Bakkalaureatsarbeit

Entwicklung einer Visualisierung für Time Series Data

Philipp Weißensteiner

21. Februar 2014

Betreuer: Dr. Peter Kraker
Begutachterin: Prof. Dr. Stefanie Lindstaedt

Technische Universität Graz
Knowledge Technologies Institute

Kurzfassung

Diese Arbeit befasst sich mit der Erweiterung einer bestehenden Webapplikation (*Headstart* (Kraker et al., 2013)) um eine Visualisierung für Zeitreihen (Time Series Data). Visualisierungen ermöglichen es, komplexe Sachverhalte in besser verständliche Formen zu bringen und diese einfacher zu interpretieren. Eine besonders interessante Art von Informationen sind Zeitreihen. Diese häufig auftretende Form von Daten bietet sich dazu an, um Trends und Muster zu erkennen und Aussagen über zukünftige Entwicklungen zu machen.

Als Proof of Concept wird eine Visualisierung entwickelt, welche es den Anwendern von *Headstart* ermöglicht, Trends und Entwicklungen in Forschungsgebieten auszumachen. Um die Erweiterung in dem bestehenden Projekt zu bewerkstelligen, muss dieses erst um einen Statusverwaltungsmechanismus bereichert werden. Dessen Implementation bildet den einen Teil dieser Arbeit, während sich der zweite Teil der neuen Visualisierung widmet.

Der Einbau der Statusverwaltung wurde mit Hilfe von vorgestellten Metriken aufgezeichnet und führte zu einer klaren Verbesserung des Projektes. Somit sind zukünftige Erweiterungen mit deutlich weniger Aufwand verbunden. Die Visualisierung für Zeitreihen durch Small Multiples bleibt der ursprünglichen Oberfläche treu und ermöglicht den Benutzern, einfach Vergleiche zwischen Forschungsgebieten anzustellen.

Inhaltsverzeichnis

Abkürzungen

UI	User Interface / Benutzeroberfläche
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
SLOC	Source Lines of Code

1 Einleitung

Die Menge an elektronischen Daten hat in den letzten Jahren durch eine unglaubliche Anzahl von Aktivitäten und Vorgängen in der Wissenschaft, Politik oder im privaten Umfeld einen enormen Anstieg erfahren. Um diese Informationen sinnvoll verarbeiten zu können, bedient man sich unter anderem verschiedener Visualisierungsmethoden (Kohlhammer et al., 2011).

Visualisierungen ermöglichen es uns, komplexe Sachverhalte und Zusammenhänge in besser verständliche Formen zu bringen und letztendlich Einsichten, Interpretationen und Erkenntnisse zu erlangen (Chen, 2010).

Eine Kategorie von Informationen mit denen wir jeden Tag konfrontiert werden nennt sich *Time Series Data* (Zeitreihen). Darunter verstehen wir einen Satz von Werten, welche sich über einen bestimmten Zeitraum immer wieder verändern. Beispiele dafür wären unter anderem Temperaturveränderungen, Aktienkurse oder die Leserzahl von diversen Publikationen. Zeitreihen eignen sich hervorragend dazu Trends und Muster zu erkennen und ist eine wichtige und weit verbreitete Basis für eine Vielzahl von Visualisierungsmethoden (Heer et al., 2010).

1.1 Problemstellung

In vielen Forschungsgebieten geht es darum Veränderungen, Trends und Muster zu erkennen und zu interpretieren. Diese Informationen können zum Beispiel bei der Problemauswahl hilfreich sein: *Is the problem feasible? What increase in knowledge is expected from the project?* (Alon, 2009). Natürlich lassen sich bereits Aussagen über das Wachstum oder die Publikationsraten von Forschungsbereichen machen (Kharabaf & Abdollahi., 2012), allerdings ist die manuelle Aufbereitung solcher Informationen ein mühsames Unterfangen.

Daher besteht die Aufgabe dieser Arbeit darin, eine Visualisierung zu entwerfen die es Forschern ermöglicht, die eben erwähnten Aussagen mit möglichst geringem Aufwand treffen zu können. Der Prototyp soll die visuelle Wahrnehmung des Anwenders unterstützen, indem eine auf das Problem zugeschnittene Visualisierung von Zeitreihen entwickelt wird. Unter Berücksichtigung von Problemen wie Change-Blindness, wird das Projekt Headstart (Kraker et al., 2013) entsprechend erweitert und soll es den Anwendern erlauben, möglichst einfach Erkenntnisse in Forschungsbereiche und

deren Entwicklung über einen bestimmtem Zeitraum zu erlangen.

1.2 Zielsetzung

Als Proof of Concept soll Headstart um eine Zeitkomponente erweitert werden. Da sich das Projekt noch in der Entwicklungsphase befindet, fehlt es im Augenblick an einem Verwaltungsmechanismus über die unterschiedlichen Status der Benutzeroberfläche. Er ist allerdings Voraussetzung für eine Weiterentwicklung des Projekts und bildet gleichzeitig eine Basis für zukünftige Erweiterungen der Benutzeroberfläche.

Als erstes Ziel ergibt sich somit die Implementation eines Statusverwaltungsmechanismus für die Benutzeroberfläche. Um dies zu bewerkstelligen sollen verschiedene Implementationsmethoden abgeschätzt werden. Weiters sollen die Veränderungen/Verbesserungen am bestehenden Projekt festgehalten, verglichen und beurteilt werden.

Das Hauptziel ist schließlich die Erweiterung von Headstart um eine Zeitkomponente. Sie soll es den Anwendern ermöglichen, Veränderungen in Forschungsgebieten auszumachen. Um ein zufriedenstellendes Ergebnis zu erhalten, gilt es die unterschiedlichen Visualisierungsmethoden von Time Series Data vorzustellen, um eine nachvollziehbare Entscheidung für die Visualisierungsmethode treffen zu können.

1.3 Überblick über die Arbeit

Die Arbeit besteht aus einem theoretischen und einem praktischen Teil.

Zuerst werden die vorhandenen, theoretischen Konzepte und Möglichkeiten im Kapitel Theorie recherchiert. Der aktuelle Zustand der Applikation, und wie diese sich von anderen traditionellen Webapplikationen unterscheidet, werden angesprochen. Es werden hilfreiche Metriken vorgestellt, die in weiterer Folge der Messung von Veränderungen am Projekt dienen. Im theoretischen Teil werden neben der Statusverwaltung auch verschiedene Visualisierungen für Zeitreihen vorgestellt und auf das Problem von Change-Blindness hingewiesen.

Der praktische Teil (Praxis) widmet sich schließlich der Umsetzung des Projektes. Die Benutzeroberfläche wird analysiert, modelliert und umgeschrieben und schließlich mit den vorgestellten Metriken beurteilt. Abschließend wird die Implementation der Zeitkomponente besprochen und deren Stärken und Schwächen gegenüber anderen Methoden abgeschätzt.

2 Theorie

2.1 Einleitung

Es folgt eine kurze Beschreibung von traditionellen Webapplikationen und deren Architekturen. Anschließend wird die bestehende Visualisierung vorgestellt und erläutert in wie fern sich diese sich von üblichen Webapplikationen unterscheidet. Hierbei soll deutlich werden, dass der Schwerpunkt der Applikationslogik in der Präsentationsebene liegt und es gilt, diese mit einer guten Statusverwaltung nachvollziehbar zu abstrahieren. In Abschnitt 2.4 werden zwei Metriken vorgestellt um Veränderungen am bestehenden Projekt besser einschätzen zu können. Anschließend werden verschieden Visualisierungen für Zeitreihen (Abschnitt 2.5) und das Problem der Change Blindness besprochen.

2.2 Webapplikationen und deren Benutzeroberflächen

2.2.1 3-Tier Architekturen

Viele Webapplikation folgen einer Client-Server Architektur und sind in verschiedene Tiers aufgeteilt *(verschiedene Variationen möglich, Two, Three- bzw. N-Tier Architecture)*. Am meisten verbreitet ist dabei die 3-Tier Architektur. Sie besteht aus einem *Presentation, Business* und *Data* Layer (siehe Abbildung 1).

Der Vorteil einer solchen Client-Server Architektur besteht darin, dass durch die klare Trennung der Layer jeder von ihnen potentiell auf einer anderen Maschine laufen kann. In der Praxis laufen Business und Data Layer meist auf einem Webserver. Am Server werden Anfragen eines Clients *(Browser)* entgegengenommen. Der Business Layer entscheidet dann auf welche Daten im Data Layer zugegriffen werden muss um schließlich eine HTML Seite zu rendern und diese dem Client als Antwort zurückzuschicken. (Papagelis, 2012)

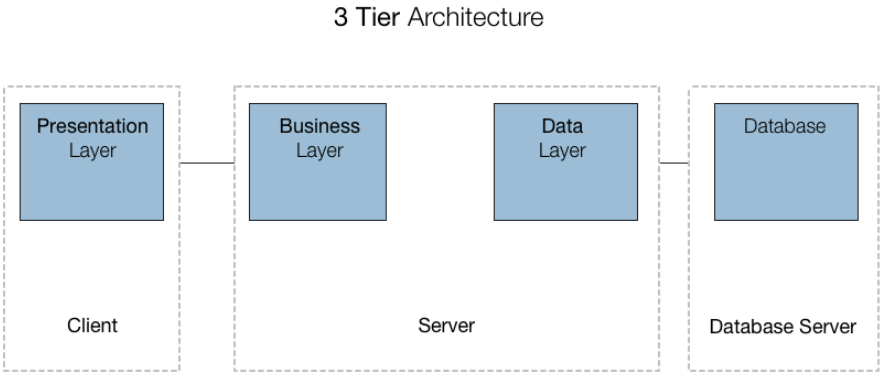


Abbildung 1: 3-Tier Architektur (nach Manos Papagelis, 2013)

2.2.2 Headstart als Beispiel

Headstart ist eine moderne Webapplikation deren Schwerpunkt in der Benutzeroberfläche (am Client) liegt. Um besser zu verstehen in wie fern sich diese von traditionellen Architekturen unterscheiden folgt ein kurzer Überblick des Projektes.

