UNIVERSITÄT OSNABRÜCK

INSTITUT FÜR INFORMATIK

Masterarbeit

Datenaggregation und -analyse im Kontext eines sozialen Netzwerks am Beispiel eines personalisierten Feed-Readers

Nicolas Neubauer

November 2010

Erstgutachter: Prof. Dr. Oliver Vornberger
Zweitgutachter: Jun.-Prof. Dr. Elke Pulvermüller

Zusammenfassung

In dieser Arbeit wird die Konzeption und Implementierung des Projekts *Nuntio* beschrieben. Dabei handelt es sich um eine für Tablet-Computer wie Apples iPad optimierte, mit dem Framework Ruby on Rails implementierte Webapplikation, die den Nutzern auf Basis von so genannten Newsfeeds eine ständig aktuelle, personalisierte Tageszeitung zur Verfügung stellt. Diese Funktionalität ist in ein eigenes soziales Netzwerk eingebettet, das neben der Gewinnung von Daten zur Optimierung der Analyse- und Aggregationsprozesse auch als Plattform für Kommunikation und das Auffinden neuer, interessanter Inhalte dient.

Neben den zugrundeliegenden Technologien und dem Aufbau des Systems werden die verschiedenen Techniken zur Ähnlichkeitsanalyse, Verknüpfung und interessensbasierter Sortierung der Nachrichtenartikel behandelt. Schließlich wird der zusätzliche Nutzen der Einbettung des Systems in ein soziales Netzwerk sowie die Einbindung der Nutzerdaten zur Optimierung der Analysetechniken diskutiert.

Inhaltsverzeichnis

Kapitel 1

Einleitung

In dieser Arbeit soll die Konzeption und Implementierung des Projekts *Nuntio*, einem personalisierten Feed-Reader vorgestellt werden. In dieser Einleitung werden dabei zunächst die Idee und das grundlegende Konzept beschrieben und schließlich das Ziel und der Aufbau dieser Arbeit vorgestellt.

1.1 Die Idee hinter Nuntio

In dem im Jahr 2002 erschienen Science-Fiction-Drama *Minority Report* ist in einer Szene ein Mann in der U-Bahn zu sehen, der eine Tageszeitung liest, die sich - während er sie liest - automatisch aktualisiert. Im Jahr 2010, in einer Zeit des allgegenwärtigen Internets, muss diese Szene deutlich weniger phantastisch wirken, als auf einen Betrachter im Jahr 2002, als das Gros der Mobiltelefone noch ein Monochrom-Display besaß und die Kosten für mobiles Internet über das GSM-Netz für Privatanwender kaum erschwinglich waren. Spätestens jedoch mit der Einführung von Apples iPhone im Jahr 2007 entwickelte sich ein Trend, der heute deutlich zu beobachten ist und immer weiter zur mobilen Nutzung des Internets führt.

Abbildung 1.1 zeigt dazu die Entwicklung des Anteils, den der mobile Internetkonsum, also auf eben solchen Geräten, im Vergleich zu der mittlerweile unvorstellbaren Größe des globalen Internetkonsums inne hat. Wie deutlich zu sehen ist, lag der Marktanteil Anfang 2007 noch unter einem Promille, während Ende 2009 bereits etwa ein mobiler Internetzugriff auf einhundert stationäre Zugriffe kommt - seither also eine Verzehnfachung. In diesem Kontext verwundert auch die Zahl von mittlerweile 40% der erwachsenen US-Amerikaner nicht, die in einer Studie des Pew-Research-Centers von Juli 2010 [Smi10] angaben, dass sie das "Internet, E-Mail oder Instant-Messaging" über ein Mobiltelefon nutzen.

Natürlich ist die Nutzung des Internets in einem mobilen Kontext nichts allzu Neues, da Laptops und mobile Datennetze bereits deutlich älter sind als die neuen Geräte. Es ist also vermutlich eher die Simplizität und deutlich höhere Portabilität dieser neuen Generation von Mobiltelefonen, die für den dargestellten Sprung des Marktanteils des mobilen Internets verantwortlich sind.

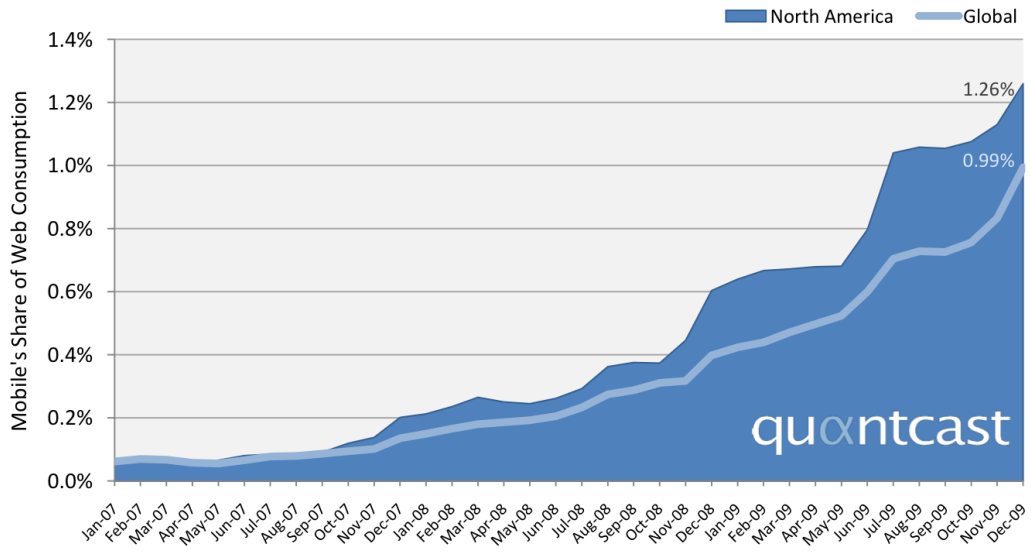


Abbildung 1.1: Marktanteil mobiler Internetnutzung laut Quantcast-Studie (aus: [Qua10])

Gerade auch mit dem Aufkommen von Geräten wie Apples iPad, die diese Konzepte - einfach und portabel - beibehalten, aber einen deutlich größeren Bildschirm und - da die meisten dieser Geräte mit einem Touch-Screen ausgestattet sind - eine größere Eingabefläche bieten, ist zu vermuten, dass der Anteil mobiler Internetnutzung weiter steigen wird.

Betrachtet man also heute die oben beschriebene Film-Szene, würde man sich wohl kaum über die sich selbst aktualisierende Tageszeitung wundern. Höchstens noch über das Aussehen des Geräts im Film, das kaum Apples iPad oder einem anderen Vertreter dieser neuen Generation von Tablet-PCs ähnelt.

