotation von bereits existierenden 3D-Modellen. Im darauffolgenden Abschnitt werden daher einige Problemstellungen bei dreidimensionalen Visualisierungen im Vergleich zu zweidimensionalen Visualisierungen erläutert. Schließlich wird diskutiert, inwiefern sich das erarbeitete Lösungskonzept auf wissenschaftliche Visualisierungen anwenden lässt.

5.4.1 Bestehende Techniken zur Annotation von 3D-Modellen

3D-Modelle bestehen in der Regel aus einer Ansammlung von Linien, Dreiecken und Vierecken, die durch zusätzliche Informationen wie z. B. Normalen oder Materialeigenschaften ergänzt werden. Damit einzelne Teile von solchen Modellen annotiert werden können, müssen die Modelle zuvor vom Anwender segmentiert werden, d. h. in einzelne Gruppen von Primitiven zerlegt werden. Um diesen aufwendigen Prozess zu vereinfachen und zumindest teilweise zu automatisieren, gibt es mehrere algorithmische Methoden. Attene et al. zeigen in ihrem Tool ShapeAnnotator [2], wie durch Anwendung verschiedener Algorithmen, für sowohl mechanische Objekten als auch für das Modell eines Menschen eine gute Vorarbeit geleistet werden kann. Auftretende Ungenauigkeiten oder Fehler lassen sich anschließend durch eine manuelle Auswahl beheben, bevor der Segmentierung ein Element aus der entsprechenden Ontologie zugeordnet wird. Für die spezielle Domäne der Möbel haben zudem Symonova et al. [28] ein Verfahren entwickelt, welches Objekte segmentiert, ihre Form und Verbindung zu anderen Segmenten analysiert und aus diesen Informationen einen Feature Vektor berechnet. Dieser Vektor wird daraufhin mit Werten aus einer Datenbank verglichen und dadurch auf einen Vektor abgebildet, der semantische Beschriftungen beinhaltet. Der Benutzer hat dann später die Möglichkeit Anfragen an die Datenbank zu schicken, indem er entweder eine Suchanfrage in natürlicher Sprache formuliert oder ein Beispielmodell des gesuchten Objekts angibt.

5.4.2 Problemstellungen bei Annotation von Visualisierungen im 3D-Raum

Im Gegensatz zu den algorithmischen Verfahren für die Vereinfachung der semantischen Annotation von bestehenden 3D-Modellen, existieren im Zusammenhang mit wissenschaftlicher Visualisierung jedoch noch keinerlei verwertbare Ansätze oder Techniken. Eine solche Annotation ist zum einen deshalb besonders schwierig zu realisieren, da sich Visualisierungstechniken im dreidimensionalen Raum im Bezug auf ihren Aufbau stark voneinander unterscheiden. So sind für die Visualisierung von Volumen generell andere Verfahren notwendig als beispielsweise für Strömungs-Visualisierung. Zum anderen verstärken sich die in Abschnitt 5.3.3 genannten Sichtbarkeitsprobleme durch das Hinzukommen einer zusätzlichen Raumdimension. Diese Problemstellungen werden in den folgenden beiden Abschnitten genauer ausgeführt.

Visualisierungstechniken: Einige Techniken, wie beispielsweise dreidimensionale Scatter Plots, die in der wissenschaftlichen Visualisierung Anwendung finden, bestehen aus einer Anordnung einzelner Objekte (Würfel, Ellipsoide, Linien etc.). Diese verhalten sich analog zu zweidimensionalen Visualisierungen und können problemlos auf ähnliche Art und Weise annotiert werden. Häufiger jedoch werden hingegen Volumengrafiken oder Isoflächen erstellt, bei denen sich dieser Prozess wesentlich

komplizierter gestaltet. Die Datenpunkte werden dabei nicht auf abgeschlossene Objekte, sondern auf Teilbereiche dieser Objekte abgebildet, was die spätere Abbildung von einem Blickpunkt auf die zugehörigen Daten erschwert.

Sichtbarkeit von Objekten: Damit dreidimensionale Visualisierungen sinnvoll betrachtet werden können, ist es notwendig bestimmte Interaktionstechniken zur Navigation durch den Raum bereitzustellen. Diese verhindern jedoch nicht, dass sich, je nach gewählter Ansicht, Objekte überdecken oder überlappen können. Befindet sich in der Szene außerdem eine Lichtquelle, so können durch den Schattenwurf eines Objekts mehrere andere verdeckt werden. Bei der Durchführung von Eye-Tracking Experimenten ist daher die Wahrscheinlichkeit, dass sich an einer Blickposition mehrere, für den Proband jedoch nicht sichtbare Objekte befinden deutlich höher als bei zweidimensionalen Darstellungen.

5.4.3 Anpassung des Lösungskonzepts

Das bisher vorgestellte, auf der Visualisierungs-Pipeline basierende Konzept, beschränkt sich grundsätzlich nicht auf den zweidimensionalen Raum. Wie genau eine semantische Annotation im Rendering-Prozess erfolgen kann, ist jedoch abhängig vom Aufbau der Visualisierungs-Anwendung. Im Folgenden werden daher für drei unterschiedliche Herangehensweisen an dreidimensionale Grafiken erläutert, wie sich jeweils eine Annotation für wissenschaftliche Visualisierungen umsetzen lässt. Neben dem gebräuchlichen Ansatz unter der Verwendung von OpenGL, werden hierbei zwei webbasierte Konzepte vorgestellt. In jedem Fall ist zusätzlich eine geeignete Bibliothek notwendig, welche die Verwendung von Ontologien ermöglicht.

OpenGL: Angenommen der Visualisierung liegt ein Szenengraph zugrunde, d. h. die dreidimensionale Szene wird mit Hilfe einer objektorientierten Datenstruktur aufgebaut. Dabei werden Transformationen sowie grafische Elemente durch Instanzen von entsprechenden Klassen realisiert. Diese Instanzen werden, wie in Abbildung 5.7 dargestellt, auch als Knoten eines Szenengraphs bezeichnet. Für die grafischen Elemente können in diesem Fall, während des Mapping-Schritts in der Visualisierungs-Pipeline, zusätzlich zu den bereits in den Knoten enthaltenen Attributen, wie beispielsweise Farbe und Position des Objekts, auch semantische Informationen in Form einer Referenz auf ein Ontologie-Element hinzugefügt werden.
Dieser Ansatz lässt sich außerdem durch geringe Anpassung auch auf Visualisierungen ohne Szenengraph anwenden. Hierbei muss die semantische Information lediglich an einer anderen Stelle, wie beispielsweise direkt in der entsprechenden Klasse, durch die ein bestimmtes grafisches Element repräsentiert wird, mit abgespeichert werden.
Für die Durchführung einer Eye-Tracking Studie ist zudem die Implementierung

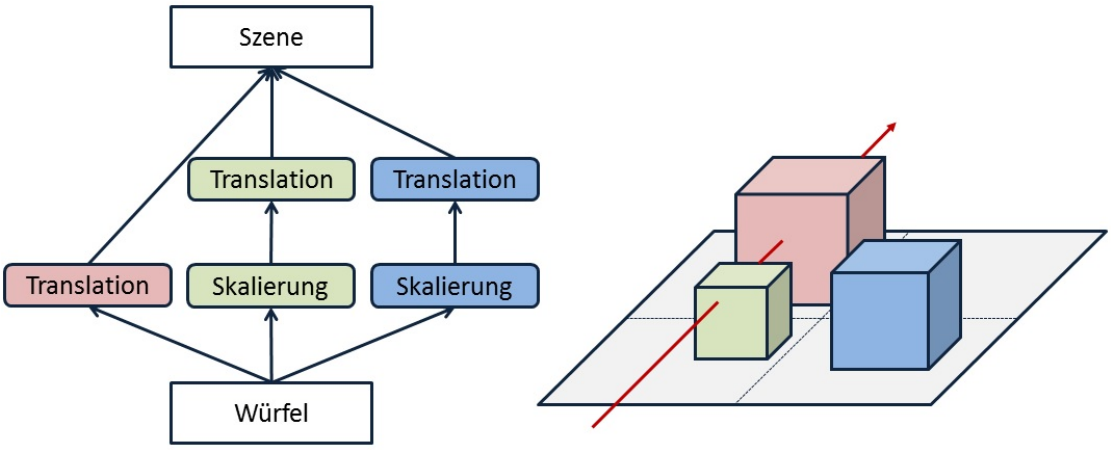

Abbildung 5.7: Darstellung eines Szenengraphs (links) und der zugehörigen Szene (rechts). Welche Knoten im Graph welchem Element in der Szene entsprechen, ist durch unterschiedliche Farben gekennzeichnet. Ein roter Pfeil symbolisiert einen Sichtstrahl, der für die Selektion von Elementen in die Szene geschossen wird.

eines Picking-Algorithmus notwendig, der für jede aufgezeichnete Blickposition die entsprechenden Knoten im Szenengraph (Visualisierung mit Szenengraph) bzw. die entsprechende grafische Elemente (Visualisierung ohne Szenengraph) zurückgibt, damit daraufhin die semantischen Informationen ausgewertet werden können. Picking wird z. B. durch das Verkleinern des Sichtfeldes auf einen wenige Pixel umfassenden Sichtstrahl (siehe Abbildung 5.7, rechts) realisiert. Für diesen Strahl werden anschließend die Schnittpunkte mit den einzelnen Objekten in einer Szene errechnet und anhand ihrer Tiefe sortiert.