Die Visualisierung bezieht die verwendeten Leserschafts- und Publikationsinformationen aus dem Referenz Management Tool Mendeley[1]. Aus diesen Daten werden durch Multidimensionale Skalierung Positionsvektoren ermittelt. Anschließend wird mittels Hierarchical Agglomerative Clustering die Gruppierung der Publikationen erzeugt bzw. die in der Benutzeroberfläche angezeigten “Blasen”. Die finale Anordnung der Publikationen wird schließlich durch ein Force Directed Layout und einen Kollisionsermittlungsalgorithmus bestimmt.[2]

Vergleicht man das in 2.2 vorgestellte Modell mit Headstart, unterscheidet sich die Applikation dadurch, dass sie nur sehr dünne Data und Businesslayer hat und es keine Datenbank gibt. Der Schwerpunkt der Applikation liegt in der Präsentationsebene (UI). Somit ist es deutlich leichter, auch alle übrigen Layer auf dem Client laufen zu lassen. Oder mit anderen Worten: Nachdem Headstart HTML, CSS und Javascript vom Webserver erhält, läuft die eigentliche (Logik) der Applikation ausschließlich im Browser. Ob sich dieser Sachverhalt in Zukunft ändert steht für den Moment noch nicht fest. Diese Möglichkeit soll aber berücksichtigt werden, d.h. der Data Layer soll wie auch in 3-Tier Applikationen möglichst unabhängig von den restlichen Layern bestehen können und so mit geringem Aufwand auszutauschen sein (falls dieser in der Zukunft auf einem Webserver anzufinden ist).

[1] http://mendeley.com

[2] Eine genauere Beschreibung der angewandten Algorithmen ist in Kraker et al. (2013) anzufinden. Diese sind für den Einbau der Statusverwaltung allerdings nicht weiter relevant.

Der für Headstart wichtigste (und interessanteste) Layer ist somit der Presentation Layer.

2.2.3 Presentation Layer

Javascript wird mittlerweile auf rund 89,2 % aller Webseiten W3TECHs (2013) verwendet und bildet durch die enormen Leistungssteigerungen von Browsern und Client in vielen modernen Applikationen bereits den Hauptbestandteil der Benutzeroberfläche (Elliot, 2013). Die erwähnten Performancesteigerungen ermöglichen es, anspruchsvolle und interaktive Benutzeroberflächen für den Client zu entwickeln. Ähnlich wie in Headstart wird nur noch wenig HTML vom Server empfangen und der Großteil der Seite bzw. des UI Aufbaus vom Client übernommen.

Bestandteile der Präsentationsebene (Browser):

1. Hypertext Markup Language
2. Javascript
3. Cascading Style Sheets

Das UI der Applikation wird überwiegend durch Interaktionen mit der Maus gesteuert, und ist wie folgt aufgebaut: es gibt eine einzige HTML Datei mit ein paar grundlegenden HTML Knoten. Durch sie werden dann die Javascript Dateien eingebunden welche weiters den Hauptteil des UI Aufbaus übernehmen. Visuelle Elemente wie Links oder Buttons bilden die primären Navigationsbestandteile. Diese Elemente sind Knoten im HTML Dokument wie beispielsweise <*div*> oder <*button*> die durch Javascript erstellt, manipuliert und schließlich durch ihre Attribute *id* oder *class* mit CSS gestaltet werden.

2.3 Zustandsmaschienen im Presentation Layer

»(Ein Status) modelliert eine Situation in jener eine (implizite) unveränderliche Bedingung gilt.« [1]

Zustandsmaschinen sind einfache Modelle, die uns im täglichen Leben ständig begegnen. In Digitale Uhren, Fernbedienungen und vielen anderen Gegenständen findet das Konzept der Statusmaschine Verwendung.

Zustandsmaschinen sind Systeme bestehend aus einer Menge von unterschiedlichen Status. Einer dieser Status ist der *Anfangsstatus*, einer oder mehrere der anderen Status wird als *Endstatus*

[1] OMG Unified Modeling Language - http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/

bezeichnet. Zwischen Status wird durch Transitions (Actions, Übergänge) gewechselt. Transitions werden mit bestimmten Events assoziiert und durch diese ausgelöst. Wenn ein Event zum Endstatus führt, wird die Statusmaschiene beendet. Diese Modelle werden oft als State-transition diagrams dargestellt (Thomas & Hunt, 2002).

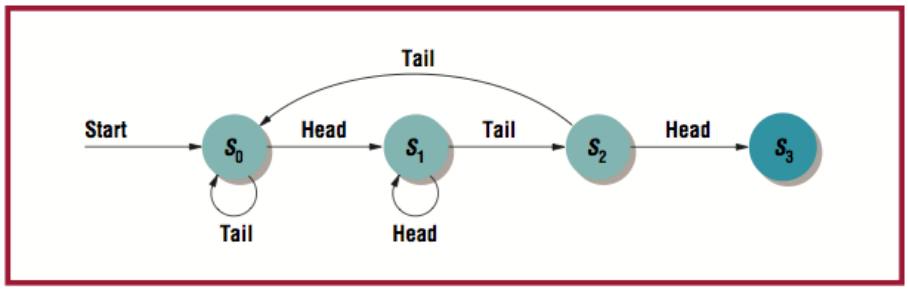


Abbildung 2: Einfache Zustandsmaschine für Münzwurf (Kopf-Zahl-Kopf) (Thomas & Hunt, 2002)

Zustandsmaschinen eignen sich gut für die Entwicklung von grafischen Benutzeroberflächen und werden für deren Ausarbeitung häufig verwendet (Samek, 2009). Die Präsentationsebene soll möglichst übersichtlich die einzelnen UI Elemente, und deren Verhalten auf Benutzereingaben widerspiegeln.

Zustandsmaschinen haben nach Bitter & Rick (2001) drei wichtige Konzepte: *»State machines revolve around three concepts: the state, the event, and the action. States describe the status of a piece of programming and are subject to change over time. . . . Events are occurrences in time that have significant meaning to the piece of code controlled by the state machine. . . . Actions are responses to events, which may or may not impact external code to the state machine.«*

Wichtig ist nach Bitter & Rick (2001) auch *» The state machine itself always makes state changes. The current state is not normally given to code external to the state machine. Under no circumstances should external code be allowed to change the current state. The only information external code should give to the state machine is an event that has occurred. Changing state and dictating actions to perform is the responsibility of the state machine. «*

Die vorhandene Benutzeroberfläche von Headstart und deren Verhalten, decken sich bereits mit diesen Definitionen. Es fehlt einfach an einem klaren Mapping dieses Verhaltens auf Statusmaschinen (*Status, Events* und *Actions*) welches sich in weiterer Folge natürlich auch im Quellcode widerspiegeln soll.

2.4 Komplexität und Schwierigkeiten bei der Erweiterbarkeit

Viele Applikationen werden im Verlauf ihrer Entwicklung unübersichtlich und schwierig zu warten. Headstart ist ein “Work-in-Progress” und befindet sich in vor einem großen um bzw. Ausbau. Das Projekt ist nicht unwartbar, Erweiterungen und Anpassungen am aktuellen Projekt sind allerdings nur schwer durchführbar. Es soll vor allem im Hinblick auf den zweiten Teil dieser Arbeit, den Einbau einer Zeitkomponente, dieser Umstand verbessert werden. Um die Veränderungen (und hoffentlich Verbesserungen) in Bezug auf die Wart- und Erweiterbarkeit von Headstart wenigstens ansatzweise zu messen und verfolgen, sollen Metriken verwendet werden. Für die Messungen in Kapitel 3.4 wurde die Plato[1] Bibliothek verwendet.

2.4.1 Source Lines of Code

Source Lines of Code (SLOC) ist eine vergleichsweise einfache Metrik um den Umfang von Programmen abzuschätzen, bei der schlichtweg die Programmzeilen gezählt werden. Allerdings ist SLOC trotzdem ein wichtiger und nützlicher Indikator.

*»SLOC (SLOC or LOC) is one of the most widely used sizing metrics in industry and literature. . . . Size is one of the most important attributes of a software product. It is not only the key indicator of software cost and time but also a base unit to derive other metrics for project status and software quality measurement. «*Nguyen et al. (2007)

Die Metrik soll für diese Arbeit eingesetzt werden, um zu vergleichen, wie sich der Programmcode nach dem Umbau verändert bzw. verteilt hat.

2.4.2 Halstead complexity measures

Eine von Maurice Howard Halstead im 1977 eingeführte Metrik kann ebenfalls zur Abschätzung der Komplexität dienen. Sie wird durch die Anzahl von Operatoren und Operanden (*in weiterer Folge Operators bzw. Operands*) berechnet und kann als Indiz für die Schwierigkeit von Programmen betrachtet werden (Serebrenik, 2011).

$$Difficulty = \frac{n1}{2} * \frac{N2}{n2}$$

[1] https://github.com/es-analysis/plato

Operators	Vorkommen	**Operators**	Vorkommen	**Operands**	Vorkommen
<	3	{	3	0	1
=	5	}	3	1	2
>	1	+	1	2	1
-	1	++	2	a	6
,	2	for	2	i	8
;	2	if	2	j	7
(	4	int	1	n	3
)	4	return	1	t	3
[]	6				

Tabelle 1: Anzahl Operators und Operands für Beispiel

n1	unique Operators
N2	total Operands
n2	unique Operands
N1	*total Operators*

Es folgt ein kurzes Beispiel (Serebrenik, 2011):

```
void sort ( int*a, int n ) {
  int i, j, t;
  if ( n < 2 ) return;
  for( i=0 ; i < n-1; i++ ) {
    for( j=i+1 ; j < n ; j++ ) {
      if ( a[i] > a[j] ) {
        t = a[i];
        a[i] = a[j];
        a[j] = t;
      }
    }
  }
}
```

1. Zuerst werden Operators und Operands gezählt (Tabelle 1):

2. Es ergeben sich die Werte:
   N1 = 50, N2 = 30, n1 = 17, n2 = 7

3. Und schließlich für die Difficulty:
   $D = \frac{17}{2} * \frac{30}{7} \approx 36$

Dateiname	Difficulty	SLOC
data.js	52	176
val.js	60	161
core.js	88	518
css.js	78	455
effects.js	108	640

Tabelle 2: Difficulty jQuery

Um diese Metrik besser einschätzen zu können, werden in den Tabellen 2, 3 die Werte[1] der Javascript Bibliotheken jQuery[2] und Grunt[3] aufgelistet. Die Difficulty wurde mit der Open Source Bibliothek Plato[4] berechnet. Die Ergebnisse liegen zwischen 36 und 108 und steigen (wie in der Abb 3 ersichtlich) in der Regel mit den SLOC's. Im Abschnitt 3.4.1 werden diese Ergebnisse als Vergleichsgrundlage verwendet.