Kurz gesagt ist dies auch die Idee hinter Nuntio. Eine digitale Tageszeitung, die stets auf dem aktuellen Stand und eben für ein solches Gerät, wie das iPad optimiert ist. Zunächst mag man sich dabei nach dem Nutzen fragen, da diese Geräte doch im Allgemeinen bereits einen voll funktionsfähigen Webbrowser installiert haben und so die Online-Versionen beliebiger Tageszeitungen, die häufig auch über den Tag aktualisiert werden, bereits verfügbar sind. Nuntio geht hier aber noch einen Schritt weiter und verfolgt drei wesentliche Konzepte, die es von der Nutzung einer bereits bestehenden Online-Version einer Tageszeitung deutlich abgrenzen.

Nachrichten-Aggregation Nuntio ist ein Nachrichten-Aggregator und stellt damit nicht nur die Nachrichten *einer* Quelle dar, sondern sammelt verschiedenste Online-Nachrichten an, die die Nutzer abonnieren können. Als Datenmaterial bieten sich die von einer Vielzahl von Internetseiten angebotenen Newsfeeds an. Diese werden in standardisierten Formaten ausgeliefert und beinhalten häufig zumindest eine Überschrift und eine Kurzbeschreibung der jeweiligen Ressource. Ressource deshalb, weil die Newsfeed-Technik nicht auf das Publizieren von Nachrichten im eigentlichen Sinne beschränkt ist, sondern ebenso für die Bekanntmachung neuer Blog-Einträge oder Postings in einem Forum genutzt werden kann. Neben dem reinen Zusammenfassen von

Ressourcen aus verschiedenen Online-Quellen in einer Art Data Warehouse, ist die Aggregations-Funktion auch so zu verstehen, dass zusammengehörige Nachrichten aus verschiedenen Quellen automatisiert miteinander verknüpft werden sollen. Für den Nutzer äußert sich dies so, dass er zum einen die Möglichkeit hat, zu einer bestimmten Nachrichtenmeldung verwandte Meldungen anderer Quellen anzeigen zu lassen, zum anderen, dass sich sehr ähnelnde Ressourcen zu einer Gruppe von Nachrichten zusammengefasst werden können.

Personalisierung In gewisser Weise überlappt die Tatsache, dass Nuntio nicht nur auf eine Nachrichten-Quelle beschränkt ist, mit dem zweiten wichtigen Konzept - der Personalisierung. Nutzer können so selbst entscheiden, welche Nachrichten-Quellen sie gerne lesen möchten und welche eben nicht. Viel wichtiger ist in diesem Rahmen jedoch die Interessensanalyse eines bestimmten Benutzers. Nutzer können bestimmte Nachrichten oder Ressourcen als *interessant* bewerten, was es ermöglicht, bei zukünftigen Meldungen besser einschätzen zu können, ob diese für einen Nutzer wichtiger oder weniger wichtig sind. In Verbindung mit der Aggregation und Gruppierung entsteht so bereits eine völlig personalisierte Tageszeitung, die auf Basis der von einem Nutzer gewünschten Quellen und seines Interesses die Nachrichten-Daten so darstellt, wie der jeweilige Benutzer es eben wünscht.

Soziale und kollaborative Funktionen Das dritte Hauptkonzept in Nuntio ist die Einbettung des Systems in ein eigenes soziales Netzwerk. Ziel hierbei ist es, die vormals sehr auf den einzelnen Benutzer ausgerichteten Funktionen auch auf zusammengehörige Nutzergruppen und letztlich über die gesamte Nutzerbasis zu erweitern. Das System ist dabei nicht darauf ausgelegt, einen möglichst großen, "virtuellen" Freundeskreis aufzubauen, sondern vielmehr, in einer kleinen Gruppe Nachrichten austauschen zu können und eine Diskussionsplattform zu schaffen. Schließlich sollen die Nutzer die Möglichkeit haben, in Form einer Art von "collaborative tagging" [GH06] den Analyse-Prozess bei der Aggregierung und Gruppierung von Artikeln zu beeinflussen.

Betrachtet man diese drei Teilaspekte, kann man sagen, dass Nuntio ein in ein soziales Netzwerk eingebetteter, personalisierter Feed-Reader ist, dessen Analyse- und Aggregationsalgorithmen durch die Nutzer beeinflusst und optimiert werden können. Um noch eine Abgrenzung zu klassischen News-Readern zu schaffen, soll folgendes Nutzungsszenario dargestellt werden:

Ein interessierter Nutzer lese regelmäßig auf den Webseiten der Nachrichtenmagazine *Spiegel Online*[1], *Zeit Online*[2] und *Tagesschau Online*[3]. Weiterhin verfolge er zwei private Blogs seiner Freunde und schließlich eine Webseite mit Produkt-Tests zu Computern und anderen technischen Geräten. Um all diese Webseiten einzeln auf Neuigkeiten zu überprüfen, ist der Aufwand relativ groß - vermutlich bereits zu groß, um beispielsweise eine Tageszeitung beim Frühstück zu ersetzen. Als Lösung bietet sich ein klassischer News-Reader an, den es für die verschiedensten Betriebssysteme, insbesondere auch für die von Tablet-PCs, gibt. Diese Programme sammeln

[1] http://www.spiegel.de
[2] http://www.zeit.de
[3] http://www.tagesschau.de

ähnlich wie Nuntio die Informationen aus den Newsfeeds der Webseiten und präsentieren diese häufig in einer Liste, sortiert nach Veröffentlichungsdatum und/oder Nachrichten-Quelle. Bei entsprechender Darstellung kann ein solches System bereits - natürlich von der inhaltlichen Qualität der Online-Nachrichten abgesehen - ein adäquater Ersatz für eine Tageszeitung sein. Bei den meisten Feed-Readern besteht jedoch eine Reihe von Problemen, die erst bei intensiverer Nutzung zu Tage treten. Zunächst einmal liefern die drei in diesem Beispiel betrachteten Nachrichtenseiten eine große Anzahl tagesaktueller Nachrichtenmeldungen, die sich inhaltlich kaum voneinander unterscheiden. Ist beispielsweise Wahlabend einer Bundestagswahl, werden alle drei Webseiten davon berichten, was es in einer Listenansicht erschwert Artikel auszumachen, die sich mit einem anderen Thema beschäftigen. In Nuntio wird die inhaltliche Ähnlichkeit der Artikel automatisch erkannt und in der Nachrichtenübersicht erscheint nur ein Artikel. Die übrigen, thematisch gleichen werden diesem als Gruppe zugeordnet, so dass jedes Thema im Optimalfall nur einmal - mit entsprechender Nachrichtengruppe - angezeigt wird. Ein weiteres Problem besteht im Unterschied der Häufigkeit des Erscheinens neuer Meldungen. Während die obigen Nachrichtenmagazine von einer Vielzahl von Redakteuren gespeist werden und mitunter mehrere hundert Nachrichtenmeldungen pro Woche veröffentlichen, erscheint in den beiden hypothetischen privaten Blogs vermutlich nur ein Bruchteil der Meldungen pro Woche. Nuntio erkennt dies und erhöht die Wichtigkeit von Artikeln, die in Quellen mit niedriger Publikationsfrequenz erscheinen, so dass sie neben den großen Nachrichtenmagazinen nicht untergehen. Schließlich besteht noch das wohl gewichtigste Problem, nämlich das Interesse des Nutzers. Ein gutes Beispiel ist hier die Nachrichten-Quelle zu Produkt-Tests, von denen den hypothetischen Nutzer diejenigen ganz besonders interessieren sollen, die von Produkten handeln, die der Nutzer schon besitzt oder aber von deren Herstellern. In Nuntio ist es für den Nutzer möglich, bestimmte Nachrichten - beispielsweise eben die relevanten Produkttests - als *interessant* zu markieren, was dazu führt, dass zukünftig erscheinende Meldungen, in denen der gleiche Herstellername vorkommt, als wichtiger gekennzeichnet werden, als Meldungen in denen die Herstellernamen vorkommen, die als *nicht interessant* markiert worden sind.