XML3D: Eine weitere Möglichkeit besteht darin, den Visualisierungs-Prozess, ähnlich wie auch bei D3, ins Web zu verlagern. Dies hat den Vorteil, dass sämtliche der Technologien des Semantischen Web für den Einsatz im Internet konzipiert wurden und daher dort zum Teil bereits integriert sind. Außerdem existieren bereits mehrere Bibliotheken, die das Erstellen von dreidimensionalen Szenen im Web erleichtern. Besonders interessant im Zusammenhang mit semantischer Annotation ist hierbei XML3D[1], da diese Bibliothek XML Syntax verwendet und auf W3C Standards aufbaut. 3D-Modelle werden durch ein `mesh` Tag aus einem XML oder JavaScript Object Notation (JSON) Dokument geladen (siehe Code-Beispiel 5.1) und können direkt durch Hinzufügen eines entsprechenden Attributs annotiert werden (siehe Abschnitt 5.3). Alternativ ist es auch möglich die Annotation in

[1] Website: `http://www.xml3d.org`

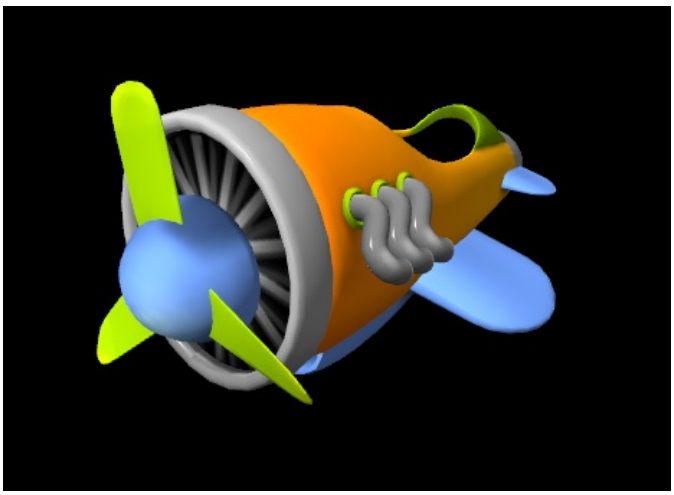

Abbildung 5.8: Im Webbrowser unter Verwendung von XML3D gerendertes 3D-Modell. Das Modell wurde hierbei aus einer externen XML Datei in ein HTML Dokument eingebettet.

```
1 <group id="plane" >
2     <group shader="resource/plane.xml#shader_matte" >
3         <mesh src="resource/plane.xml#mesh1" type="triangles"/>
4     </group>
5     ...
6 </group>
```

Quellcode 5.1: Einbinden eines 3D-Modells aus der externen Datei `plane.xml`. Hierfür werden einzelne Teile der XML Datei durch entsprechende Attribute referenziert.

den 3D-Modellen selbst vorzunehmen, da sich Dateiformate wie XML und JSON ohne technische Schwierigkeiten um zusätzliche semantische Elemente erweitern lassen. Das Code-Beispiel 5.1 zeigt den Aufbau eines 3D-Modells unter Verwendung von XML3D. Einzelne Dreiecke des Meshes, die dazugehörigen Normalen sowie Texturkoordinaten lassen sich durch verschiedene `float` Tags definieren (Zeile 6-8). Das vollständige, im Browser gerenderte Modell ist in Abbildung 5.8 zu sehen[2]. Zur Selektion einzelner Komponenten wird bei XML3D zusätzlich ebenfalls ein Picking-Algorithmus benötigt.

[2]Quellcode: `http://xml3d.github.io/xml3d-examples/examples/externalXml/externalXml.xhtml`