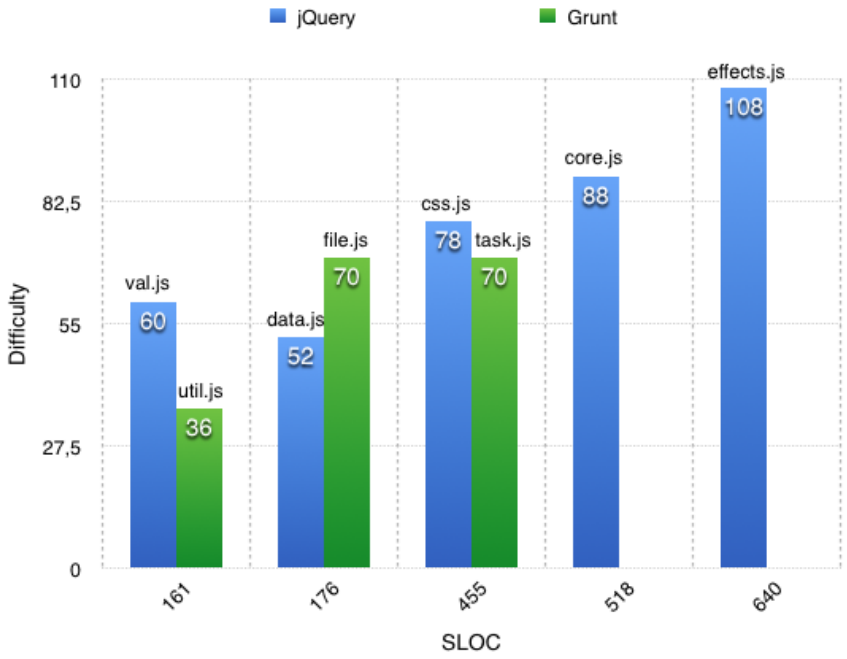


Abbildung 3: Difficulty und SLOC für Grunt und jQuery

[1] http://es-analysis.github.io/plato/examples/jquery/, http://es-analysis.github.io/plato/examples/grunt/
[2] http://jquery.com/
[3] http://gruntjs.com/
[4] https://github.com/es-analysis/plato

Dateiname	Difficulty	SLOC
file.js	70	429
task.js	70	436
util.js	36	180

Tabelle 3: Difficulty Grunt

2.5 Visualisierungen für Time Series Data

Handelt es sich um Datensätze die zu unterschiedlichen Zeitpunkten aufgenommen wurden (Wetterdaten, Aktienpreise), spricht man von Time Series Data (Zeitreihen). Dies ist eine gut verstandene und häufig auftretende Problematik in der Visualisierung. Diese Problemstellung tritt beispielsweise auch auch im Finanzbereich bzw. am Aktienmarkt, bei Temperaturveränderungen, Kriminalitätsraten uvm. auf (Heer et al. 2010). Deshalb gibt es zu diesem Problem schon einige "Rezepte" die nun vorgestellt werden, um zu entscheiden, welches von ihnen die Domäne am besten abdeckt und in der Implementation dann das Umzusetzende ist.

2.5.1 Index Charts

Wenn die relative Veränderung der unterliegenden Daten am wichtigsten ist, wie es zum Beispiel oft im Finanz und Aktienmarkt der Fall ist (die Preisveränderung ist interessanter als der Preis selbst), eignet sich ein Index Chart gut für die Darstellung (siehe Abb 4). In diesem Beispiel reagiert die Visualisierung zudem noch auf Positionsveränderungen der Maus in x-Richtung (Zeit) und passt die Graphen entsprechend an (d.h. Bewegung der Maus nach links oder rechts bewirkt Veränderung der Kurven) (Heer et al., 2010).

2.5.2 Stacked Graphs

Andere Datensätze eignen sich dazu, übereinander gelegt zu werden. Stacked Graphs entstehen durch die Zusammenlegung von mehreren Area Charts. Diese visuelle Aufsummierung unterstützt oft die Betrachtung einzelner Bestandteile des Gesamtaggregates. Allerdings sind die Stacked Graphs auch eingeschränkt. So lassen sich keine negativen Zahlen (z.B. Temperaturen) summieren. Auch das Erkennen und Interpretieren von Veränderungen wird durch das Aufeinanderlegen erschwert (Abhilfe kann hier aber eine Suche oder ein Filtersystem schaffen).

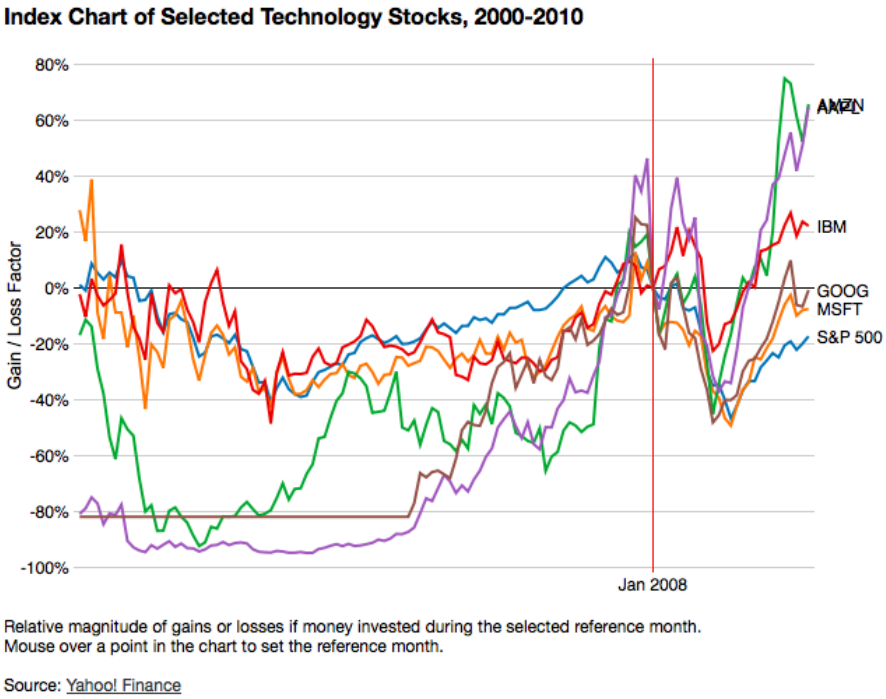


Abbildung 4: Index Chart http://hci.stanford.edu/jheer/files/zoo/ex/time/index-chart.html (J., 2010)

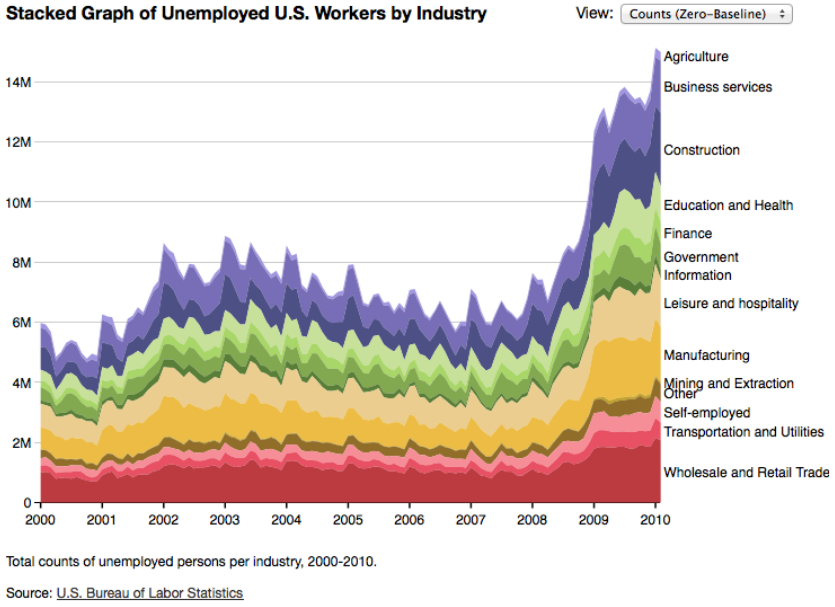


Abbildung 5: Stacked Graph http://hci.stanford.edu/jheer/files/zoo/ex/time/stack.html

2.5.3 Horizon Graphs

Wenn viele Zeitreihen auf einmal verglichen werden sollen, kann ein sogenannter Horizon Graph gute Ergebnisse liefern. Im Horizon Graph wird die Datendichte einer Time Series erhöht, während gleichzeitig die Auflösung beibehalten wird. Um das nachzuvollziehen wird nachfolgend kurz der Aufbau eines solchen Graphen beschrieben (siehe Abb 6) in welchem die nationale Arbeitslosenrate als Prozentsatz der Arbeiterschaft dargestellt wird.

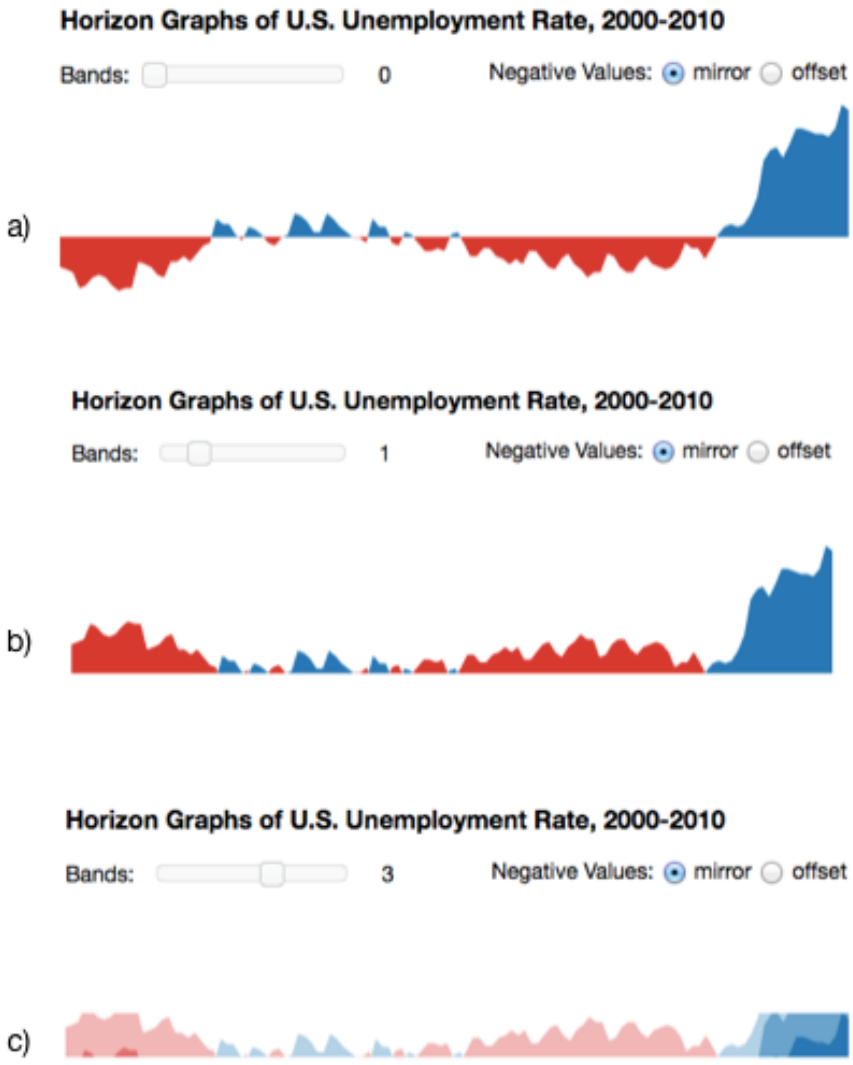


Abbildung 6: Horizon Graphs http://hci.stanford.edu/jheer/files/zoo/ex/time/horizon.html

Zuerst ein normaler Area Chart (a) positive Werte blau, negative Werte rot.