Auf diese Weise werden drei Problemfelder von traditionellen Feed-Readern überwunden, wozu noch die oben genannten Vorteile der sozialen und kollaborativen Funktionen kommen.

1.2 Ziel und Aufbau der Arbeit

Allein das schlichte Fehlen eines Systems, das die zuvor genannten Restriktionen oder gar Mängel eines klassischen Feed-Readers überkommt, scheint Motivation genug, um sich mit der Konzeption und letztlich auch Implementierung auseinanderzusetzen. Zudem ist das Gebiet um das so genannte "Web 2.0", also die Sicht auf das Internet als eine Plattform (vgl. [O'R05]) zur Nutzung des "kollektiven Wissens" [Gru08] oder der "kollektiven Intelligenz" ihrer Nutzer, nach wie vor ein außerordentlich aktuelles Forschungsgebiet. Auch die Schnelllebigkeit des Internets und die neuen Technologien, mit denen deren Nutzer - auch und gerade im Hinblick auf die zuvor genannte Verschiebung der Aktivitäten in den mobilen Bereich - nahezu täglich konfrontiert werden, ist Grund genug, sich diesem Thema auch aus wissenschaftlicher Sicht zu widmen.

Das Hauptaugenmerk soll in diesem Rahmen auf dem dritten Kapitel zur Konzeption und Entwicklung der Technologie hinter Nuntio stehen. Dabei werden die Techniken und Algorithmen vorgestellt, die eine automatisierte, weitgehend sprachunabhängige Extraktion relevanter Schlüsselwörter, auch *Tags* genannt, aus textbasierten Quellen vornehmen und über diese Ähnlichkeitsbeziehungen zu anderen Quellen und auch zwischen Quellen und Nutzern aufbauen. Letztlich werden beide Algorithmen durch die Aktivität der Nutzer beeinflusst, so dass - eine ausreichend partizipierende Nutzerschaft vorausgesetzt - ein sich in gewissem Maße selbst optimierendes System entsteht. Neben diesen Techniken wird in Kapitel 3 auch ein Überblick über den Entwurf der skalierbaren Systemarchitektur gegeben und die implementierte Webapplikation als solche vorgestellt.

Zuvor werden im zweiten Kapitel einige der technischen Grundlagen für die konkrete Implementierung von Nuntio besprochen. Im Zuge dessen werden einige Datenbank-Grundlagen, auch im Kontext des Data-Warehousing und der Knowledge Discovery in Databases, aufgearbeitet. Weiterhin wird das Framework Ruby on Rails, das dem Projekt zugrunde liegt, ebenso wie die Technik der Newsfeeds vorgestellt. Abschließend wird in diesem Kapitel ein Blick auf die Idee der sozialen Netzwerke und die geplante Plattform für Nuntio, Tablet-Computer beziehungsweise das iPad geworfen.

Im vierten Kapitel werden schließlich einige weiterführende Überlegungen zu den bestehenden Techniken und der möglichen Nutzung der resultierenden Daten diskutiert und nach einem Rückblick auf die vorgestellten Themen ein abschließendes Fazit gegeben.

An dieser Stelle soll noch ein kurzer Hinweis auf die Begriffsnutzung innerhalb der Arbeit gegeben werden. Der Begriff “Newsfeed” wird dabei synonym mit den Begriffen “Nachrichten-Quelle” oder “Feed” verwendet und beinhaltet jeweils eine Menge von - ebenfalls synonym verwendeten - “Einträgen”, “Nachrichten” oder “Artikeln”.

Abbildung 1.2: Das Ziel der Arbeit: Nuntio auf dem iPad

Kapitel 2

Grundlagen

Bevor in Kapitel 3 die konkrete Konzeption und Implementierung von Nuntio diskutiert wird, ist dieses Kapitel einigen der grundlegenden Technologien und Konzepten gewidmet, auf denen Nuntio aufgebaut ist. Zunächst werden hier einige Grundlagen des genutzten Datenbank-Systems MySQL und dem Konzept eines Data Warehouses besprochen, gefolgt von einem kurzen Überblick über die genutzte Sprache beziehungsweise das Framework Ruby on Rails. Schließlich wird die Technologie der Newsfeeds sowie das Konzept von sozialen Netzwerken und Tablet-Computern erörtert. Auch ohne bereits die Konzepte der konkreten Implementierung vorwegzunehmen, soll dabei stets ein Bezug zum späteren System auf Basis der in der Einleitung vorgestellten Ideen hergestellt werden.

2.1 Datenbanken und Informationsintegration

Nach der Einleitung und Beschreibung der Grundidee hinter Nuntio wird schnell deutlich, dass das Vorhalten der aus den verschiedenen Quellen aggregierten Daten von zentraler Wichtigkeit ist. Zudem wurde deutlich gemacht, dass es sich bei dem System letztlich um eine Webapplikation im Mehrbenutzerbetrieb handeln soll, so dass der Einsatz einer Datenbank keiner weiteren Rechtfertigung bedarf. Die allgemeinen Grundlagen von Datenbank-Systemen sollen an dieser Stelle als bekannt vorausgesetzt werden, vielmehr soll es in diesem Abschnitt um die speziellen Anforderungen an die Datenhaltung im Hinblick auf das Nutzungsszenario bei Nuntio gehen.