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xml3d xmlns="http://www.w3.org/1999/xhtml">
3      ...
4      <!-- Mesh Data -->
5      <data id="base">
6          <float3 name="position" seqnr="0.0">1.84219 ... </float3>
7          <float3 name="normal" seqnr="0.0">0.24334 ... </float3>
8          <float2 name="texcoord" seqnr="0.0">0.033889 ... </float2>
9      </data>
10 </xml3d>
```

Quellcode 5.2: Auszug aus der Datei `plane.xml`. Innerhalb von `float` Tags werden entsprechend Dreiecke, deren Normalen sowie Texturkoordinaten festgelegt.

three.js: Die Beispielgalerie der Bibliothek `three.js`[3] zeigt eindrucksvoll, wie auch ohne Verwendung eines HTML 5 Canvas Elements einfache dreidimensionale Szenen auf der Grundlage von CSS Transformationen erstellt werden können. In Abbildung 5.9 ist die dreidimensionale Visualisierung eines Moleküls[4] abgebildet, die vollständig in einem Webbrowser gerendert wurde. Die grafischen Elemente werden hierbei durch gewöhnliche `div`-Tags erstellt und verhalten sich daher sehr ähnlich zu Grafiken, welche in D3 erstellt wurden. Anders als bei den beiden obigen Ansätzen, müsste in diesem Fall kein Algorithmus implementiert werden, der berechnet, wie die Fixationspunkte eines Probanden auf grafische Elemente abgebildet werden. Da solche Szenen ausschließlich aus transformierten DOM Elementen besteht, wird dieser Abbildungsvorgang vollständig vom jeweiligen Webbrowser übernommen. Hierfür traversiert dieser lediglich den DOM Baum, der dem HTML Dokument zugrunde liegt, und gibt das entsprechende Element an der angegebenen Position zurück.

5.4.4 Anpassung der Ontologie-Ebenen

Prinzipiell unterscheidet sich der Aufbau der Ontologie-Ebenen kaum von dem in Abschnitt 5.2.2 vorgestellten Schema. Die bisher aus dem zweidimensionalen Raum bekannten grafischen Primitive von Ebene 1 der Visualisierungs-Ontologie bleiben bestehen und zusätzlich werden weitere Primitive, wie z. B. Dreiecke, Vierecke, Kugeln usw., hinzugefügt. In Ebene 2 müssen die benötigten Visualisierungstechniken anschließend analysiert und aus den grafischen Primitiven aufgebaut werden. Hierbei ist es besonders wichtig herauszufinden, welche Relationen für das Durchführen der nachfolgenden Eye-Tracking

[3] Website: `http://www.threejs.org`
[4] Quellcode: `http://threejs.org/examples/css3d_molecules.html`

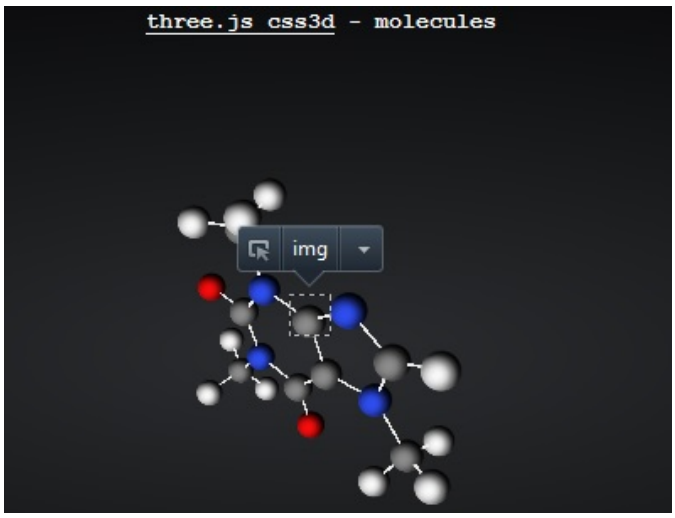

Abbildung 5.9: Auf der Basis von CSS Transformationen erstellte Darstellung eines Moleküls. Der Zugriff auf die einzelnen Visualisierungs-Elementen ist daher direkt durch das DOM möglich. In der Mitte der Abbildung wurde zur Verdeutlichung eines der Visualisierungs-Elemente selektiert.

Studie relevant sind, da ansonsten durch die hinzugekommene Dimension unnötiger Aufwand bei der Auswertung entsteht. Das Visualisierungsmodell (Ebene 3) besteht, wie auch im zweidimensionalen Fall, aus sämtlichen Instanzen von Ebene 2 und bedarf daher keiner besonderen Anpassung.

6 Implementierung

Aufbauend auf dem vorangegangenen Lösungskonzept, erläutert dieses Kapitel einige der Kernaspekte der Implementierung des Prototyps. Umgesetzt wurde hierbei die vollständige automatische Annotation der SVG Primitive in D3, sowie eine beispielhafte Annotation der folgenden drei in Abschnitt 2.2 vorgestellten Visualisierungstechniken: Säulendiagramme, Knoten-Kanten Diagramme und Scatter Plots. Zunächst wird vorgestellt, mit welchen Browsern das Visualisierungs-Frameworks D3 verwendet werden kann, welche Schritte zur Kompilierung des Frameworks erforderlich sind sowie welche JavaScript Bibliotheken dabei benutzt wurden. Anschließend folgt die Beschreibung der Implementierung, wobei neben der Entwicklung einer Visualisierungs-Ontologie außerdem die Integration einer semantischen Annotation in das Framework ausführlich beschrieben wird.

6.1 Vorbedingungen

An dieser Stelle wird zunächst eine Reihe von Vorbedingungen aufgeführt, die zur Verwendung und zur Kompilierung des Frameworks nötig sind (siehe auch offizielles Wiki von D3[1]). Sämtliche die Implementierung des Prototyps betreffende Aussagen beziehen sich auf das Visualisierungs-Framework D3 in Version 3.1.4.

6.1.1 Unterstützte Browser

Das Visualisierungs-Framework D3 baut auf verschiedenen aktuellen Web Technologien auf und setzt aus diesem Grund die Benutzung eines moderner Webbrowsers voraus. D3 verwendet neben JavaScript und der W3C DOM API vor allem SVG als Ausgabeformat, sowie CSS3 Transitions um Animationen zu realisieren. Welche Browser-Versionen minimal erforderlich sind, lässt sich anhand der Kompatibilität bezüglich der SVG Spezifikation des W3C festmachen, da die Visualisierungen ohne die Unterstützung dieser Spezifikation in der Regel nicht korrekt dargestellt werden können. Die in der nachfolgenden Tabelle aufgetragenen Browser-Versionen stammen von www.caniuse.com (1. Juni 2013) und beschreiben, welche Webbrowser die aktuelle SVG Spezifikation [10]

[1] Offizielles Wiki: `https://github.com/mbostock/d3/wiki`

unterstützen bzw. welche Webbrowser sie nicht unterstützen:

Internet Explorer	Firefox	Chrome	Safari	Opera	Android Browser
					2.3
					3.0
8.0	19.0	25.0			4.0
9.0	20.0	26.0	5.1		4.1
10.0	21.0	27.0	6.0	12.1	4.2
11.0	22.0	28.0		15.0	

6.1.2 Lokale Entwicklung mit D3

Sowohl die Bibliothek jOWL (siehe Abschnitt 6.1.4) als auch das XMLHttpRequest Modul `xhr` von D3 verwenden die XMLHttpRequest API, welche das asynchrone und dynamische Nachladen von Dateien im Browser ermöglicht. Für die lokale Entwicklung oder Betrachtung von Visualisierungen muss daher allerdings ein Webserver auf dem entsprechenden System eingerichtet werden, da ansonsten auf diese API nicht zugegriffen werden kann. Die einfachste Möglichkeit hierfür stellt die Programmiersprache Python bereit, welche einen bereits voll funktionsfähigen Webserver beinhaltet, der lediglich durch einen Aufruf in der Kommandozeile gestartet werden muss. Dazu navigiert man in den gewünschten Ordner, der als Stammverzeichnis für den Webserver dienen soll und startet den Server mit folgendem Befehl in Python 2.7:

```
python -m SimpleHTTPServer 8080
```

beziehungsweise ab Python 3 durch:

```
python -m http.server 8080
```

Die Zahlenkombination 8080 bezeichnet dabei den Port des Webservers und kann bei Bedarf entsprechend angepasst werden. Im Browser können die im Stammverzeichnis abgelegten Dokumente nun durch die URL http://localhost/:<PORT> erreicht werden. Wird außerdem zusätzliche Funktionalität wie z. B. MySQL, PHP oder Perl Unterstützung benötigt, so bietet es sich an stattdessen die von Apache Friends entwickelte, plattformunabhängige Distribution XAMPP[2] zu verwenden.

[2]Website: `http://www.apachefriends.org/de/xampp.html`

6.1.3 Kompilieren von D3

Zur Implementierung des Lösungskonzept wurden an einem Modul des Frameworks D3 Änderungen vorgenommen, sowie ein weiteres Modul eingefügt. Um diese Änderungen auch in die resultierende JavaScript Datei `d3.js` und die minifizierte Version `d3.min.js` zu übertragen, ist das erneute Kompilieren des Frameworks nötig. D3 verwendet zur Verwaltung der Module und somit auch für die Kompilierung Node.js, ein ursprünglich eine für Netzwerkanwendungen konzipierte Plattform. Bevor daher das mitgelieferte Makefile ausgeführt werden kann, muss zunächst Node.js installiert und im Stammverzeichnis von D3 der Befehl `npm install` ausgeführt werden. Durch diesen Aufruf des Paket Managers von Node.js werden alle notwendigen Abhängigkeiten installiert. Anschließend kann das Projekt unter Verwendung des Makefiles neu kompiliert werden.

6.1.4 Verwendete Bibliotheken

Zur Implementierung des Lösungskonzepts wurde neben D3 zwei weitere JavaScript Bibliotheken benötigt, welche im Folgenden kurz beschrieben werden.

jQuery 1.8.3: Eine auf verschiedenen Browsern lauffähige Bibliothek, welche unter anderem den Zugriff auf DOM Elemente, sowie deren Manipulation erleichtert.

jOWL 1.0: jOWL stellt ein Plugin für jQuery dar und ermöglicht es Ontologien in Form von RDF/S oder OWL Dateien zu laden und in Dokumente einzubinden.

6.2 Implementierung der Visualisierungs-Ontologie

Die im Lösungskonzept vorgestellte Visualisierungs-Ontologie (siehe Abschnitt 5.3.2) wurde in OWL Syntax unter Verwendung des Editors Protégé[3] implementiert. Aus technischer Sicht ist es in OWL möglich, die einzelnen Ebenen der Ontologie auf unterschiedliche Dateien zu verteilen und diese durch `imports` Statements miteinander zu verknüpfen. Allerdings verwendet der hier beschriebene Prototyp zur Verwaltung der Ontologie die einzige existierende JavaScript Bibliothek jOWL. Diese ist dahingehend beschränkt, dass sie gleichzeitig lediglich eine Ontologie verwalten kann und die von OWL definierten `imports` ignoriert. Daher wurden die drei Ebenen der Visualisierungs-Ontologie in einer einzigen Datei zusammengefasst.

Ebene 1 - Taxonomie grafischer Primitive: Entsprechend der in Abbildung 5.5 dargestellten hierarchischen Anordnung, wurden in Protégé zunächst die benötigten Klassen für die SVG Primitive angelegt und anschließend durch die Relation

[3] Website: `http://protege.stanford.edu`

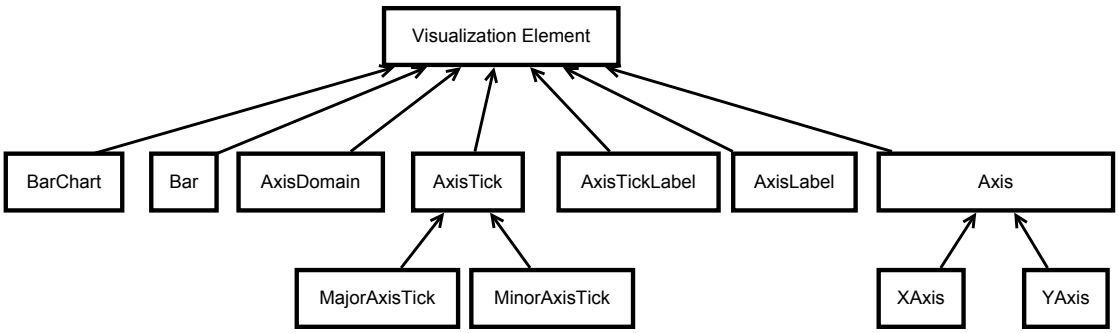

Abbildung 6.1: Hierarchische Struktur der Visualisierung eines Säulendiagramms. Alle Visualisierungs-Elemente wurden von der Klasse `VisualizationElement` abgeleitet und weiter differenziert. Die einzelnen Elemente sind gleichzeitig außerdem Subklasse eines SVG Primitivs aus Ebene 1 der Ontologie (nicht Teil der Abbildung).

`rdfs:subClassOf` in Beziehung gesetzt. Des Weiteren wurde eine `Data` Klasse, sowie zusätzliche Klassen für sämtliche in JavaScript verfügbare Datentypen erstellt. Die Klassen der JavaScript Datentypen wurden schließlich ebenfalls mit `rdfs:subClassOf` als Subklassen von `Data` definiert.

Ebene 2 - Visualisierungsschema: Die Implementierung des Visualisierungsschemas setzt sich aus mehreren Schritten zusammen und wird im Folgenden anhand der beispielhaften Integration eines Säulendiagramms in die Visualisierungs-Ontologie demonstriert.
Zunächst wurden hierfür die relevanten Bestandteile der Visualisierung analysiert, um zu ermitteln, wie diese mit Hilfe einer Ontologie dargestellt werden können. Für alle noch nicht in der Visualisierungs-Ontologie befindlichen Bestandteile wurde anschließend in Protégé eine entsprechende Klasse erzeugt. Diese Klassen von Visualisierungs-Elementen sind jeweils Subklassen von `VisualizationElement` und können zusätzlich weitere Relationen beinhalten. So wurde für die Säulen eines Säulendiagramms eine Klasse `Bar` erstellt und mit `rdfs:subClassOf` eine Beziehung zu `VisualizationElement`, sowie `Rectangle` aus Ebene 1 der Ontologie hergestellt. Abbildung 6.1 zeigt sämtliche Klassen der Visualisierung des Säulendiagramms, sowie ihre hierarchische Struktur. Die Subklassen-Relationen zu den entsprechenden SVG Primitiven wurden aus Gründen der Übersichtlichkeit nicht in der Abbildung aufgeführt.

Durch die bisher hinzugefügten Relationen wurde definiert, dass es sich bei den jeweiligen Ontologie-Elementen um Teile einer Visualisierung handelt, sowie, dass diese Teile auf SVG Primitiven beruhen. Wie genau aus den Elementen eine entsprechende Visualisierungstechnik aufgebaut wird, steht jedoch noch nicht fest. Um

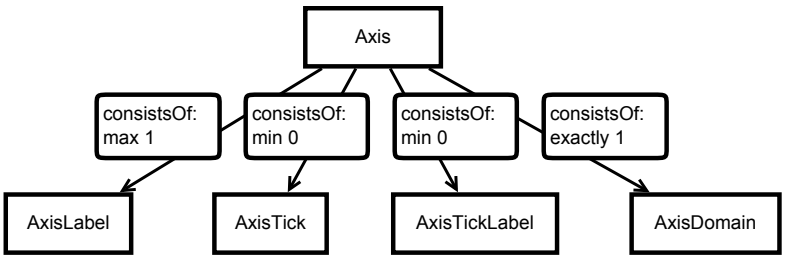

Abbildung 6.2: Das Visualisierungs-Element `Axis` setzt sich aus den Visualisierungs-Elementen `AxisLabel`, `AxisTick`, `AxisTickLabel` und `AxisDomain` zusammen. Angelehnt an die in Protégé übliche Schreibweise, sind die Kardinalitäten durch die Schlüsselwörter `exactly x` (genau `x` Objekte), `min x` (mindestens `x` Objekte) und `max x` (maximal `x` Objekte) angegeben.

zu modellieren, welche zusammengesetzten Komponenten aus welchen einzelnen Visualisierungs-Elementen bestehen, wurde eine spezielle `consistsOf` Relation eingeführt. Durch Verwendung dieser Relation kann ausgedrückt werden, dass ein Visualisierungs-Element aus einem Visualisierungs-Element oder mehreren Visualisierungs-Elementen besteht. In Abbildung 6.2 sind die für die Achsen vorgenommenen `consistsOf` Relationen mit ihren entsprechenden Kardinalitäten dargestellt. Die Klasse `Axis` besteht somit aus Instanzen der Klassen `AxisLabel`, `AxisTick`, `AxisTickLabel` sowie `AxisDomain`. Wie viele Objekte einer Relation zugewiesen werden können, wird durch die Kardinalitätseinschränkungen `exactly x` (genau `x` Objekte), `min x` (mindestens `x` Objekte) und `max x` (maximal `x` Objekte) festgelegt. Diese sind angelehnt an die von Protégé verwendete Schreibweise und bezeichnen die in der OWL Spezifikation definierten Einschränkungen `owl:cardinality`, `owl:minCardinality` sowie `owl:maxCardinality`. Analog muss schließlich die `BarChart` Klasse aus den Visualisierungs-Elementen `XAxis`, `YAxis` und `Bar` zusammengesetzt werden. Das Code-Beispiel 6.1 zeigt den von Protégé generierten OWL Code für die Klasse `BarChart` (Zeile 1). Hierbei wurde definiert, dass in jedem Säulendiagramm jeweils exakt eine x-Achse (Zeile 3-9) und eine y-Achse (Zeile 10-16) vorhanden sein muss, sowie mindestens eine Säule (Zeile 17-23).

Ebene 3 - Visualisierungsmodell: Das Visualisierungsmodell enthält die Annotationsdaten und wird von der annotierten Visualisierung automatisch generiert (siehe Abschnitt 6.3.1 und 6.3.2).

```
<owl:Class rdf:about="&base;BarChart">
    <rdfs:subClassOf rdf:resource="&base;VisualizationElement"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&base;consistsOf"/>
            <owl:onClass rdf:resource="&base;XAxis"/>
            <owl:qualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger
                ">1</owl:qualifiedCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&base;consistsOf"/>
            <owl:onClass rdf:resource="&base;YAxis"/>
            <owl:qualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger
                ">1</owl:qualifiedCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&base;consistsOf"/>
            <owl:onClass rdf:resource="&base;Bar"/>
            <owl:minQualifiedCardinality rdf:datatype="&xsd;
                nonNegativeInteger">1</owl:minQualifiedCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Quellcode 6.1: Durch Protégé erstellter OWL Code für eine Säulendiagramm Klasse. Die Klasse `BarChart` wird als eine Subklasse von `VisualizationElement` definiert und besteht aus genau zwei Achsen, einer `XAxis` sowie einer `YAxis`. Zusätzlich enthält ein `BarChart` eine oder mehrere Instanzen der Klasse `Bar`.

6.3 Implementierung von semantischer Annotation in D3

Nach dem Entwurf und der Erstellung einer Visualisierungs-Ontologie, welche die in Abschnitt 2.2 vorgestellten Visualisierungstechniken enthält, wurde die Integration einer automatischen Annotation von SVG Primitiven in das Framework D3 durchgeführt. Anschließend wurden die vorgestellten Visualisierungstechniken schließlich mit Hilfe der Ontologie beispielhaft annotiert.

6.3.1 Implementierung der Annotation von SVG Primitiven

Das Visualisierungs-Framework D3 besteht aus einer Vielzahl einzelner Module. Diese sind als JavaScript Dateien in einer entsprechenden Ordnerstruktur abgelegt und werden bei der Kompilierung durch das Node.js Modul `smash` zu einer Datei zusammengefügt. Der Implementierungsvorgang gliedert sich in zwei Schritte, die im Folgenden beschrieben werden. Der erste Schritt befasst sich mit dem Erstellen eines `semantics` Moduls zur Annotation, während der zweite Schritt zeigt, in welche Module von D3 die Annotation integriert wurde.