Im zweiten Graphen (b) werden die negative Werte in die gleiche Fläche der positiven Werte

gespiegelt (somit wird die Datendichte verdoppelt).

Der dritte Graph (c) is nun ein Horizon Graph. Noch einmal wird die Datendichte verdoppelt indem die Höhe des Graphen verringert wird. Die dadurch überschüssigen Spitzen werden nicht verworfen, sondern als neue Ebene (mit dunklerer Farbe) einfach über die bestehenden Ebenen gezeichnet. Das Endergebnis ist also ein Graph der mit nur einem Viertel der Größe die selbe Datendichte repräsentiert. Obschon etwas mehr Zeitaufwand benötigt wird, um sich mit dem Graphen vertraut zu machen, kann sich dieser als effektiver erweisen als ein Standard Plot (wenn die Größe schrumpft) (Heer et al., 2010).

2.5.4 Small Multiples

Die nächste Visualisierung nennt sich Small Multiples und wurde bereits (indirekt) für die obige Erklärung vom Horizon Graph verwendet. Bei Small Multiples werden anstatt wie bei Index Charts die Zeitreihen nicht in den selben Graphen, sondern jede Series in ihren eigenen Graphen gezeichnet. Das hat den Vorteil, dass es nicht zu ungünstigen Überschneidungen kommen kann. Small Multiples lassen sich fast aus jedem Visualisierungstyp erstellen (bar charts, pie charts) und liefern für die meisten dieser Fälle ein besseres Ergebnis als alle Graphen in einen einzigen zu stecken (Tufte, 1990).

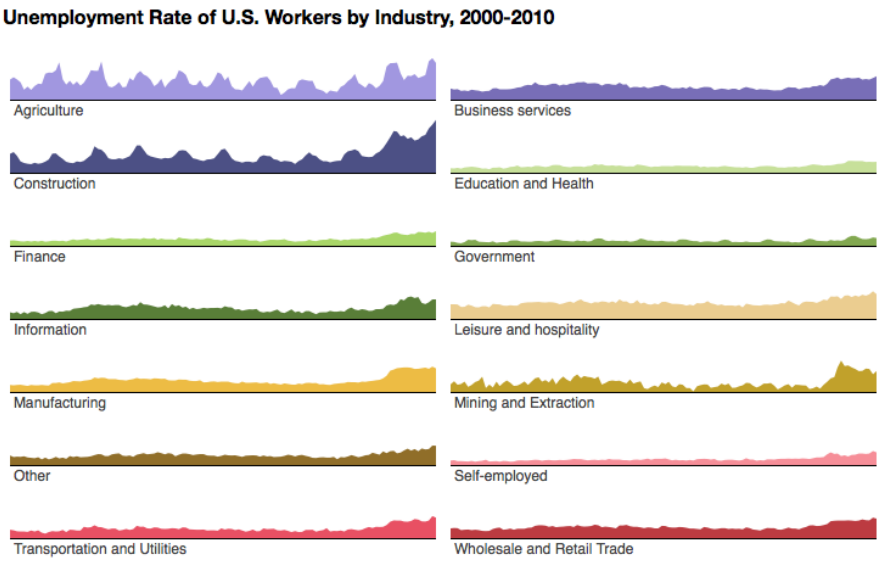


Abbildung 7: Small Multiples

»*"At the heart of quantitative reasoning is a single question: Compared to what?"*«(Tufte, 1990)

Small Multiple Designs haben die wunderbare Eigenschaft direkte visuelle Vergleiche von Formen, Unterschieden von Objekten zu provozieren. Sie eignen sich damit für eine große Anzahl von

Präsentationen und liefern oft die beste Designlösung. Weil die visuellen Bestandteile der Small Multiples “auf einen Blick” positioniert sind, lassen sich Vergleiche ohne jegliche Unterbrechungen machen. Einheitliches Design verdeutlicht Veränderungen in den Daten (und nicht dem Data-frame) (Tufte, 1990).

2.6 Change Blindness

Als Change Blindness bezeichnet man das menschliche Unvermögen, Veränderungen an einem Objekt oder einer Szene zu erkennen (Simons & Rensink, 2005).

Eine Vielzahl von Studien (Simons & A., 1997) widmen sich dem Phänomen Change Blindness auf unterschiedliche Weise. Um mögliche Ursachen besser zu verstehen, beschreibt Simons verschiedene Experimente, von denen hier zwei kurz erwähnt werden sollen.

2.6.1 Sakkaden-bedingte Veränderungen

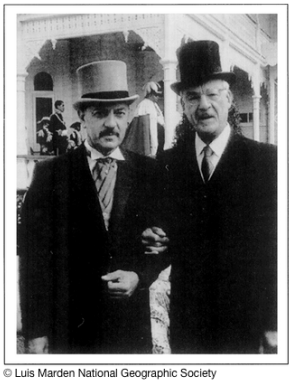
In diesem Versuch werden den Beobachtern eine Reihe von Bildern gezeigt. Während die Augen der Beobachter schnell von einem Objekt zum nächsten in der Szene wandern kann es zu Veränderungen in der Szene kommen. Überraschenderweise werden in diesem Versuch 70% der Veränderungen überhaupt nicht erkannt. 50% der Teilnehmer waren nicht einmal in der Lage zu erkennen das die Gesichter zweier Menschen vertauscht wurden (Grimes, 1996) (Siehe Abbildung 8).

a) Zuerst wird das Bild präsentiert.

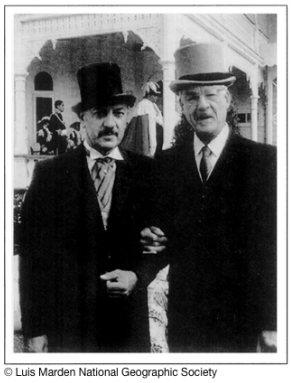
b) Anschließend wird, während die Testteilnehmer die Objekte im Bild *scannen*, etwas am Bild verändert (in diesem Fall wird der Hut vertauscht). Durch die Bewegung des Auges wird diese Veränderung aber nicht wahrgenommen.

2.6.2 Globale Transienten

Noch schlechtere Ergebnisse liefern sogenannte Flicker Paradigm Experimente. Hier wird zuerst das Original, dann eine leere Seite und schließlich die Abwandlung angezeigt. Im ersten Durchlauf dieser Versuche wurden überhaupt keine Veränderungen erkannt. Wird aber die leere Seite entfernt können nahezu alle Abwandlungen schnell erkannt werden.



a)



b)

Abbildung 8: Veränderungen werden durch Bewegungen der Augen übersehen(Grimes, 1996)

Daraus folgt: Menschen scheinen Schwierigkeiten zu haben Veränderungen verschiedenster Art auszumachen. Vor allem aber, führen Unterbrechungen zu einem verschlechterten Ergebnis.

3 Praxis

3.1 Einleitung

Im zweiten Teil dieser Arbeit sollen nun die im theoretischen Teil erwähnten Aspekte berücksichtigt und die Implementierung der Statusverwaltung und Zeitkomponente beschrieben werden. Es folgt eine Analyse der bestehende UI, um diese anschließend durch Zustandsmaschinen modellieren zu können. In Kapitel 3.4 wird dann die Vorgehensweise genauer beschrieben und die Auswirkungen des Umbaus anhand der vorgestellten Metriken gemessen.

In Kapitel 3.6 wird aus den vorgestellten Visualisierungsmethoden ein Design für die Zeitkomponente von Headstart entwickelt und vorgestellt.

3.2 Die Benutzeroberfläche von Headstart

Headstart wird im Augenblick über ein einziges Fenster im Browser bedient (siehe Abb 9). Alle Veränderungen in der Benutzeroberfläche werden durch Interaktionen mit den Navigationselementen und der Maus ausgelöst. Nun sollen diese Elemente kurz vorgestellt werden, um anschließend deren Statusverwaltung durch Zustandsmachschinen zu definieren.

3.2.1 Bubbles

Die Blasen (in weiterer Folge auch als *Bubbles* bezeichnet) sind Hauptbestandteil von Headstart und repräsentieren die Forschungsbereiche in welche die darunter sichtbaren Papers zugeordnet werden können. Die Blasenelemente reagieren auf *Mouseover/enter* und *Click Events*.

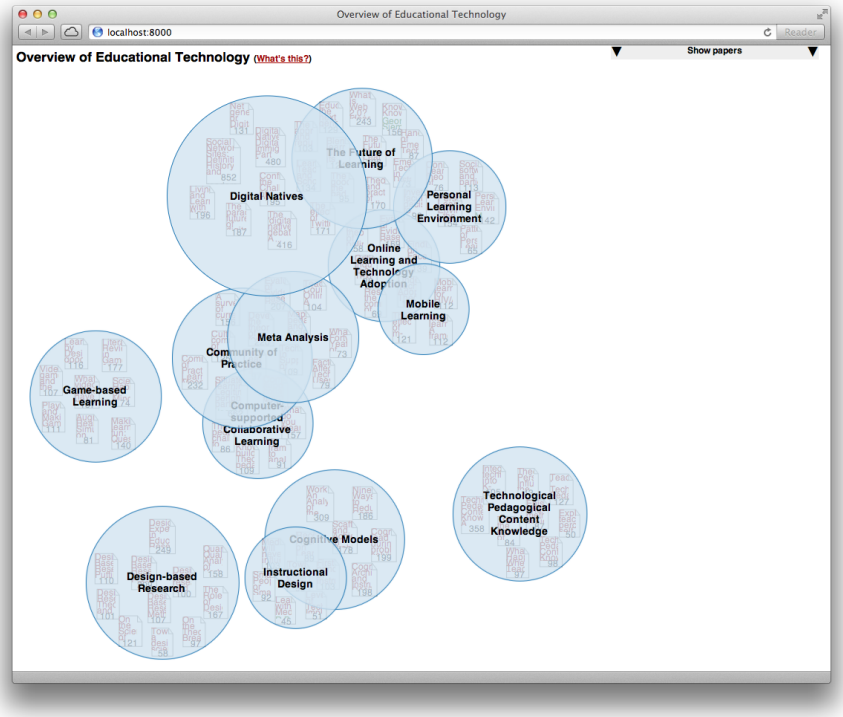


Abbildung 9: Headstart

3.2.2 Papers

In dieser Ansicht ist jedes Paper seinem Forschungsgebiet zugeordnet. Hier ist jeweils der Titel der Arbeit (rot) und die Anzahl seiner Leser sichtbar. Sie befinden sich anfangs “hinter” und innerhalb der Blasen und rücken erst durch ein *Mouseover* Event in den Vordergrund. Anzumerken ist, dass sich die Papers nur visuell innerhalb der Blasen befinden, sie sind im Canvas keine Kinderknoten der Bubbles.