2.1.1 Informationsintegration

Betrachtet man die Zielsetzung des Projekts, wird schnell klar, dass man vor einem Informationsintegrations-Szenario steht. Die einzelnen, von den Nutzern zu abonnierenden Nachrichten-Quellen liegen auf verschiedenen, verteilten Online-Systemen vor und sollen später von Nuntio in einer einheitlichen Sicht, also über ein einheitliches Schema, wenn man so möchte, dargestellt werden. Man unterscheidet für ein solches Integrationsszenario zwei verschiedene Vorgehensweisen: Die

materialisierte und die *virtuelle* Integration der Daten. Bei letzterer verbleiben die Quelldaten auf den verteilten Systemen und die eigentliche Integration der Daten, also die Konsolidierung, findet erst im Moment der konkreten Anfrage statt. Man spricht auch von einem föderierten Informationssystem. Gerade im Hinblick auf die Tatsache, dass die Daten im weiteren Analyse-Prozess untereinander und mit dem Profilen der Nutzer verknüpft werden sollen, scheidet dieser Ansatz schnell aus. Bei der materialisierten oder physischen Integration der Daten werden diese aus den Quellen geladen und in ein zentrales Datenlager, beispielsweise ein *Data Warehouse* oder einen *Operational Datastore* überführt. Man spricht dabei auch vom ETL-Prozess.

ETL-Prozess Die Abkürzung ETL steht für die drei enthaltenen Teilprozesse *Extract*, *Transform* und *Load*. Dieser Begriff wird häufig genutzt, wenn man davon spricht, Daten aus mehreren Quellen in einer Zieldatenbank zu konsolidieren.

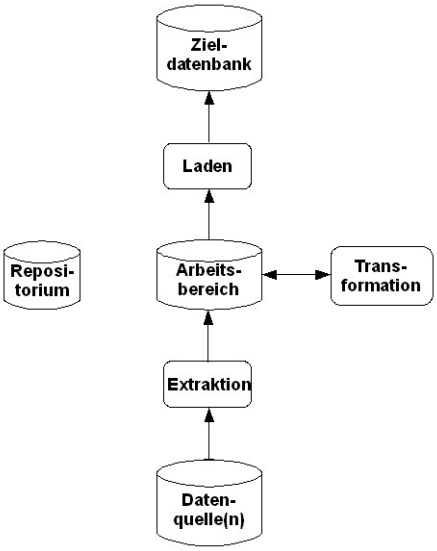


Abbildung 2.1: ETL-Prozess-Visualisierung (aus: Wikipedia[1])

In Abbildung 2.1 ist dieser Prozess, der in Nuntio auch konkret umgesetzt ist, schematisch dargestellt. Zunächst werden die Daten der einzelnen Quellsysteme in einem Extraktions-Prozess geladen. In diesem Fall ist das Laden einfach, da die Daten per HTTP sogar in einem standardisierten Format (siehe auch Abschnitt 2.4) geladen werden können. Die Daten stehen danach im Quellformat, in diesem Fall XML, im Arbeitsbereich zur Verfügung, wo sie zunächst in ein eine objektorientierte und später eine relationale Repräsentation überführt werden, so dass sie im letzten Schritt, dem Load-Prozess in das zentrale Datenbanksystem geladen werden können. Die konkreten Arbeitsschritte die in Nuntios Backend hierbei durchgeführt werden, sind in Abschnitt 3.2.1 beschrieben.

[1] `http://de.wikipedia.org/wiki/ETL-Prozess` - Abruf 05.11.2010

Da eine virtuelle Datenintegration aus den oben genannten Gründen nicht in Frage kommt, sollen an dieser Stelle auch die mit der physischen Integration verbundenen Nachteile nicht unerwähnt bleiben. Zunächst wäre da die Datenaktualität zu nennen, die einer virtuellen Integration, wo stets mit den aktuellen Daten gearbeitet wird, grundsätzlich nachsteht. Man kann jedoch durch entsprechend geringe Aktualisierungsintervalle versuchen, diesen Nachteil zu relativieren. Ein deutlich gewichtigerer Nachteil ist der erhöhte Hardware-Bedarf im Vergleich zur virtuellen Integration, wo sich die Rechenlast im Beispiel von Nuntio sogar auf die Klienten verteilen ließe. Auch hier muss oder kann man diesen Nachteil jedoch relativieren, da die Analyse-Prozesse schlicht eine globale Datensicht benötigen, und der Hardware-Bedarf, der allein für den ETL-Prozess besteht, im Vergleich zu dem der Analyse-Prozesse eher zu vernachlässigen ist.

2.1.2 Data Warehouse und Operational Data Store

Bei den System-Architekturen im Kontext der materialisierten Datenintegration lässt sich zunächst zwischen den *Data Warehouses* und den *Operational Data Stores* unterscheiden. Beide Begriffe sind dabei eher konzeptueller Natur und umschreiben den Aufbau geeigneter Systeme für die zentralisierte Datenhaltung. Ursprünglich kommen beide Begriffe aus dem betriebswirtschaftlichen Umfeld, das sich in einer der klassischen Definitionen des Data Warehouses nach [Inm96] wiederfindet: “A data warehouse is a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management’s descision making process.” Vernachlässigt man den in der Definition formulierten Zweck des Data Warehouse, lässt sich die Definition auch auf allgemeine Datenhaltungen übertragen. So soll ein Data Warehouse um ein bestimmtes “subject”, also ein Thema oder eine Domäne, organisiert sein, was auf Nuntio zutrifft, da es das Ziel der Datenhaltung die Verwaltung der einzelnen Online-Nachrichten ist. Auch als “integrated” lässt sich die Datenhaltung von Nuntio beschreiben, wie im vorherigen Kapitel ja bereits erläutert wurde. Der Punkt “time-variant” hingegen passt nun weniger gut zum Konzept von Nuntio. Hier wird gefordert, dass das Data Warehouse historische Daten vorhält, die jeweils eine zeitliche Information zur späteren Zuordnung enthalten müssen. Unproblematisch ist dies beim Vorhalten der einzelnen Online-Nachrichten, problematischer ist dies bei den Analyse-Daten. Wie in der Einleitung bereits erläutert wurde, sollen einzelne Nachrichten mit Nutzern und anderen Nachrichten verknüpft werden. Da die Nutzer selbst aber auf die Funktionsweise der Algorithmen und die Verknüpfungen Einfluss nehmen können und auch sollen, gestaltet sich das Vorhalten von historischen Daten schwierig, zumal - unter der Annahme, dass die Ergebnisse durch das Nutzerfeedback verbessert werden - eine historische Sicht auf diese Daten gar nicht nötig ist. Ähnliche Abstriche müssen bei der Erfüllung der Forderung nach dem Attribut “nonvolatile” gemacht werden. In [HK01] wird beschrieben, dass in diesem Punkt nicht nur die physische Persistenz der Daten gefordert wird, sondern auch die Unabhängigkeit des Data Warehouse vom operativen System, so dass im eigentlichen Data Warehouse lediglich Daten eingeladen und gelesen werden können, es aber nicht zu transaktionsbasierten Datenmanipulationsoperationen kommen sollte. Hier liegt wohl das Hauptproblem, möchte man das Data Warehouse als System-Architektur für Nuntios Backend berücksichtigen. Offenbar ist es so, dass die eigentlichen Quelldaten - also die Online-Nachrichten - sich ausgezeichnet für das Ablegen in einem Data Warehouse eignen würden. Auch einen Teil der Daten daraus für Analyse-Prozesse zu nutzen, wäre unproblematisch.