Implementierung eines `semantics` Moduls: Für die Annotation von SVG Primitiven wurde ein neues Modul mit der Bezeichnung `semantics` erstellt. Dazu wurde im `src` Verzeichnis zunächst ein `semantics` Ordner, sowie darin die Datei `index.js` angelegt. Diese enthält den Pfad zu einzelnen Skripten, die Teil des Moduls sind, und ermöglicht Node.js dadurch die entsprechenden `require` Aufrufe im Quellcode aufzulösen. Angelehnt an die Architektur von D3, definiert `semantics.js` lediglich die Klasse `d3.semantics`, während die Funktionalität der Klasse in die Datei `annotate.js` ausgelagert wird. So können dem Modul zukünftig durch Erstellen einer neuen Skriptdatei weitere Funktionen hinzugefügt werden.
Die Funktion `d3.semantics.annotate` befindet sich in der Datei `annotate.js` und implementiert die Annotation der SVG Primitive. Sie besitzt als einzigen Eingabeparameter das entsprechende DOM Element, welches mit semantischen Informationen versehen werden soll. Die Annotation wurde unter Verwendung des W3C Standards RDFa (siehe Abschnitt 2.1.2) durchgeführt und erfolgt daher durch Hinzufügen von entsprechenden Attributen im DOM Baum. Sämtliche SVG Primitive in Ebene 1 der Visualisierungs-Ontologie (siehe Abschnitt 6.2) sind in Form einer Klassenhierarchie organisiert und müssen zur Annotation instanziiert werden. RDFa verwendet zum Kennzeichnen von Instanzen das `typeof` Attribut. Der Inhalt des Attributs besteht aus einem Verweis auf das entsprechende Element in der Visualisierungs-Ontologie, der mit Hilfe der jOWL Bibliothek erzeugt werden kann. Hierfür wird das Objekt `jOWL(<Ontologie-Element>)` aufgerufen, welches das zugehörige Ontologie-Element zurückgibt. Dieses Ontologie-Element enthält unter anderem die

```
1 function append() {
2     return this.appendChild(d3.semantics.annotate(d3_document.
          createElementNS(this.namespaceURI, name)));
3 }
4
5 function appendNS() {
6       return this.appendChild(d3.semantics.annotate(d3_document.
            createElementNS(name.space, name.local)));
7 }
```

Quellcode 6.2: Auszug aus der Datei `selection/append.js`. Bevor eine der Funktionen `append` bzw. `appendNS` ein DOM Element an den DOM Baum des Dokuments anhängt, wird entsprechend die semantische Annotation vorgenommen.

gesuchte URI, die zum setzen des Verweises nötig ist. Die Annotation eines einzelnen DOM Elements `element` kann also durch `element.setAttribute("typeof", jOWL(<Ontologie-Element>).URI)` realisiert werden.

Integration der Annotation: Das Visualisierungs-Framework D3 generiert sämtliche DOM Elemente, und somit auch sämtliche SVG Primitive, durch jeweils zwei Funktionen in den Dateien `selection/append.js` und `selection/insert.js`. Die Annotation wurde in allen diese Funktionen integriert, damit sichergestellt werden kann, dass im Nachhinein beim Erstellen von Visualisierungen keine visuellen Elemente entstehen, die nicht vorher durch mindestens ein Attribut annotiert wurden. Codebeispiel 6.2 zeigt den vollständigen Quellcode der Funktionen `append` und `appendNS`. Hierbei ist in den Zeilen 2 und 6 zu sehen, wie zuerst durch `d3_document.createElementNS` ein DOM Element erstellt und anschließend mit Hilfe des `semantics` Moduls annotiert wird. Erst nach der Annotation wird das Element an den DOM Baum angehängt. Die Annotation der entsprechenden zwei Funktionen in `selection/insert.js` erfolgte analog. Das Framework D3 enthält außerdem das vielseitige Modul `axis` zur Erstellung von Achsen in Diagrammen. Da dieses eigenständig SVG Elemente erzeugt, wurden auch diese durch Hinzufügen von `typeof` Attributen in der Datei `svg/axis.js` entsprechend annotiert.

6.3.2 Annotation von Visualisierungen

Um den praktischen Nutzen des Lösungskonzepts zu demonstrieren, wurden unter Verwendung von D3 Säulendiagramme, Knoten-Kanten Diagramme und Scatter Plots semantisch annotiert. Im Folgenden werden die beiden zur Annotation notwendigen Schritte beschrieben. In Schritt 1 werden hierbei alle notwendigen Bibliotheken in das Dokument eingebunden, sowie die Visualisierungs-Ontologie geladen. So konnte anschließend in

Schritt 2 den Visualisierungs-Elementen die jeweiligen semantischen Informationen zugewiesen werden.

Schritt 1 - Vorbereitung des Dokuments: Um mit dem webbasierten Visualisierungs-Framework D3 eine annotierte Visualisierung erstellen zu können, muss zunächst ein HTML Dokument angelegt werden. Dieses besteht, wie auch gewöhnliche Webseiten, aus einem Paar von `<html></html>` Tags, welche die Tags `<head></head>` und `<body></body>` enthalten. Während durch `<head></head>` die HTML-Kopfdaten, wie beispielsweise den Titel oder Metainformationen über das Dokument, markiert werden, befinden sich innerhalb von `<body></body>` sämtliche Inhalte des Dokuments.
Das Laden der verschiedenen JavaScript Bibliotheken, der Ontologie, sowie die darauffolgende Erstellung der Visualisierung und deren Annotation wurde im Body des Dokuments implementiert. Das Code-Beispiel 6.3 zeigt das Grundgerüst, in welches später der entsprechende Code zum Erstellen einer Visualisierung bzw. deren Annotation eingefügt werden kann. Beim Laden der Bibliotheken (Zeile 5-7 und Zeile 11-15) muss besonders darauf geachtet werden, welche Abhängigkeiten der Bibliotheken untereinander existieren. jQuery wird von der Bibliothek jOWL benötigt und muss daher zuerst in das Dokument eingebunden werden. Nachdem jQuery und jOWL erfoglreich geladen wurden, kann die entsprechende Visualisierungs-Ontologie durch den Aufruf von `jOWL.load` (Zeile 9) geöffnet werden. Bei erfolgreicher Ausführung von `jOWL.load` wird von anschließend ein Callback aufgerufen, in dem auf die geparsten Ontologie-Elemente zugegriffen werden kann. Erst innerhalb dieses Callbacks wird das Visualisierungs-Framework D3 nachgeladen (Zeile 11-14). Dieser geschachtelte Aufbau ist notwendig, da im `semantics` Modul von D3 die Bibliothek jOWL zur Annotation verwendet wird. Ab Zeile 17 folgt die Implementierung der Visualisierungstechnik und deren Annotation.

Schritt 2 - Annotation der Visualisierungs-Elemente: Damit mit der Annotation begonnen werden kann, muss in das Code-Skelett aus Schritt 1 zuvor die gewünschte Visualisierungstechnik eingefügt werden. Da das Erstellen von Visualisierungen in D3 nicht Teil dieser Studienarbeit war, wurden zur Annotation die Beispiele von Bostock[4] verwendet und entsprechend modifiziert.
Der erste Teil der Annotation setzt die Instanziierung der in Ebene 2 der Visualisierungs-Ontologie definierten Klassen von Visualisierungs-Elementen um. Dies wird beispielhaft anhand des SVG Wurzel-Elements in Code-Beispiel 6.4 gezeigt. Nachdem das entsprechende Visualisierungs-Element durch den Aufruf von `append` oder `insert` in den DOM Baum eingefügt wurde (Zeile 1), wird anschließend ein `typeof` Attribut angelegt. Dem Attribut wird als Wert, wie auch bei der Implementierung des `semantics` Moduls, die URI des jeweiligen Ontologie-Elements zugewiesen.

[4] Beispiel-Galerie: `https://github.com/mbostock/d3/wiki/Gallery`

```
<!DOCTYPE html>
<html>
<head><title>Grundgerüst für Visualisierungen</title></head>
<body>
<script src="jquery-1.8.3.min.js"></script>
<script src="jOWL.js"></script>
<script>
var options = {locale: 'en'};
jOWL.load("visontology.owl", function() {

    var documentBody = document.getElementsByTagName('body').item(0);
    var d3Script = document.createElement('script');
    d3Script.setAttribute('src', 'd3.js');
    documentBody.appendChild(d3Script);
    d3Script.onload = function() {

     // Hier Implementierung der Visualisierung einfügen

    };
}, options);
</script>
</body>
</html>
```

Quellcode 6.3: Grundgerüst für das Erstellen einer annotierten Visualisierung. Hierfür werden die Bibliotheken jQuery und jOWL nacheinander in das Dokument eingebunden, damit anschließend die Visualisierungs-Ontologie geladen werden kann. Nach dem Einbinden des Frameworks D3 beginnt schließlich die Implementierung der Visualisierung.

```
1 var svg = d3.select("body").append("svg")
2     .attr("width", width + margin.left + margin.right)
3     .attr("height", height + margin.top + margin.bottom)
4     .attr("typeof", jOWL("BarChart").URI);
```

Quellcode 6.4: Annotation des SVG Wurzel-Elements als Säulendiagramm. Hierfür wird dem Element ein `typeof` Attribut mit der URI der `BarChart` Klasse als Wert hinzugefügt.