3.2.3 Liste

Die Liste beinhaltet aller verfügbaren Papers und zeigt für jedes von ihnen einen Titel, Autoren, einen Abstract, die Anzahl der Leser und das Publikationsorgan. Die Liste kann zur Navigation verwendet werden. *Klickt* der Anwender auf eines der Papers, wird die dazugehörige Bubble (das Forschungsgebiet) vergrößert. Eine Publikation in der Liste wird auch angezeigt, wenn der Benutzer das selbe Paper in einer der Bubbles auswählt.

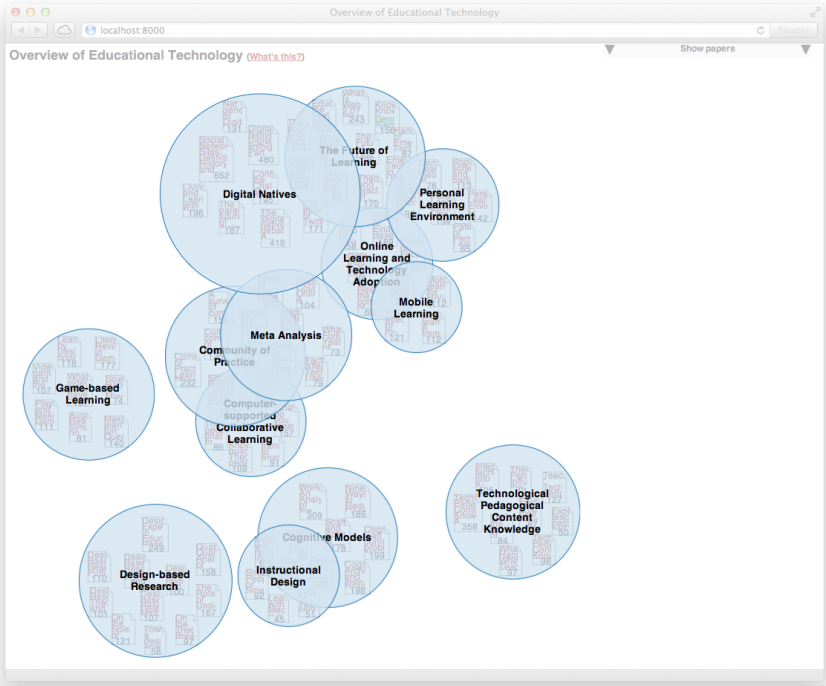


Abbildung 10: Bubbles - Forschungsgebiete als Blasen dargestellt

3.2.4 Header

Der Header Text passt sich der aktuell gewählten Forschungsbereich an. Wird auf eine der Blasen *geklickt*, ändert sich der Text zu deren Forschungsgebiet.

3.2.5 Popup

Das Popup-Element beinhaltet eine kurze Beschreibung von Headstart und wird über den “Whats this” link im Header angezeigt. Zudem können im Popup Paper-Previews angezeigt werden.

3.3 Statusdefinition

Jetzt werden die vorgestellten Komponenten der Benutzeroberfläche als Statusmaschinen modelliert, um anschließend die Umsetzung zu erleichtern.

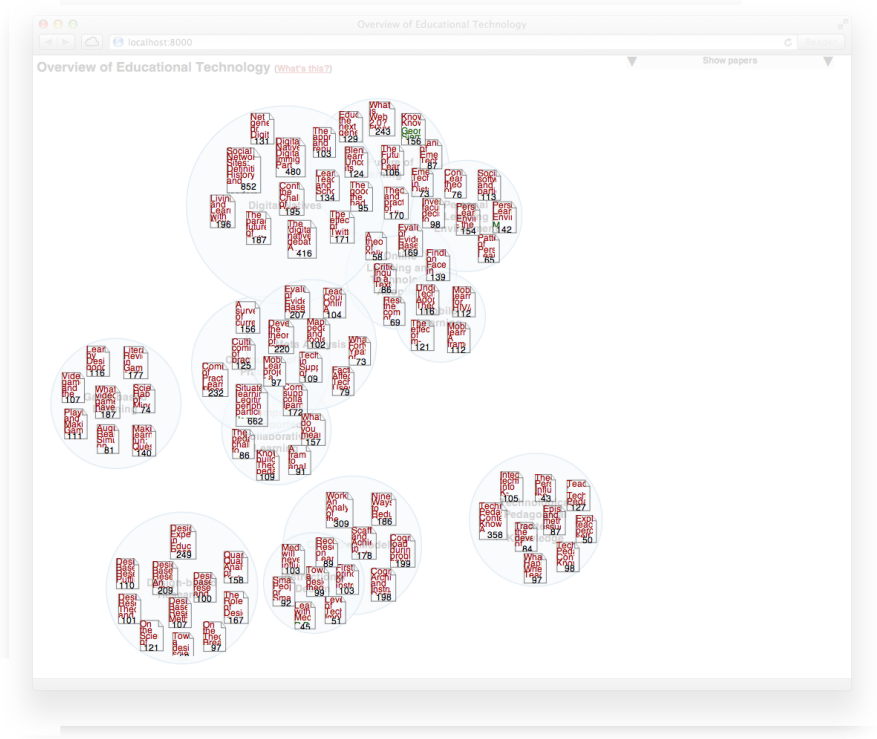


Abbildung 11: Papers - Die Publikationen nach FDA in Bubbles

3.3.1 Bubbles

Jede Bubble befindet sich nach dem Start im Zustand *ZoomedOut* in der normalen Ansicht (siehe Abb 9). Bubbles verändern ihre Darstellung wenn mit der Maus über eine von ihnen *gehovered* wird. Dann werden die darunterliegenden Papers deutlicher sichtbar. Dieser Zustand wird als *Hover* bezeichnet. Wenn also auf eine Bubble geklickt wird, wird immer der Zustand *Hover* erreicht, bevor die Oberfläche in den *ZoomedIn* Zustand wechselt. Eine Bubble ist prinzipiell immer sichtbar, kann aber nicht im Canvas eingeblendet werden, wenn sich eine andere Bubble gerade in *ZoomedIn* befindet. Dies erfordert allerdings keinen eigenen Zustand. Dieser Zustand kann auch durch Listennavigation erreicht werden. Das dazugehörige Event wird als *listZoom* bezeichnet.

3.3.2 Papers

Die Papers sind ein wenig komplexer als die Bubbles. Sie befinden sich nach dem Start innerhalb der Grenzen der Bubble, welcher sie zugeordnet bzw. als Status: *BehindBubble*. Dieser Zustand kann nun entweder durch die Navigation in der Liste oder Interaktion mit einer der Bubbles

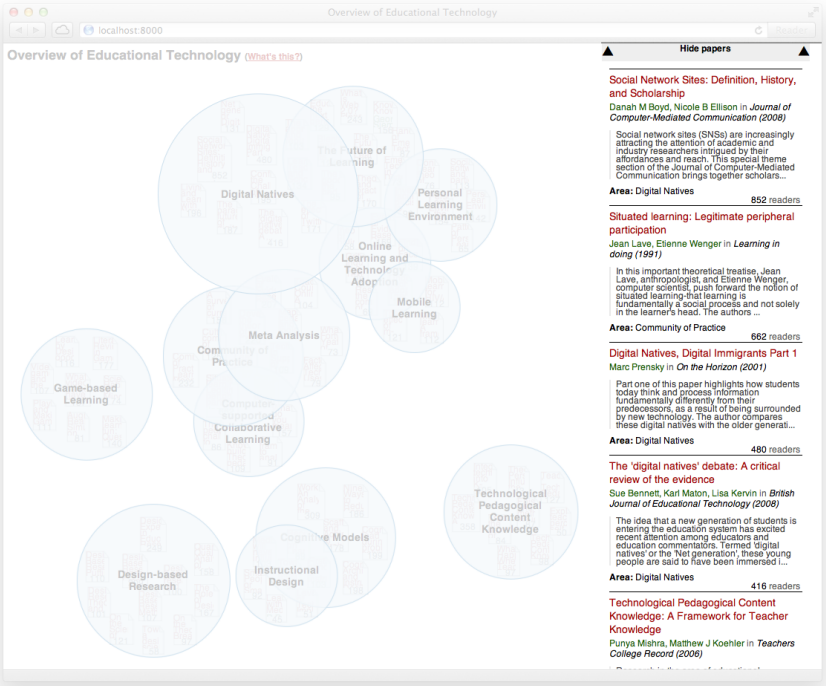


Abbildung 12: Liste der verfügbaren Papers

verändert werden.

Zunächst der Navigationspfad über die Bubble: Wird mit der Maus über eine Bubble *gehovered* ändert sich der Zustand aller darunterliegenden Papers zu *Infrontofbubble*. Nun wird durch einen Klick auf eine Bubble (die sich somit *vom Hover zum ZoomedIn* Status wechselt) in den Zustand *Viewableable* gewechselt. In Viewable kann sich der Status erneut durch ein *Mouseover* event verändern, dieser wird als *Enlarged* bezeichnet. Der Finale Zustand wird als *Active* bezeichnet. Er wird durch einen Klick auf ein *Enlarged* Paper erreicht, ebenso wird die Liste das sich im Active befindende Paper anzeigen.

3.3.3 List

Auch die Liste der Papers kann zwei Zustände annehmen: *Visible* oder *Hidden*. Die Events *Show und Hide* werden noch um ein *Toggle* ergänzt.

Abbildung 13: Header - Zeigt aktuell gewählte Bubble

3.3.4 Popup

Das Popup kann sich in zwei Zuständen befinden: *Visible* oder *Hidden*. Durch die Events Show und Hide wird zwischen diesen beiden Zuständen gewechselt.

3.4 Umsetzung Zustandsverwaltung

Der erste Schritt des Umbaus besteht darin die Javascript Logik der Einzelnen Komponenten aufzuteilen. Sie befindet sich im Moment in einer einzigen Datei mit einem Umfang von *1503 SLOC*.