Doch das Arbeiten auf den Daten, also das Ablegen von temporären Analyseergebnissen, die jeweils bis zur nächsten Iteration Gültigkeit haben, passt nicht in das Konzept des klassischen Data Warehouse.

Inmon sieht in [Inm99] für diese Arbeiten die eben andere System-Architektur, den Operational Data Store vor. Die analoge Definition lautet dabei: "An operational data store is a subject-oriented, integrated, volatile, current-valued, detailed-only collection of data in support of an organizations's need for up-to-the-second, operational, integrated collective information." Auf den ersten Blick erfreut das Fehlen der gewichtigen Problempunkte mit der Definition des Data Warehouse, nämlich der zeitlichen Invarianz und der Nicht-Flüchtigkeit der Daten. Problematisch wiederum an dieser Definition ist die direkte Umkehrung der beiden Attribute in ihr Negativ. Zwar sind die entstehenden Analyse-Daten "current-valued" und eben "volatile", trotzdem werden die Basisdaten der Analyse-Prozesse historisch vorgehalten, um in der jeweils nächsten Analyse-Iteration wiederum Gesamtsicht auf die Daten zur Verfügung zu haben und nicht auf den vorherigen Analyse-Ergebnissen aufbauen zu müssen.

Offenbar lässt sich keines der beiden Konzepte mit den vorgestellten, klassischen Definitionen als Grundlage für einen Architektur-Entwurf nutzen. Es ist schlicht so, dass die einfache Anforderung, zwar eine gewisse Menge von Daten historisch vorhalten zu wollen - also im Sinne des Data Warehouse - aber auf diesen Daten auch Analysen laufen zu lassen, die zwar nicht historisch aber doch persistent abgelegt sein sollen, in einem System - nach obigen Definitionen - nicht erfüllbar ist. Natürlich gibt es hier auch andere Definitionen, die weniger restriktiv sind. Es sollte aber deutlich werden, dass es - auf Basis der Ideen aus der Einleitung und ohne die Systemarchitektur vorweg zu nehmen - im vorgestellten System nötig ist, sowohl eine invariante Komponente als auch eine temporäre Komponente zu berücksichtigen, die jedoch beide - zumindest zeitweise - persistent abgelegt werden sollten. Die konkrete Umsetzung ist in Abschnitt 3.1 beschrieben.

2.1.3 OLTP und OLAP

Wie in obigem Abschnitt zur Data Warehouse-Architektur beziehungsweise dem Operational Data Store bereits angeklungen ist, sind die meisten Begriffe aus dem Umfeld der Datenanalyse betriebswirtschaftlich motiviert. Dort unterscheidet man im allgemeinen eben die operationalen Datenbanken, beispielsweise die Kassensysteme eines Verkaufsgeschäfts, wo jeder Verkauf in einem eigenen Datensatz gesichert wird, und die aggregierte, integrierte Sicht auf diese Daten, wo beispielsweise alle Verkäufe eines Produkts an einem Tag zu einem Datensatz zusammengefasst werden. Stellt man sich die schiere Menge an entstandenen Daten in den operationalen Datenbanken für eine große Einkaufskette, die gegebenenfalls sogar international vertreten ist, vor, wird schnell deutlich, warum in diesem Umfeld im Allgemeinen (vgl. [KE01]) der Konsens besteht, dass die Analyse-Vorgänge nicht auf den gleichen Daten und schon gar nicht in den operationalen Datenbanken durchführt werden sollten. In der Literatur wird hier zwischen *OLTP*, Online Transaction Processing, und *OLAP*, Online Analytical Processing, gesprochen. Ersteres ist also im Wesentlichen die Verbuchung einzelner Aktionen, häufig eben Ver- und Einkäufe, und letzteres das Analysieren dieser Einzel-Aktionen in einem häufig globaleren Kontext. Um diesen globaleren Kontext überhaupt herzustellen oder Aussagen über ein komplettes Unternehmen

treffen zu können, deren Daten auf die vielen operationalen Datenbanken verteilt sind, braucht es eben ein Data Warehouse, einen Operational Data Store oder etwas vergleichbares.

Möchte man nun das in der Einleitung skizzierte System von Nuntio in diesen Kontext einordnen, so existieren hier eben auch beide Seiten. Die operationalen Datenbanken wären in diesem Fall die an die Web-Server angebundenen, die beispielsweise Aktionen wie "Nutzer A möchte den Nachrichten-Feed B abonnieren." oder aber "Nutzer C hat einen Kommentar zu einem Artikel D abgegeben." erfassen - "Transaction Processing" also. Außerdem wird aus diesen Systemen für die einzelnen Nutzer ihre persönliche Tageszeitung generiert. Um dies jedoch tun zu können, muss ja bereits analysiert worden sein, welche der Nachrichten, die für diese spezielle Tageszeitung in Frage kommen, angezeigt werden sollen - beispielsweise auf Basis des Interesses eines Nutzers. Das Interesse eines Nutzers auszuwerten ist jedoch ein globalerer Prozess. Ein Analyse-Prozess, der unabhängig von der Anfrage nach der gerade aktuellen Tagessicht auf die Nachrichten-Meldungen erfolgen muss. Dieser Prozess wäre dann dem "Analytical Processing" zuzuschreiben, der - wie im obigen Abschnitt beschrieben - eben nicht auf den einzelnen operationalen Datenbanken ausgeführt werden sollte, sondern in einem in Abschnitt 3.1 noch zu skizzierenden, Data Warehouse-ähnlichen System.

2.1.4 MySQL

Die Tatsache, dass für das System ein persistenter Speicher, in dem nicht nur die aggregierten Online-Nachrichten, sondern auch die Ergebnisse der Analyseprozesse abgelegt werden sollen, nötig ist, wurde bereits im vorherigen Abschnitt erläutert. Auch in diesem Abschnitt sollen die Grundkonzepte und Vorteile von Datenbank-Systemen (eine gute Einführung bietet etwa [KE01]) beispielsweise gegenüber der dateibasierten Datenhaltung als bekannt vorausgesetzt werden. Trotzdem soll an dieser Stelle zumindest kurz auf das für das Projekt gewählte Datenbanksystem MySQL eingegangen werden.

MySQL ist derzeit die am häufigsten eingesetzte OpenSource-Datenbank der Welt[2] und bietet eine Vielzahl an Funktionen und Vorteilen, die an dieser Stelle jedoch nicht umfassend besprochen werden sollen. Einige der Argumente lassen sich auf der MySQL-Webseite[3] nachlesen. Drei davon wurden bei der Wahl für Nuntio als besonders wichtig eingestuft.