Dieser Vorgang wird für alle übrigen Visualisierungs-Elemente wiederholt.

Im zweiten Teil der Annotation werden in die Visualisierung zusätzlich OWL Datatype Properties eingefügt. Diese Art von Properties dient dazu, einzelne Instanzen mit Datenwerten zu verknüpfen, wodurch Metainformationen über Visualisierungs-Elemente gespeichert werden können. In allen drei annotierten Visualisierungen wird jeweils festgehalten, welcher Datenpunkt dem entsprechenden Visualisierungs-Element zugrunde liegt, sowie von welchem Datentyp dieser ist. Weitere denkbare Metainformationen wären beispielsweise die Farbe oder Größe eines Elements, sowie dessen Abstand zu anderen Elementen. An dieser Stelle, d. h. wenn mehrere Relationen für ein SVG Element definiert werden sollen, tritt allerdings ein Problem im Zusammenhang mit RDFa auf. Die Spezifikation von XML, und dadurch auch die Spezifikation von SVG, legt fest, dass pro Element jedes Attribut nur einmal vorkommen darf. Um aber dennoch mehrere Relationen im Kontext von SVG zu erlauben, werden daher `<semantics>` Tags eingeführt. Diese besitzen per Definitionen keinen Inhalt und lassen sich in die entsprechenden SVG Elemente schachteln. Da sie außerdem nicht Teil einer Spezifikation sind, werden sie folglich beim Rendering-Prozess vom Webbrowser automatisch ignoriert. Das Code-Beispiel 6.5 zeigt, wie sich jeweils zwei Datatype Properties für die Säulen eines Säulendiagramms mit Hilfe von `<semantics>` Tags realisieren lassen (Zeile 12-14 bzw. Zeile 16-18). Für jede der beiden Relationen `originDataType` und `originDataPoint` wird ein `property` Attribut benötigt, welches die URI des Datatype Properties enthält, sowie ein `resource` Attribut für den Wert des Literals. Diejenigen Properties, die Beziehungen zwischen zwei Klassen darstellen, müssen nicht explizit für in jeder Instanz einer Visualisierung gesetzt werden, da sie bereits im Visualisierungsschema definiert sind.

6.3.3 Speichern der SVG Grafik

Oftmals ist es hilfreich eine von D3 erzeugte SVG Grafik lokal abzuspeichern zu können, um diese anschließend zu bearbeiten oder in andere Dokumente einzufügen. Auch kann es nützlich sein, solche Grafiken in anderen Anwendungen, wie beispielweise Eye-Tracking

```
1  var bars = svg.selectAll(".bar")
2      .data(data)
3      .enter().append("rect")
4      .attr("class", "bar")
5      .attr("typeof", jOWL("Bar").URI)
6      .attr("x", function(d) { return x(d.letter); })
7      .attr("width", x.rangeBand())
8      .attr("y", function(d) { return y(d.frequency); })
9      .attr("height", function(d) { return height - y(d.frequency); });
10
11     // Säulen im Säulendiagramm werden durch zwei Datatype Properties
           annotiert.
12     bars.append("semantics")
13     .attr("property", jOWL("originDataType").URI)
14     .attr("resource", jOWL("Array").URI);
15
16     bars.append("semantics")
17     .attr("property", jOWL("originDataPoint").URI)
18     .attr("resource", function(d) { return d.letter + " " + d.frequency; });
```

Quellcode 6.5: Annotation der Säulen eines Säulendiagramms. Für die Säulen werden jeweils `typeof` Attribute hinzugefügt, während die Datatype Properties mit Hilfe von `<semantics>` Tags realisiert werden.

Software, weiterverwenden zu können. Bei der Implementierung des Speichervorgangs von SVG Grafiken gilt es allerdings einige Dinge zu beachten. D3 erlaubt zur Formatierung des HTML Dokuments, und somit auch zur Formatierung der darin enthaltenen SVG Elemente, die Verwendung von CSS Stylesheets. Wird eine SVG Grafik, die in Form eines DOM Objekts im Browser vorliegt, in eine Datei geschrieben, so gehen diese zusätzlichen Informationen verloren. Ein weiteres Problem stellt das Öffnen eines Dialogfensters zum Speichern von Dateien dar. Eine von allen Webbrowsern unterstützte Möglichkeit, um Downloads von Dateien zu ermöglichen, kann mit Hilfe eines Servers realisiert werden. Dabei wird zunächst eine Anfrage an einen Server geschickt, der die gewünschten Daten mit einem Content-Disposition-Header versieht und diese anschließend zurückschickt. Dadurch wird dem Browser mitgeteilt, dass die entsprechenden Daten nicht geöffnet, sondern gespeichert werden sollen. Um das Speichern von SVG Grafiken ohne einen Server zu realisieren, gibt es die Möglichkeit Data-URIs zu verwenden. Das entsprechende DOM Objekt lässt sich hierbei zunächst mit Hilfe von Base64 als Data-URI in ASCII Zeichen kodieren (Code-Beispiel 6.6, Zeile 8) und anschließend durch ein `<img>` Tag als Bild deklarieren (Zeile 11). Dieses kann daraufhin durch Klicken der rechten Maustaste → „Bild speichern unter...“, gespeichert werden. Damit die SVG Grafik später von anderen Werkzeugen korrekt geladen werden kann, sollte außerdem zuvor angegeben werden, welcher Namespace und welche Version SVG verwendet wurde (Zeile 2).

```
1  // Setzen von Version und Namespace
2  var vis = d3.select("svg")
3          .attr("version", 1.1)
4          .attr("xmlns", "http://www.w3.org/2000/svg")
5          .node().parentNode.innerHTML;
6
7  // Base64 Enkodierung
8  var visEncoded = btoa(vis);
9
10 // Erstellen eines <img> Tags
11 d3.select("body")
12    .append("img")
13    .attr("src", "" + visEncoded);
```

Quellcode 6.6: Speichern einer D3 Visualisiernug mit Hilfe eines `<img>` Tags. Nach dem Hinzufügen des SVG Namespace sowie der SVG Version, wird die Grafik in Base64 kodiert und als `src` Attribut des `<img>` Tags angegeben.

Alternativ kann das Bookmarklet SVG Crowbar[5] benutzt werden, welches automatisch die aktuelle Webseite nach SVG Objekten durchsucht und das Dialogfenster zum Speichern öffnet. Dieses Bookmarklet integriert zusätzlich zudem die oben erwähnten CSS Abhängigkeiten.

6.3.4 Eye-Tracking mit annotierten Visualisierungen

Auf Grundlage der vorgenommenen Annotation lässt sich die Auswertung von Eye-Tracking Experimenten deutlich vereinfachen. So können zu einem gegebenen Fixationspunkt automatisch die betrachteten Visualisierungs-Elemente inklusive der jeweiligen semantischen Informationen ausgegeben werden. Dazu genügt ein Aufruf der JavaScript Funktion `document.elementFromPoint`, die als Parameter die beiden Koordinaten des Fixationspunkts übergeben bekommt. Diese durchsucht den DOM Baum eines Dokuments und gibt das Element zurück, welches sich an dieser Position befindet. Existieren an dieser Position mehrere, sich überdeckende oder überlappende Elemente, so wird das sich am weitesten im Vordergrund befindliche Element zurückgegeben. Durch Ausblenden der bereits abgearbeiteten Elemente und anschließend erneutes Aufrufen der Funktion, lassen sich folglich alle Elemente an einer bestimmten Position auffinden. Da alle semantischen Informationen sich in den jeweiligen Attributen der Elemente befinden, können diese schließlich mit Hilfe von JavaScript ausgewertet werden. Abbildung 6.3 zeigt eine annotierte Visualisierung eines Säulendiagramms in Verbindung mit einem Gaze Plot. Für die entsprechenden Fixationspunkte 1-6 im Gaze Plot, die als beschriftete rote