Das heißt schlichtweg, der erste Schritt besteht darin, alle Funktionen nach ihrer Aufgabe in eine der folgenden Dateien zu verschieben:

- headstart.js
- bubbles.js

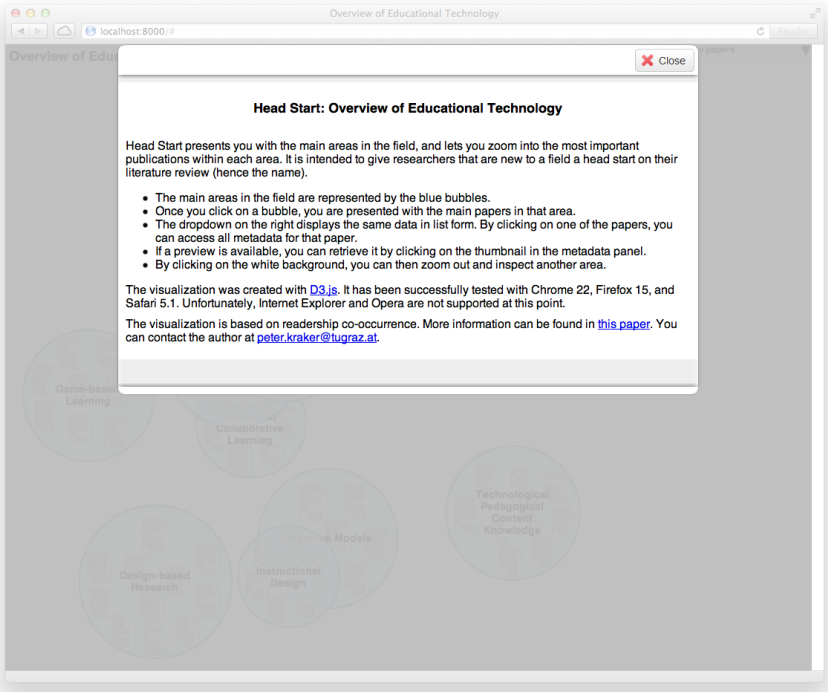


Abbildung 14: Popup - Gibt eine kurze Beschreibung des Projektes wieder

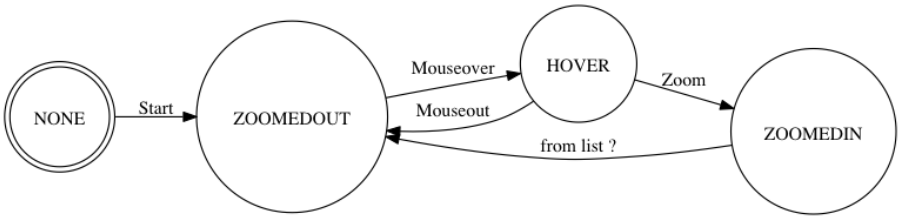


Abbildung 15: Bubbles - Zustände

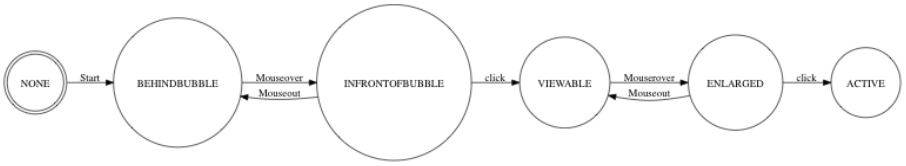


Abbildung 16: Papers - Zustände

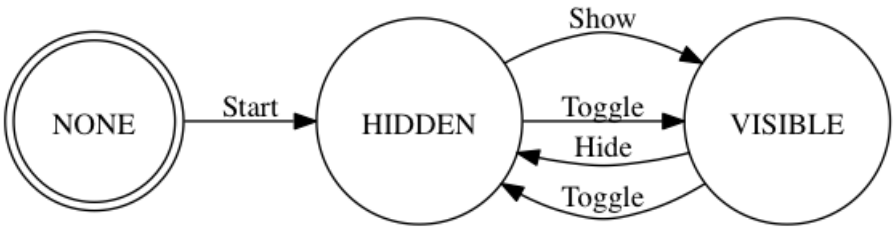


Abbildung 17: Liste - Zustände

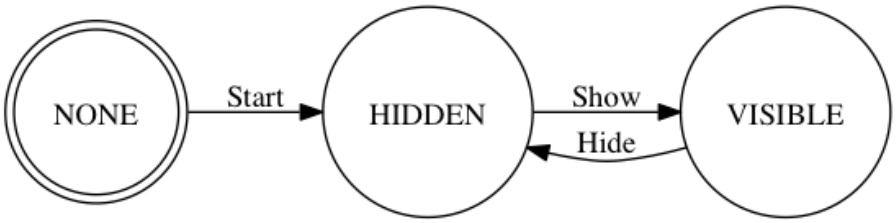


Abbildung 18: Popup - Zustände

- list.js
- papers.js
- popup.js

Anschließend müssen die Statusdefinitionen nach Abschnitt 3.3 implementiert werden. Hierzu bieten sich einige Javascript Bibliotheken an.

- Machina.js[1]
- Machine.js[2]
- javascript-state-machine[3]

Nach dem Motto*»keep your code as simple as possible«*(Fried et al., 2006) soll die kleinste und unkomplizierteste Bibliothek für den Zweck ausgewählt werden. So wird zum einem die Ladezeit von Headstart nicht unnötig erhöht, und zum anderen *»each time you increase the amount of code, your software grows exponentially more complicated.«*(Fried et al., 2006). Die kleinste Bibliothek ist in diesem Fall javascript-state-machine[4].

[1] http://machina-js.org
[2] http://machinejs.maryrosecook.com
[3] https://github.com/jakesgordon/javascript-state-machine
[4] https://github.com/jakesgordon/javascript-state-machine

Es folgt eine kurzer Auszug der zeigen soll, wie übersichtlich die Statusdefintionen mit diesem Micro-Framework aussehen:

```
var popup = StateMachine.create({
  // States for Popup UI Element
  // 1. hidden
  // 2. visible
  events: [
    { name: "start", from: "none", to: "hidden" },
    { name: "show", from: "hidden", to: "visible" },
    { name: "hide", from: "visible", to: "hidden" }
  ],

  callbacks: {
    onbeforestart: function( event, from, to ) {
      // ...
    },

    onshow: function( event, from, to ) {
      // ...
    },

    onhide: function( event, from, to ) {
      // ...
    }
  }
});
```

3.4.1 Auswirkungen des Umbaus

Vor dem Umbau befand sich die ganze Logik des Presentation-Layers in einer einzigen Datei mit einem Umfang von ca 1500 SLOC. Ziel war es, diesen Umstand zu verbessern, um zukünftige Erweiterungen zu ermöglichen. Durch die Anwendung von Statusmaschinen soll der UI Code übersichtlich und wartbar werden. Hier nun die Ergebnisse die mit Hilfe von Plato[1] erzeugt wurden.

Vor dem Umbau ergab sich eine Difficulty von **126** bei 1502 SLOC für das Projekt. Nach dem Umbau erhalten wir die in Tabelle 4 ersichtlichen Werte.

Der Umbau führte zwar zu einer Steigerung der SLOC (Summe: 1910). Dafür wurden deutlich bessere Werte für die Difficulty erreicht. Die Resultate lassen sich durchaus mit denen von jQuery und Grunt vergleichen und sind bei ähnlich großen (SLOC) Dateien meist sogar niedriger

[1] https://github.com/es-analysis/plato

Dateiname	Difficulty	SLOC
popup.js	18	117
headstart.js	55	254
list.js	32	348
bubbles.js	78	621
papers.js	86	570

Tabelle 4: Difficulty Headstart nach Umbau

Dateiname	Difficulty	SLOC
val.js	60	161
data.js	52	176
css.js	78	455
core.js	88	518
effects.js	108	640

Tabelle 5: Difficulty jQuery

ausgefallen (siehe Abb 19).

Abgesehen von den gemessenen Ergebnissen (Verbesserungen) nun ein paar subjektive Anmerkungen: Das Projekt befindet sich in einem übersichtlicheren Zustand. Die in der UI ersichtlichen Elemente spiegeln sich nun deutlich im Quellcode. Das Projekt ist Aufgrund der Aufteilung von Funktionen in Objekte und Dateien leichter zu warten und der Einbau der Zeitkomponente ist mit deutlich weniger Aufwand zu erreichen.

3.5 Zielsetzung für die Zeitkomponente

Der ursprüngliche Gedanke für Headstart war, sich als Wissenschaftler schnell und übersichtlich ein Bild von unterschiedlichen und vor allem neuen Forschungsgebieten machen zu können. Nun wäre es auch sehr interessant, sich Gedanken darüber zu machen wie sich diese Gebiete denn im Laufe der Zeit eigentlich verändern und welche Trends man eventuell von diesen Daten ablesen und ausmachen kann. Der Benutzer soll die Möglichkeit haben, sich unterschiedliche Datensätze zu unterschiedlichen Zeitpunkten anzusehen, um daraus Schlussfolgerungen zu ziehen

Dateiname	Difficulty	SLOC
util.js	36	180
file.js	70	429
task.js	70	436

Tabelle 6: Difficulty Grunt

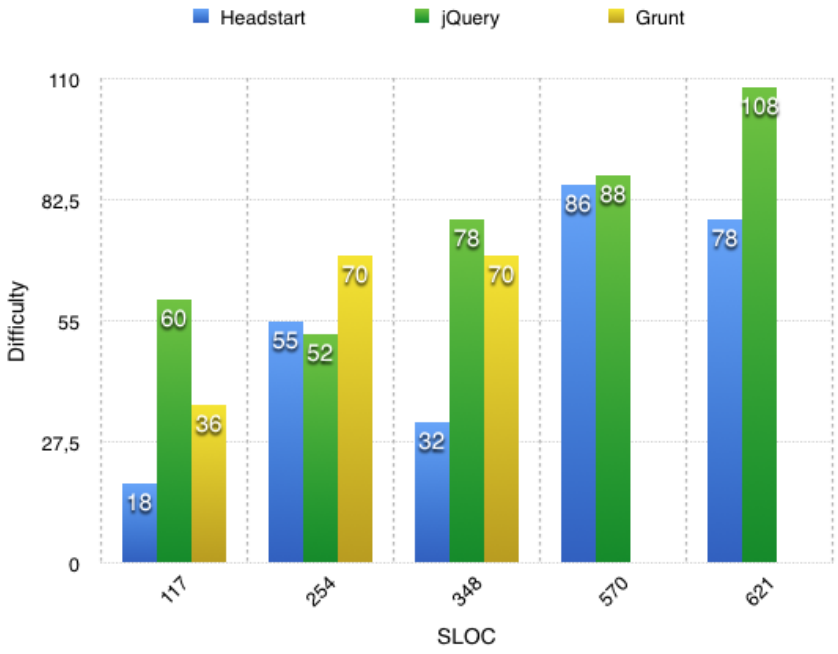


Abbildung 19: Difficulty und SLOC Vergleich mit Grunt und jQuery

(Chen, 2010). Es sollen z.B. Fragen wie diese beantwortet werden: Welches Forschungsgebiet hat sich nach seinen Publikationen in den letzten Jahren deutlich vergrößert? Ziel ist es eine intuitive und schnelle Umsetzung zu programmieren um das Konzept zu testen. Das menschliche Vermögen, Veränderungen zu interpretieren, soll ausgenutzt werden und somit mögliche Trends leicht ersichtlich gemacht werden. Das heißt unter anderem, Ursachen von Change Blindness zu vermeiden. Außerdem soll die neue Visualisierung nicht zu stark von der bereits vorgestellten und dem Benutzer bereits bekannten Ansicht abweichen, um möglichst die gleichen Informationen, nur eben über eine Zeitspanne, darstellen zu können.