Geringe Betriebskosten MySQL selbst unterliegt unter anderem der General Public License[4] und kann folglich auch in kommerziellen Umgebungen kostenlos genutzt werden. Weiterhin ist das System im Internet auch im Rahmen des so genannten LAMP-Technologie-Stacks, also Linux mit Apache, MySQL und PHP, extrem weit verbreitet. Schließlich arbeitet auch das für das Projekt genutzte Framework Ruby on Rails von Haus aus mit dieser Datenbank zusammen, so dass der Verwaltungs- und Konfigurationsaufwand während der Entwicklung sehr gering gehalten werden

[2] http://www.mysql.de/why-mysql/marketshare/ - Abruf 05.11.2010
[3] http://www.mysql.de/why-mysql/topreasons.html - Abruf 05.11.2010
[4] http://www.gnu.org/licenses/gpl-3.0.html - Abruf 05.11.2010

konnte, aber auch potentielle Folgekosten bei einem weiteren Betrieb des Systems minimiert werden können.

Transaktionsunterstützung und ACID Da in der Einleitung bereits beschrieben wurde, dass es sich bei dem angestrebten System und ein Mehrbenutzer-System handeln soll, ist es wichtig, dass das zugrundliegende Datenbanksystem auch eine Transaktionsunterstützung vorsieht. Bei MySQL ist dies mit der Speicher-Engine *InnoDB* der Fall, wobei die ACID-Kriterien - also *Atomicity*, *Consistency*, *Isolation* und *Durability* (vgl. beispielsweise [KE01] S.269) - vollständig unterstützt werden. Gerade im Hinblick auf die spätere Parallelisierung der Analyse-Prozesse stellt sich dies als enorm wichtiges Kriterium heraus. Zudem unterstützt MySQL mit *InnoDB* das Erzwingen einer ständigen referentiellen Integrität der Daten, was für die ebenso in Abschnitt 3.1 dargestellte, verteilte Architektur eine wichtige Voraussetzung ist.

Performance und Skalierbarkeit Eine hohe Performance ist vermutlich in jedem Software-System eine wünschenswerte Eigenschaft. Betrachtet man aber die Idee, dass viele Nutzer eine ganz persönliche Auswahl von Datenquellen angeben können, die später die Grundlage für die Analyse-Prozesse darstellen, wird schnell klar, dass die Leistungsanforderungen mit jedem Nutzer weiter steigen. MySQL bietet hier zunächst den Vorteil, dass das Datenbank-System Mehrkern-Prozessoren unterstützt und Anfragen so parallel bewältigen kann. Außerdem bietet MySQL selbst ein separates Produkt für den Parallelbetrieb mehrerer Datenbank-Server, "MySQL Cluster", sowie Unterstützung für verschiedene Datenbank-Replikationsverfahren, die im nächsten Abschnitt näher beschrieben werden sollen.

2.1.5 Replikation von Datenbanken und Datenbank-Cluster

Die Wichtigkeit der Skalierbarkeit des Datenbank-Systems wurde im vorherigen Abschnitt bereits angesprochen. Gerade in einem Umfeld, in dem Nutzer für die anfallenden Datenmengen und im Fall von Nuntio sogar für die anfallende Analyse-Arbeit verantwortlich sind, sollte möglichst früh eine Strategie zur Anpassung an erhöhte Anforderungen feststehen. MySQL bietet hier im wesentlichen drei Konzepte: Das separate Produkt "MySQL Cluster", eine Master-Slave-Replikation oder eine Master-Master-Replikation.

MySQL Cluster Grundsätzlich gibt es verschiedene Formen von Datenbank-Clustern und verteilten Datenbanken (vgl. auch [KE01]), wobei MySQL Cluster eine so genannte Shared-Nothing-Architektur implementiert[5]. Dabei werden die Daten innerhalb des Clusters partitioniert und auf die einzelnen Knoten verteilt, die Anfragen unabhängig voneinander bearbeiten können. Durch das Hinzufügen weiterer Knoten kann die Leistung so relativ einfach und kostengünstig erhöht werden. Durch den hohen Grad der Parallelisierung kann auf diese Weise eine sehr hohe Lese- und Schreibrate erzielt werden. Da die Daten bei der Partitionierung jedoch über mehrere

[5] vgl. auch http://www.mysql.de/products/cluster/faq.html - Abruf 05.11.2010

Knoten verteilt vorliegen, ist das System für Abfragen, die komplexe Joins über mehrere Tabellen beinhalten, oder für Abfragen, die sich auf komplette Tabellen beziehen, weniger geeignet.

Master-Slave-Replikation Ein weitere Form der Verteilung von Daten ist die so genannte Master-Slave-Replikation. Dabei wird ein Datenbank-Server als Master gekennzeichnet, an den eine beliebige Anzahl von Slaves angebunden werden können. Der Master-Server in der MySQL-Implementierung (vgl. auch [Ora10]) verwaltet dabei alle verfügbaren Daten und führt über die auf den Daten vorgenommenen Änderungen eine binäre Logdatei. Diese Log-Einträge werden dann an alle Slaves gesendet, die die darin enthaltenen Aktualisierungen ihrerseits auf ihrem eigenen Datenbestand durchführen. Durch eine Nummerierung der Log-Einträge entsteht dabei kein Problem, sollte ein Slave oder gar der Master zeitweise die Netzwerkverbindung verlieren. Nach Wiederherstellung werden die Log-Einträge gesendet beziehungsweise empfangen, die noch nicht im Datenbestand berücksichtig sind, so dass nach einer kurzen Wiederanlaufphase die Synchronisation der Daten wiederhergestellt ist. Ziel der Architektur ist es, mit einem asynchronen Verfahren alle Daten vom Master-Server auf die Slaves zu replizieren, so dass - ohne den Versatz der Synchronisation zu berücksichtigen - stets auf allen Klienten die gleichen Datenbestände zur Verfügung stehen. Bei dieser Technik ist es jedoch wichtig, dass Datenänderungen nur auf dem Master-Server ausgeführt werden dürfen. Mit anderen Worten dürfen die Slaves nur für lesende Operationen genutzt werden. Das klingt zwar zunächst nach einem großen Nachteil, häufig ist es aber so, dass das Gros der Anfragen an Datenbanken, insbesondere bei Web-Applikationen, Lesezugriffe sind. Gerade in einem System, das extrem viele und gegebenenfalls auch komplexe Leseoperationen durchführen muss, erreicht man mit dieser Technik eine außerordentliche Skalierbarkeit mit extrem wenig Aufwand, da auf jedem Slave, parallel zu allen anderen, Analysen durchgeführt oder Anfragen aus der Webapplikation beantwortet werden können. Lediglich Analyse-Ergebnisse müssen im Anschluss an den Master gesendet werden, wenn diese persistent abgelegt werden sollen. Abbildung 2.2 stellt die System-Architektur bei der Master-Slave-Replikation dar, wobei der gerichtete Kommunikationspfad verdeutlichen soll, dass die geänderten Daten vom Master an die Slaves verteilt werden.