[5] Website: `http://nytimes.github.io/svg-crowbar/`

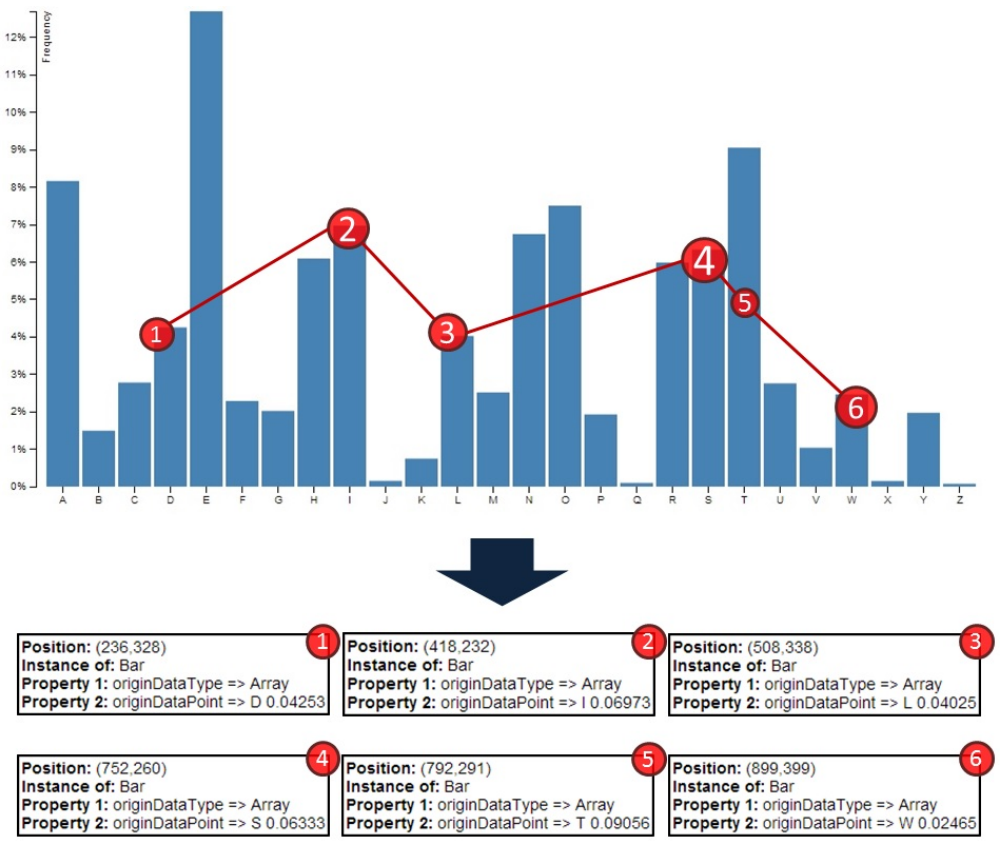

Abbildung 6.3: Eine mit Hilfe von D3 annotierte Visualisierung eines Säulendiagramms. Die im unteren Teil der Grafik aufgeführten semantischen Informationen wurden hierbei direkt anhand der Fixationspunkte des eingeblendeten Gaze Plots aus der Visualisierung extrahiert.

Kreise eingezeichnet sind, wurden die semantischen Informationen aus den Attributen der Visualisierungs-Elemente extrahiert und im unteren Teil der Abbildung dargestellt.

7 Zusammenfassung und Ausblick

Dieses Kapitel fasst das im Rahmen dieser Studienarbeit entwickelte Lösungskonzept, sowie die daraus resultierenden Ergebnisse zusammen. Anschließend wird ein Ausblick gegeben, welcher sich sowohl mit der Weiterentwicklung des Konzepts, als auch mit der Erweiterung des angefertigten Prototyps beschäftigt.

7.1 Zusammenfassung

Die Zielsetzung dieser Studienarbeit bestand darin, ein Lösungskonzept für eine automatische semantische Annotation von Visualisierungen zu entwickeln. Hierbei wurde darauf geachtet, dass sowohl Besonderheiten aus dem Bereich der Informationsvisualisierung als auch der wissenschaftliche Visualisierung mit in das Konzept einfließen. So entstand, nach der Durchführung der Recherche zu bereits existierenden Visualisierungs-Frameworks (siehe Kapitel 3), das Konzept einer mit semantischen Informationen angereicherten Visualisierungs-Pipeline.
Auf Grundlage dieses Konzepts soll zukünftig unter anderem die Durchführung von Eye-Tracking Studien zur Evaluierung von Visualisierungstechniken vereinfacht werden. Die durch einen Eye-Tracker aufgezeichneten Sakkaden und Fixationen von einzelnen Probanden lassen sich mit Techniken wie beispielsweise Gaze Plot oder Heatmap Visualisierungen zwar auf geeignete Weise darstellen, jedoch muss die darauffolgende Auswertung der Studien zum größten Teil von Hand erfolgen. Werden annotierte Visualisierungen verwendet, so ist es möglich zu gegebenen Fixationspunkten automatisch die semantische Bedeutung zu berechnen. Dies bedeutet, dass, anstatt lediglich Pixelkoordinaten von Rastergrafiken auszugeben, die Reihenfolge der betrachteten Visualisierungs-Elemente sowie weitere Metainformationen gespeichert werden können.
Um aus diesen Annotationsdaten maschinell verwertbare Aussagen ziehen zu können, wurde im Konzept sowie in der Implementierung außerdem gezeigt, wie semantische Informationen über den Aufbau von Visualisierungenstechniken erstellt werden können. Die einzelnen Bestandteile von Visualisierungen und deren Anordnung werden hierfür mit Hilfe einer aus drei Ebenen bestehenden Visualisierungs-Ontologie formal beschrieben.
Der implementierte Prototyp demonstriert, wie sich in nur wenigen Schritten Visualisierungen mit semantischen Attributen annotieren und daraufhin rendern lassen. Für die Implementierung wurde das Visualisierungs-Framework D3 als Grundlage verwendet, da

sich dieses aufgrund seines modularen Aufbaus und des SVG Ausgabeformats als geeignet herausstellt hatte.

7.2 Ausblick

Das Lösungskonzept wurde mit besonderem Fokus auf zweidimensionale Visualisierungstechniken entwickelt. Zwar wurden einige der in der wissenschaftlichen Visualisierung auftretenden Besonderheiten in das Konzept aufgenommen, jedoch bleiben Fragestellungen, wie beispielsweise die Umsetzung einer Annotation von Volumen- und Strömungsdaten, weitgehend unbeantwortet. Auch sollte in diesem Zusammenhang genauer untersucht werden, inwiefern es möglich ist semantische Annotation in das umfangreiche Framework VTK zu integrieren.
Zum Zeitpunkt der Erstellung dieser Studienarbeit existieren außerdem nur wenige Arbeiten, die sich mit der Entwicklung von Ontologien über Visualisierungen beschäftigen. In Kapitel 5 wurden einige Ansätze hierfür vorgestellt. Allerdings befinden sich darunter kaum Konzepte oder Implementierungen, die sich mit der Kombination und Anordnung von Visualisierungs-Elementen befassen. Daher wäre es nützlich, eine umfangreiche Ontologie zu gestalten, welche sowohl das Gebiet der Informationsvisualisierung als auch der wissenschaftlichen Visualisierung abdeckt.
Im Rahmen der Implementierung wurde beschrieben, wie mit Hilfe von D3 zu einem gegebenen Fixationspunkt die entsprechenden Visualisierungs-Elemente mit ihren semantische Informationen gefunden werden können. Da zu einem Fixationspunkt jedoch beliebig viele Visualisierungs-Elemente gehören können, ist es nötig die Menge der möglichen betrachteten Visualisierungs-Elemente einzuschränken. Die im Lösungskonzept vorgeschlagen Methoden können dabei für eine erste Implementierung dieses Vorgangs herangezogen werden. Die Spezifikation von RDFa [20] erläutert außerdem, wie es möglich ist aus den semantischen Attributen äquivalente OWL Tripel zu generieren, damit auf diesen z. B. Reasoning-Prozesse durchgeführt werden können. Zudem wäre es interessant dabei zu erforschen, welche Zeitersparnis sich bei der Auswertung von Eye-Tracking Studien durch die Annotation ergibt und wie weit dabei die Auswertung möglicherweise automatisiert werden kann.
Wichtig werden hierfür auch geeignete Darstellungsmethoden für Ontologien. Diese sollten nicht allein mit dem Ziel entwickelt werden, eine möglichst hohe Skalierbarkeit bezüglich der Anzahl der Klassen und Relationen zu erreichen, sondern vor allem dynamische Markierungen einzelner Instanzen in geeigneter Art und Weise darstellen können. Eine von Neupert, Eikmeier und Plohmer durchgeführte Fachstudie diskutiert hierbei mögliche Ansätze, die sich zur Visualisierung von dynamischen Ontologien eignen. Hierbei wurden bereits existierende Visualisierungen für Eye-Tracking sowie für Ontologien untersucht und aus diesen Teillösungen ein Lösungskonzept entwickelt [18].