3.6 Umsetzung Zeitkomponente

Es wurden Index Charts, Stacked Graphs, Horizon Graphs und Small Multiples als mögliche Visualisierungsmethoden für Zeitreihen vorgestellt und das Problem der Change Blindness erwähnt, nun sollen die für unseren Zweck unpassenden Methoden ausgeschlossen werden, um schließlich zu einer zufriedenstellenden Lösung zu kommen. Hier sei erwähnt, dass man keine "richtige"Lösung finden kann. Es gilt vielmehr die Vor- und Nachteile der möglichen Implementationen, unter Berücksichtigung von Aspekten wie Change Blindness oder Skalierung, abzuwägen und danach eine gute Designentscheidung zu treffen.

3.6.1 Index Charts, Stacked Graphs, Horizon Graphs

Würden wir uns für eine dieser Visualisierungen entscheiden, beschränkten wir uns darauf, ein bestimmtes Attribut (wie zum Beispiel die Anzahl der Leser, die dem Umfang der Bubbles/Forschungsgebiete entspricht) und dessen Verlauf anzuzeigen. Der Grund dafür liegt darin, dass die X-Achse in allen Methoden die Zeit darstellt, und somit natürlich nur noch die Y-Achse für ein anderes Attribut zur Verfügung steht.

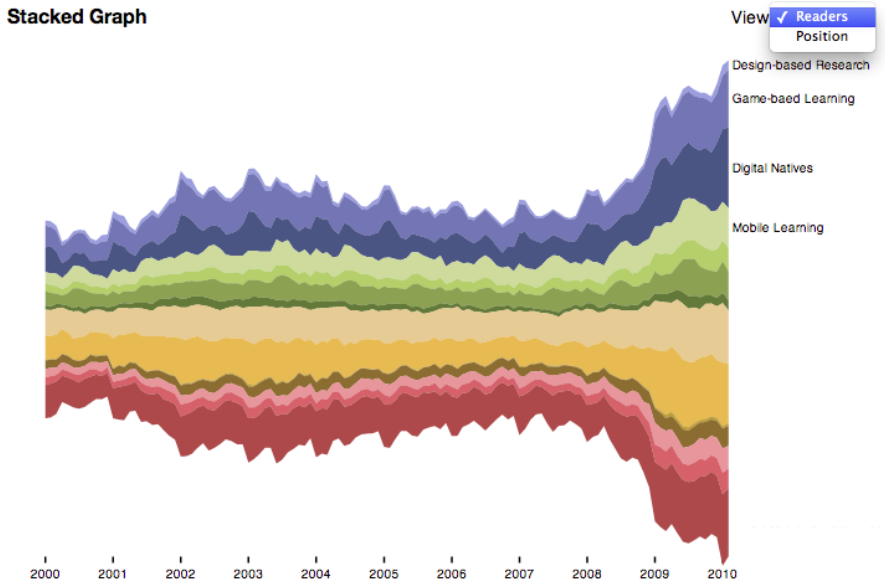


Abbildung 20: Stacked Graph mit Dropdown

Falls sich der Anwender aber für den Zusammenhang von zwei Forschungsbieten interessiert (Position der Blasen) ist ein anderer Graph zu zeichnen. Man könnte zum Beispiel ein Dropdown wie in (Abbildung 20) zur Verfügung stellen, welches zwischen den beiden Graphen wechselt. Das Problem ist aber auf alle weiteren Informationen aus der ursprünglichen Visualisierung (und zukünftigen Veränderungen) zu erweitern.

Das größte Schwachpunkt dieser Implementation ist allerdings die starke Abweichung der Visualisierung zu der normalen Ansicht von Headstart. Der Benutzer, der nach Verwendung der normalen Ansicht mit der Visualisierung bereits vertraut ist, muss sich mit einer deutlich unterschiedlichen Repräsentation der Informationen vertraut machen. Bei der Verwendung der Visualisierung soll der Benutzer aber im Idealfall so wenig kognitiven Aufwand wie möglich aufbringen müssen, und schnell Erkenntnisse über die Entwicklung der Forschungsbiete treffen.

3.6.2 Zeitkomponente mit Small Multiples

Eine deutlich besserer Lösungsansatz ergibt sich mit Small Multiples. Dieser ist in Abbildung 21 ersichtlich.

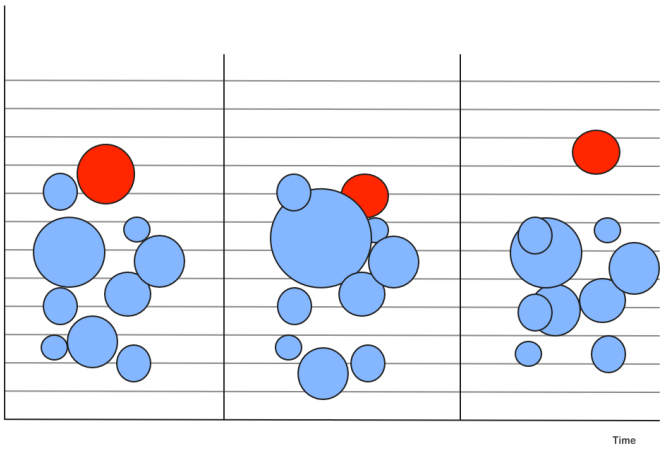


Abbildung 21: Small Multiples Entwurf

Hier wird die ursprüngliche Visualisierung mit Small Multiples verwendet. Das heißt, Headstart wird wie in der normalen Ansicht einfach für mehrere Zeitpunkte gezeichnet und nebeneinander platziert. Wir haben somit das Problem des Informationsverlustes weitgehend vermieden, da praktisch alle Informationen der ursprünglichen Ansicht 9 vorhanden bleiben. Diese Visualisierung hat auch den Vorteil, dass die in Change Blindness angesprochenen Unterbrechungen vermieden werden (solange die Zeitreihe klein bleibt). Und wie bereits erwähnt haben Small Multiples die Eigenschaft direkte visuelle Vergleiche von Formen, Unterschieden von Objekten zu provozieren (Tufte, 1990).

Um Veränderungen noch leichter zu erkennen (*Change-Detection erleichtern*), werden drei weitere visuelle Hilfsmittel eingesetzt. Zuerst wird ein Gitter gezeichnet, welches wiederum zwei Funktionen hat: Zum einen werden die unterschiedlichen Datensätze deutlich voneinander trennt. Zum anderen erleichtert es dem Anwender die Größe und Position der Forschungsbereiche einzuschätzen.

Als zweites Hilfsmittel reagieren die Forschungsbereiche auf Mausbewegungen. Das bedeutet konkret: wenn der Anwender mit der Maus über eines der Forschungsgebiete fährt, wird die entsprechende Bubble auch in den anderen Momentaufnahmen visuell hervorgehoben (sprich Hintergrundfarbe ändert sich).

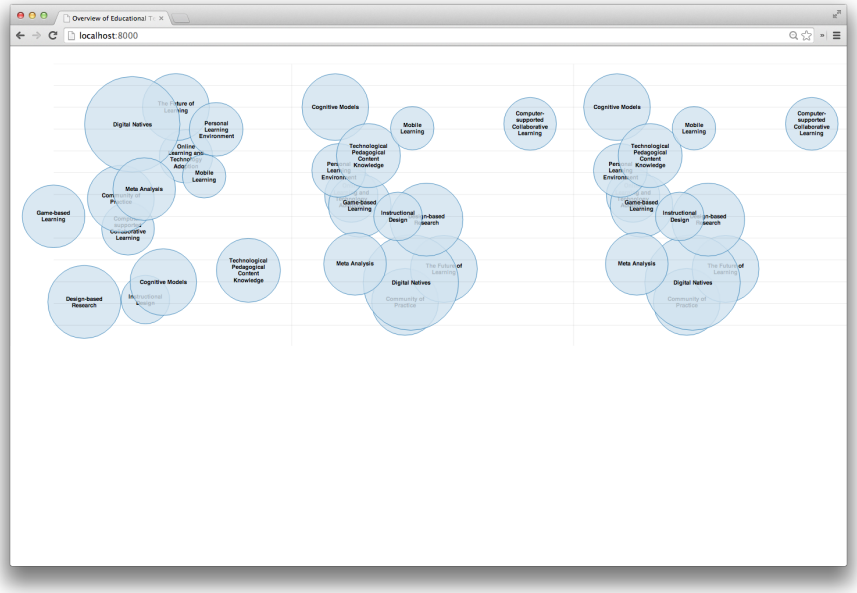


Abbildung 22: Headstart Timelineview Small Multiples - Zufallswerte

Als letzte Unterstützung werden zwischen gleichen Forschungsgebieten Verbindungslinien eingezeichnet. Dies führt zu einer noch schnelleren Assoziation als nur die Hintergrundfarbe zu verändern. Außerdem ist die Verbindungslinie sehr hilfreich wenn ein Forschungsgebiet in einem Zeitabschnitt schlichtweg nicht vorhanden ist (siehe Abbildung 23).

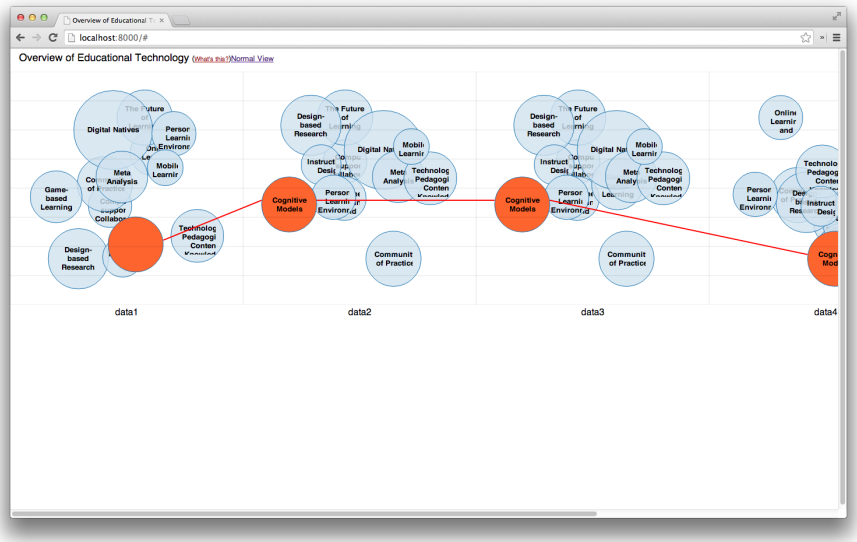


Abbildung 23: Verbindungslinien als Orientierungshilfe

Ein weiteres Kriterium, welches es zu berücksichtigen gilt, ist die Skalierung. Wie gut verhält

sich die Visualisierung bei vielen Datensätzen? Der Entwurf mit Small Multiples nimmt hier im Vergleich zu den anderen vorgestellten Methoden deutlich mehr Platz ein. Während im Index Chart einfach die X-Achse anzupassen ist, muss für Small Multiples jeder neue Datensatz gezeichnet werden. Als Lösung wird in unserem Fall das Scrollverhalten des Browsers ausgenutzt. Die Zeitreihe wird einfach kontinuierlich in X-Richtung für jeden Datensatz gezeichnet. Dies kann zu Change Blindness führen; wenn beispielsweise der erste und letzte Zeitpunkt miteinander verglichen werden sollen, muss der Anwender zwischen diesen hin und her scrollen. Im Normalfall (Standard Auflösung bzw. Browserfenster) erlaubt es die Visualisierung 3-4 Zeitpunkte auf einmal zu betrachten. Das sollte genug Kontext schaffen, um Vergleiche anstellen zu können. Im Zweifelsfall kann die Visualisierung einfach angepasst werden, um mehr Datensätze anzuzeigen.