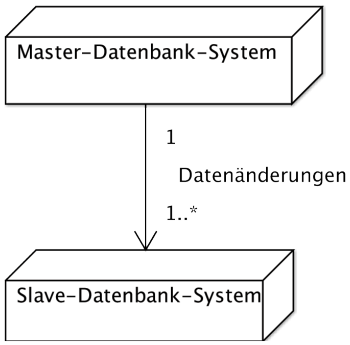


Abbildung 2.2: Architektur bei der Master-Slave-Replikation in UML-Notation

Master-Master-Replikation Analog zur Master-Slave-Replikation besteht in MySQL auch die Möglichkeit, statt einem Master und vielen Slaves mehrere gleichberechtigte Master-Server miteinander zu verbinden. Die Idee ist hierbei die gleiche, allerdings sind nun Datenänderungen auf allen Master-Servern erlaubt, die diese untereinander propagieren. Auf diese Weise lässt sich der Nachteil, dass bei der Master-Slave-Replikation Datenänderungsoperationen nur auf dem Master-Server durchgeführt werden dürfen, umgehen. Problematisch ist hierbei jedoch, dass die Datenverteilung wiederum asynchron erfolgt, so dass - sollen die ACID-Kriterien auch über den gesamten Server-Verbund gelten - ein separater Transaktionsmechanismus eingeführt werden müsste. Um dieses Problem zu umgehen, ist es ratsam in solchen Szenarios zu überlegen, ob nicht die unproblematischere Master-Slave-Replikation genügt oder besser ein Cluster eingesetzt werden kann, wo die Datenverteilung synchron, also auch transaktionsgeschützt, erfolgt.

2.2 Ruby on Rails

Nachdem nun einige Grundlagen der verwendeten Datenbank-Konzepte und -Technologien erläutert wurden, soll in diesem Abschnitt die genutzte Programmiersprache Ruby und das Framework Ruby on Rails oder kurz Rails vorgestellt werden.

2.2.1 Ruby

Ruby ist eine im Jahr 1995 erschienene, in Japan von Yukihiro Matsumoto entwickelte Programmiersprache, die verschiedenste Konzepte und Paradigmen unterstützt. Die Sprache ist interpretiert, plattformunabhängig und ist seit ihrer Erfindung - auch Dank der späteren Veröffentlichung des in Abschnitt 2.2.2 vorgestellten Rails-Frameworks - immer bekannter geworden, so dass sie im Oktober 2010 immerhin auf Platz 10 der populärsten Programmiersprachen laut TIOBE-Index[6] vertreten ist. An dieser Stelle sollen einige der Grundkonzepte der Sprache vorgestellt und anhand von kurzen Code-Beispielen verdeutlicht werden. Einen umfassenderen Überblick bieten beispielsweise [Mat02] vom Autor der Sprache, [Tho05] mit einem ausführlichen, anwendungsorientierten Ansatz sowie die Kurzreferenz [Fit07].

Das vielleicht wichtigste Merkmal der Sprache ist ihre grundlegende Objektorientiertheit. Alles in Ruby ist ausnahmslos ein Objekt. Eine einfache Klassendefinition kann dabei wie in Listing 2.1 aussehen.

[6] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html - Abruf am 05.11.2010

```
1  class Admin < User

3    def name
4      @name
5    end

7    def name=(new_name)
8      @name = new_name
9    end

11 end
```

Listing 2.1: Einfache Ruby-Klassendefinition (verkürzt)

Die Klassendefinition mit `class Admin < User` dürfte dabei noch bekannt vorkommen, wobei `Admin` hier der Name der Klasse ist und durch das <-Zeichen angedeutet wird, dass diese Klasse von der Oberklasse User erbt. Ähnlich wie in Java erbt grundsätzlich jede Klasse - implizit oder explizit - von der Oberklasse `Object`. Man erkennt außerdem, dass Methodendefinitionen mit dem Schlüsselwort `def` eingeleitet werden, gefolgt vom Namen der Methode und gegebenenfalls einer Parameterliste. Auffällig ist dabei, dass keine Typenbezeichnungen angegeben sind, was darauf zurückzuführen ist, dass Ruby eine dynamisch typisierte Sprache ist, also das so genannte *Duck-Typing* nutzt. Weiterhin fällt auf, dass keine `return`-Anweisung angegeben ist, obwohl die Methode `name` den Namen eines `User`s zurückliefern soll. Das liegt daran, dass jede Methode implizit das zurückliefert, was ihr letzter Ausdruck zurückgeliefert hat. Daraus folgt, dass jeder Ausdruck einen Wert hat, was wiederum funktionale Programmierung begünstigt. Außerdem erkennt man, dass Instanz- oder Member-Variablen mit einem `@`-Zeichen beginnen und nicht separat deklariert werden müssen. Mit dieser Klasse könnte man wie in Listing 2.2 umgehen.

```
1  my_admin = Admin.new

3  my_admin.name = 'Friedhelm'
4  my_admin.name=('Friedhelm')

6  puts my_admin.name
```

Listing 2.2: Umgang mit Objekten

Die Klassen-Methode `new` erzeugt eine neue Instanz, an der sich dann die definierten Methoden aufrufen lassen. Eine Besonderheit erkennt man nun am oben bereits beschriebenen Setter für den Namen, da Zeile 3 wie Zeile 4 äquivalent sind und die gleiche Methode aufrufen. Hier klingt bereits an, dass die Nutzung der runden Klammern, wie beispielsweise in Java bei Methodenaufrufen verpflichtend, nicht immer nötig ist und häufig weggelassen wird. So ist beispielsweise der Aufruf `my_array.include?(42)` äquivalent zu `my_array.include? 42`, wobei in beiden Fällen die Methode mit dem Kopf `def include?(value)` aufgerufen wird. Obiger Code stellt dabei - vorausgesetzt die Klasse Admin ist bekannt - ein gültiges Ruby-Programm dar, da die Sprache kontextlosen Quellcode innerhalb eines dafür automatisch angelegten `main`-Objekts ausführt. Auf diese Weise ist auch prozedurale Programmierung mit Ruby möglich.

Eine weitere Besonderheit in Ruby ist das häufig genutzte Übergeben von Code-Blöcken an Methoden, wie es beispielsweise auch in JavaScript möglich ist. Insbesondere beim Iterieren über Mengen-Datentypen kommt diese in Listing 2.3 dargestellte Technik häufig zum Einsatz.

```
1 my_array.each do |element|
2   puts element.to_s
3 end
```

Listing 2.3: Iterieren mit Code-Blöcken

Die Methode `each` des Objekts vom Typ `Array` bekommt in Listing 2.3 einen Code-Block übergeben, der bei dessen Ausführung genau einen Parameter mit dem lokalen Namen `element` erwartet. In der Methode wird nun für jedes Element des Arrays einmal der Code-Block ausgeführt, wobei als Parameter eben genau das aktuelle Element übergeben wird. Der angegebene Block gibt das Element als String interpretiert - mit der `Object`-Methode `to_s` - auf der Standardausgabe aus.