Diese Studienarbeit stellt grundlegende Herangehensweisen zur Annotation von Visualisierungen mit semantischen Informationen dar. Die hier vorgestellten Konzepte können in Zukunft weiter ausgebaut und für speziellere Anwendungen entsprechend angepasst werden.

Literaturverzeichnis

[1] Antoniou, Grigoris ; vanHarmelen, Frank: *A Semantic Web Primer*. Cambridge, MA, USA : MIT Press, 2004. – ISBN 0262012103

[2] Attene, Marco ; Robbiano, Francesco ; Patanè, Giuseppe ; Mortara, Michela ; Spagnuolo, Michela ; Falcidieno, Bianca: Semantic annotation of digital 3D objects. In: *SAMT (Posters and Demos)* (2007)

[3] Bederson, Benjamin B. ; Hollan, James D. ; Perlin, Ken ; Meyer, Jonatham ; Bacon, David ; Furnas, George: PAD++: A Zoomable Graphical Sketchpad for Exploring Alternate Interface Physics. In: *Journal of Visual Languages and Computing* 7 (1995), S. 3–31

[4] Bostock, Michael ; Heer, Jeffrey: Protovis: A Graphical Toolkit for Visualization. In: *IEEE Transactions on Visualization and Computer Graphics* 15 (2009), November, Nr. 6, S. 1121–1128. – ISSN 1077–2626

[5] Bostock, Michael ; Ogievetsky, Vadim ; Heer, Jeffrey: D3: Data-Driven Documents. In: *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)* (2011). `http://vis.stanford.edu/papers/d3`

[6] Brodlie, Ken ; Noor, Nurul M.: Visualization Notations, Models and Taxonomies. In: *EG UK Theory and Practice of Computer Graphics* (2007)

[7] Carroll, Jeremy J. ; Klyne, Graham: Resource Description Framework (RDF): Concepts and Abstract Syntax / W3C. 2004. – W3C Recommendation. – http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

[8] Engelhardt, Stephan: *Modellierung kognitiver Prozesse in der Visualisierung*, Universität Stuttgart, Diplomarbeit, 2013

[9] Fekete, Jean-Daniel: The InfoVis Toolkit. In: *Proceedings of the IEEE Symposium on Information Visualization*. Washington, DC, USA : IEEE Computer Society, 2004 (INFOVIS '04). – ISBN 0–7803–8779–3, S. 167–174

[10] Ferraiolo, Jon ; Jackson, Dean ; Fujisawa, Jun: Scalable Vector Graphics (SVG) 1.1 Specification / W3C. 2003. – W3C Recommendation. – http://www.w3.org/TR/2003/REC-SVG11-20030114/

[11] Fruchterman, Thomas M. J. ; Reingold, Edward M.: Graph Drawing by Force-directed Placement. In: *Softw., Pract. Exper.* 21 (1991), Nr. 11, S. 1129–1164

[12] Ghoniem, Mohammad ; Fekete, Jean-Daniel ; Castagliola, Philippe: A Comparison of the Readability of Graphs Using Node-Link and Matrix-Based Representations. In: *Proceedings of the IEEE Symposium on Information Visualization*. Washington, DC, USA : IEEE Computer Society, 2004 (INFOVIS '04). – ISBN 0–7803–8779–3, S. 17–24

[13] Gruber, Thomas R.: A translation approach to portable ontology specifications. In: *Knowl. Acquis.* 5 (1993), Juni, Nr. 2, S. 199–220. – ISSN 1042–8143

[14] Heer, Jeffrey ; Card, Stuart K. ; Landay, James: Prefuse: A Toolkit for Interactive Information Visualization. In: *ACM Human Factors in Computing Systems (CHI)*, 2005, 421–430

[15] Kamada, T. ; Kawai, S.: An algorithm for drawing general undirected graphs. In: *Inf. Process. Lett.* 31 (1989), April, Nr. 1, S. 7–15. – ISSN 0020–0190

[16] Kitware, Inc ; Avila, L.S.: *The VTK User's Guide*. Kitware, Incorporated, 2012. – ISBN 9781930934238

[17] Minard, Charles J.: *Carte figurative des pertes successives en hommes de l'Armée Française dans la campagne de Russie 1812-1813*. `http://www.edwardtufte.com/tufte/minard`

[18] Neupert, Andreas ; Eikmeier, Tobias ; Plohmer, Sven: *Dynamische Ontologie Visualisierung*, Universität Stuttgart, Diplomarbeit, 2013

[19] Nicol, Gavin ; Champion, Mike ; Hégaret, Philippe L. ; Robie, Jonathan ; Wood, Lauren ; Hors, Arnaud L. ; Byrne, Steve: Document Object Model (DOM) Level 3 Core Specification / W3C. 2004. – W3C Recommendation. – http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407

[20] Pemberton, Steven ; Adida, Ben ; McCarron, Shane ; Birbeck, Mark: RDFa in XHTML: Syntax and Processing / W3C. 2008. – W3C Recommendation. – http://www.w3.org/TR/2008/REC-rdfa-syntax-20081014

[21] *Kapitel* Cognitive Ergonomics in Visualization. In: Raschke, Michael ; Blascheck, Tanja ; Ertl, Thomas: *Building Bridges between HCI and Visualization*. 2013. – (accepted)

[22] Raschke, Michael ; Engelhardt, Stephan ; Ertl, Thomas: A Framework for Simulating Visual Search Strategies, 2013. – (accepted)

[23] Raschke, Michael ; Heim, Philipp ; Ertl, Thomas: Interaktive verständnisorientierte Optimierung von semantisch-annotierten Visualisierungen. In: *INFORMATIK 2011: Informatik schafft Communities; 41. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*, Bonner Köllen Verlag, 2011 (Lecture Notes in Informatics (LNI))

[24] Santos, Selan dos ; Brodlie, Ken: Gaining understanding of multivariate and multidimensional data through visualization. In: *Computers & Graphics* 28 (2004), Nr. 3, S. 311–325

[25] Schreiber, Guus ; Dean, Mike: OWL Web Ontology Language Reference / W3C. 2004. – W3C Recommendation. – http://www.w3.org/TR/2004/REC-owl-ref-20040210/

[26] Seneviratne, Lasantha ; Izquierdo, Ebroul: *Image Annotation Through Gaming*

[27] Shi, Lei ; Gu, Guochang ; Liu, Haibo ; Shen, Jing ; Shi, Lei: A Semantic Annotation Algorithm Based on Image Regional Object Ontology. In: *Proceedings of the 2008 International Conference on Computer Science and Software Engineering - Volume 04*. Washington, DC, USA : IEEE Computer Society, 2008 (CSSE '08). – ISBN 978–0–7695–3336–0, S. 540–543

[28] Symonova, Olga ; Dao, Minh-Son ; Ucelli, Giuliana ; De Amicis, Raffaele: Ontology Based Shape Annotation and Retrieval. In: *Proc. of the ECAI*, 2006

[29] Tobii Technology (Hrsg.): *Tobii T60 XL Eye Tracker Reference Manual*. Second. Tobii Technology, 2011. `http://www.tobii.com`

[30] Voigt, M. ; Polowinski, J.: *Towards a Unifying Visualization Ontology*. Techn.Univ., Fakultät Informatik, 2011 (Technische Berichte)

[31] Wehrend, Stephen ; Lewis, Clayton: A problem-oriented classification of visualization techniques. In: *Proceedings of the 1st conference on Visualization '90*. Los Alamitos, CA, USA : IEEE Computer Society Press, 1990 (VIS '90). – ISBN 0–8186–2083–8, 139–143

Alle URLs wurden zuletzt am 16.06.2013 geprüft.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Stefan Strohmaier)