3.6.3 Implementation

Durch das im ersten Teil durchgeführte Refactoring ist die Umsetzung tatsächlich nicht all zu aufwendig. Hier sollen nur die konzeptionell wichtigen Schritte erwähnt werden.

Um die Zeitkomponente und den Wechsel zwischen beiden Ansichten zu implementieren, wurde die gesamte *Headstart Visualisierung* als Objekt abstrahiert. In weiterer Folge wurde die Statusmaschine mit Hilfe der javascript-state-machine Bibliothek definiert.

Headstart befindet sich entweder in der *normalen-* oder *Zeitreihen* Ansicht. Der Wechsel findet durch einen Klick des Benutzers statt. Weiters wurde die Entscheidung getroffen, auf das rendern der Papers zu verzichten. Dies hat den Vorteil, dass die Visualisierung sehr schnell gezeichnet werden kann. Und ist aus Sicht der Zielsetzung kein wirkliches Problem, da das Interesse in den Forschungsgebieten liegt.

Im nachfolgenden Listing die Statusdefinition von Headstart:

```
StateMachine.create({

target: HeadstartFSM.prototype,

  events: [
    { name: "start", from: "none", to: "normal" },
    { name: "totimeline", from: "normal", to: "timeline" }
  ]

});
```

Es ist klar ersichtlich, das sich das *Headstart* Objekt also entweder im *normal* oder *timeline* Zustand befindet und mit dem *totimeline* Event von *normal* zu *timeline* gewechselt wird. Der

Wechsel in die andere Richtung (d.h. *timeline* zu *normal*) geschieht einfach durch das Neuladen der Seite.

Interessant ist jetzt natürlich das *totimeline* Event. Aus ihm sollte nämlich ersichtlich werden ob der Umbau tatsächlich zu einer Vereinfachung geführt hat und das Projekt in Zukunft schneller zu erweitern ist.

Dazu folgt ein Auszug aus dem Projekt, der verdeutlichen soll wie übersichtlich die Implementation der Zeitkomponente durch den Einbau der Statusverwaltung geworden ist.

```
// 'ontotimeline' transitions from Headstarts "normal" view to the
   timeline
  // view. In a nutshell:
  // 1. it requires some cleanup
  //   - objects which are no longer needed
  //   - the canvas
  // 2. rendering of new elements, on a bigger
  //   chart
  ontotimeline: function( event, from, to ){

    // remove bubbles
    delete this.bubbles;
    this.bubbles = {};
    // clear the canvas
    $("#chart_canvas").empty();

    // clear the list list
    $("#papers_list_container").empty();

    popup.current = "hidden";
    papers.current = "none";
    list.current = "none";

    // change heading to give an option to get back to normal view
    popup.drawNormalViewLink();
    this.initDynamicVariables();

    // 1
    var bubbles = new BubblesFSM();
    this.registerBubbles(bubbles);
    bubbles.title = "data1";

    // 2
    var bubbles2 = new BubblesFSM();
    this.registerBubbles(bubbles2);
    bubbles2.title = "data2";
```

```
    // 3
    var bubbles3 = new BubblesFSM();
    this.registerBubbles(bubbles3),
    bubbles3.title = "data3";

    // need a bigger width for the timeline view
    this.svg.attr("width", (this.max_chart_size * this.bubblesSize()
        + "px") );
    this.svg.attr("height", this.max_chart_size);
    this.chart_id.attr("overflow-x", "scroll");

    d3.csv("data/educational-technology.csv", function( csv ) {
      bubbles.start( csv );
    });
    d3.csv("data/edu.csv", function( csv ) {
      bubbles2.start( csv );
    });
    d3.csv("data/edu2.csv", function( csv ) {
      bubbles3.start( csv );
    });


    this.drawGrid();
    this.drawGridTitles();
  }
}
```

Die wesentlichen Aufgaben von *totimeline* lassen sich aus den Kommentaren entnehmen. Für die neue Visualisierung müssen ein paar Elemente und Status der Applikation zurückgesetzt werden. Anschließend wird die Navigation angepasst um dem Anwender den Wechsel zur normalen Ansicht zu ermöglichen. Es folgt die Initialisierung der Small Multiples und letztlich werden die Hilfslinien gezeichnet. Der Code ist leicht lesbar und reflektiert die Visualisierung sehr deutlich.

4 Fazit und Ausblick

Das bestehende Projekt wurde um einen Statusverwaltungsmechanismus und eine Visualisierung von Zeitreihen erweitert. Die Visualisierung ermöglicht es den Benutzern schnell, und einfach Vergleiche zwischen Forschungsgebieten und deren Entwicklung anzustellen.

Der Umbau ist durch die vorgestellten Metriken gemessen worden und zeigt eine deutliche Verbesserung des Projektes. Dieser Umstand ist auch durch die in Abschnitt 3.6 angeführten Listings zu erkennen. Headstart reflektiert nun das Verhalten der Benutzeroberfläche und deren Elementen in Statusmaschinen.

Durch diese klaren Abstraktionen war auch die Entwicklung des Visualisierung für Time Series Data mit relativ geringem Aufwand verbunden. Die Designentscheidung für Visualisierung selbst wurde nach einer umfassenden Recherche und unter Berücksichtigung diverser Aspekte wie Change Blindness getroffen. Um Vergleiche noch einfacher zu machen, wurden zusätzliche Elemente in die Small Multiples Visualisierung eingebaut.

Es handelt sich aber um ein Proof-of-Concept. Weil die Visualisierung zum einem mit zufallsgenerierten Daten, und zum anderen nicht in einer Benutzerevaluierung getestet wurde, lassen sich über deren Effektivität nur begrenzt Aussagen machen. In diesen Bereichen besteht also starkes Verbesserungspotential. Zudem hat die Visualisierung Probleme bei der einer großen Anzahl von Datensätzen. Je länger die Zeitreihe, desto unübersichtlicher wird die Visualisierung.

Schlussfolgernd ist mit dem Umbau eine stabile Basis für zukünftige Erweiterungen und Anpassungen gelegt worden. Die Entwicklung von weiteren Features sollte mit deutlich weniger Zeitaufwand verbunden sein. Die Visualisierung der Zeitreihen ist der ursprünglichen Visualisierung treu geblieben und bietet Forschern die Möglichkeit, sich schnell Eindrücke über die Entwicklungen und Veränderungen von Forschungsgebieten zu verschaffen.

Abbildungen

Tabellen

Quellen

Alon U., 2009: *How to choose a good scientific problem*, in: Molecular cell, 35(6), S. 726–728.

Bitter, Rick, 2001: *State Machines*. `http://www.physics.utah.edu/~jui/3620/supp/LV_Adv_Prog/2049ch03.pdf` (Zugriff: 12.12.2013)

Chen C., 2010: *Information visualization*, in: Wiley Interdisciplinary Reviews: Computational Statistics, 2(4), S. 387–403.

Elliot E., 2013: *Programming Javascript Applications*, O'Reilly.

Fried J., Hansson H., Linderman M., 2006: *Getting Real: The Smarter, Faster, Easier Way to Build a Successful Web Application*, 37signals.

Grimes J., 1996: *Perception*, Vancouver studies in cognitive science, Oxford University Press, USA.

Heer J., Bostock M., Ogievetsky V., 2010: *A Tour Through the Visualization Zoo*, in: Commun. ACM, 53(6), S. 59–67.

J. H., 2010: *Index Chart*.

Kharabaf S., Abdollahi. M., 2012: *Science growth in Iran over the past 35 years*, in: J Res Med Science.

Kohlhammer J., Keim D., Pohl M., Santucci G., Andrienko G., 2011: *Solving Problems with Visual Analytics*, in: Procedia Computer Science, 7(0), S. 117 – 120. <ce:title>Proceedings of the 2nd European Future Technologies Conference and Exhibition 2011 (FET 11)</ce:title>.

Kraker P., Jack K., Schlögl C., Trattner C., Lindstaedt S., 2013: *Head Start: Improving Academic Literature Search with Overview Visualizations based on Readership Statistics.*, in: .

nach Manos Papagelis P.W., 2013: *3 Tier Architektur*.

Nguyen V., Deeds-Rubin S., Tan T., Boehm B., 2007: *A SLOC Counting Standard*. `http://sunset.usc.edu/events/2007/CIIForum/presentationByDay/frinov2/2ToolFair/ASLOCCountingStandard.pdf` (Zugriff: 27.11.2013)

Papagelis M., 2012: *WEB APP ARCHITECTURES: MULTI-TIER (2-TIER, 3-TIER) & MVC*. http://queens.db.toronto.edu/~papaggel/courses/csc309/docs/lectures/web-architectures.pdf

Samek M., 2009: *A Crash Course in UML State Machines*. http://classes.soe.ucsc.edu/cmpe013/Spring11/LectureNotes/A_Crash_Course_in_UML_State_Machines.pdf (Zugriff: 12.12.2013)

Serebrenik A., 2011: . http://www.win.tue.nl/~aserebre/2IS55/2010-2011/10.pdf (Zugriff: 27.11.2013)

Simons D.J., A. D.T.L., 1997: *Change blindness*, in: Trends in cognitive sciences, S. 261–267.

Simons D.J., Rensink R.A., 2005: *Change blindness: Past, present, and future*.

Thomas D., Hunt A., 2002: *State machines*, in: Software, IEEE, 19(6), S. 10–12.

Tufte E., 1990: *Envisioning Information*, Graphics Press.

W3TECHs, 2013: *Useage of Javascript for websites*. http://w3techs.com/technologies/details/cp-javascript/all/all (Zugriff: 15.10.2013)