Dieser kurze Überblick über die Syntax soll zunächst ausreichen, um die späteren Beispiele der Implementierung nachvollziehen zu können. Bevor nun das auf die Sprache aufbauende Framework Rails im nächsten Abschnitt vorgestellt wird, soll insbesondere im Hinblick auf dieses ein letztes Konzept nicht unerwähnt bleiben: Das der Meta-Programmierung oder des "dynamic programming" [Mat02]. Dabei handelt es sich um die Möglichkeit eines Programms sich selbst oder Teile von sich zur Laufzeit zu modifizieren oder zu ergänzen. Eine abgeschwächte Form hiervon ist unter dem Namen Reflection oder Introspection bekannt. Listing 2.4 skizziert das Vorgehen.

```
1 my_admin.instance_variables
2 my_admin.method(:name=).arity

4 my_admin.class.send(:define_method, "say_hello") { puts "hello!" }
```

Listing 2.4: Reflection und Meta-Programmierung in Ruby

Zeile 1 und 2 zeigen dabei einfach Beispiele für Reflection. Dabei können im ersten Beispiel zur Laufzeit alle Instanzvariablen eines Objekts abgefragt werden, im zweiten Beispiel wird zunächst nach dem Methoden-Objekt - auch Methoden sind Objekte - der Methode `name=` (siehe oben) gefragt und an diesem Objekt wird die Methode `arity` aufgerufen, die die Anzahl der erwarteten Parameter zurück gibt. In obigen Fall wäre das eine 1, da der Parameter `new_name` von der Methode erwartet wird. Die Meta-Programmierung geht nun aber noch einen Schritt weiter, als Informationen über den aktuellen Zustand von Klassen und Objekten zur Verfügung zu stellen - sie erlaubt es, diese Strukturen auch dynamisch zu ändern. In Zeile 4 wird dazu beispielsweise eine neue Methode `say_hello`, die auf der Standardausgabe `hello!` ausgibt, an die bereits vorhandenen Objekte angehängt. Auf diese Weise ist es möglich, extrem dynamische Programme zu schaffen, was das im Folgenden vorgestellte Framework Rails ausnutzt.

2.2.2 Rails

Rails oder in Langform Ruby on Rails ist das Framework, das für die Implementierung von Nuntio genutzt wurde. Es wurde erstmals im Jahr 2004 von David Heinemeier Hansson vorgestellt und Ende August 2010 in der hier genutzten Version 3.0 veröffentlicht. Wie im vorherigen Abschnitt bereits erwähnt wurde, ist das Framework in Ruby geschrieben und speziell auf die Anforderungen von datenbankgestützten Web-Applikationen ausgerichtet. Dem Framework liegen dabei drei fundamentale Konzepte zu Grunde.

Don't Repeat Yourself Das Prinzip DRY oder in Langform eben "Don't Repeat Yourself" ist dabei das allgemeinste Konzept, das für nahezu alle Formen des Programmierens wünschenswert ist: Die Forderung, Redundanz beim Programmieren zu verringern oder ganz zu vermeiden. Die Idee dieses Konzepts ist es, dass beim Programmieren häufig ähnliche Aufgaben ausgeführt werden müssen, wobei man als Programmierer dazu neigen könnte, den Quellcode, der eine ähnliche Aufgabe bereits bewältigt, zu kopieren und nur an wenigen Stellen an die neue Aufgabe anzupassen. Dieses Vorgehen spart zwar zunächst Arbeit, wird aber spätestens dann problematisch, wenn später an allen Aufgaben etwas geändert werden soll. In diesem Fall muss die Änderung auch in jeder der Kopien eingepflegt werden. Dem DRY-Prinzip folgend, würde in einem solchen Fall häufig eine abstraktere Variante der Ursprungsaufgabe implementiert werden, die mit zwei verschiedenen Aufrufen - beispielsweise veränderte Parameter - jeweils die gewünschte, spezielle Aufgabe erledigt. Änderungen sind nun kein Problem mehr, da nur die neue Aufgaben-Abstraktion an *einer* Stelle geändert werden muss, was sich daraufhin auf alle davon abgeleiteten Aufgaben auswirkt.

Ein einfaches Beispiel aus der Implementierung von Nuntio ist die Berechnung eines *User-Ratings*. Dabei werden als Funktions-Parameter Für- und Gegenstimmen erwartet und ein Wert zwischen 0 und 2 auf Basis der Eingabe ermittelt. In den späteren Algorithmen dienen diese Werte zur Gewichtung von Daten-Verknüpfungen. Während der Entwicklung von Nuntio wurde die Funktion, die aus den beiden Parametern einen skalaren Wert ermittelt, mehrfach angepasst. Die Methode selbst wird aber in mehreren separaten Skripten benötigt. Wegen der Einhaltung des DRY-Prinzips wurde die Methode aber an einer zentralen Stelle abgelegt und dann aus den separaten Skripten aufgerufen. Es wäre dabei einfacher gewesen, die Methodendeklaration von einer Skript-Datei in die nächste zu kopieren, was - nach den jeweiligen Änderungen - im schlimmsten Fall sogar bedeutet hätte, dass in verschiedenen Programmteilen mit verschiedenen Berechnungsgrundlagen gearbeitet würde, wären nicht alle Versionen der Methode korrekt konsolidiert worden.

Convention over Configuration Im Gegensatz zum DRY-Prinzip ist das Convention over Configuration-Konzept durchaus kontrovers zu betrachten. Die Idee hinter diesem Paradigma ist, die Menge an nötiger Konfiguration durch das Einführen von Konventionen zu verringern, dabei jedoch nach Möglichkeit nicht zu viel Flexibilität einzubüßen. In Ruby on Rails ist dies Paradigma extrem ausgeprägt, was es ermöglicht - bei Kenntnis der Konventionen - sehr schnell

auch größere Software-Projekte umzusetzen, da eben der initiale Konfigurations-Aufwand wegfällt. Problematisch ist dieses Konzept genau dann, wenn man die einzelnen Konventionen nicht kennt.

Ein konkretes Beispiel aus Ruby on Rails ist die Handhabung der Benennung von Tabellen beziehungsweise Entitäten. Als Beispiel soll hier die Entität "Student" gewählt werden. Sollen Ausprägungen dieser in einer Datenbank abgelegt werden, besagt die Konvention, dass die entsprechende Tabelle `students` heißen muss. `students` ist hier klein geschrieben und besteht aus dem englischen Plural des Namens der Entität. Die zugehörige Klasse in Rails hätte den Namen `Student`, also groß geschrieben und im Singular. Sollen die Entität "Student" und die weitere hypothetische Entität "Lecture" durch eine N:M-Relation miteinander verknüpft werden, muss dazu in Rails kein Konfigurationsaufwand betrieben werden, hält man sich an die Konvention, dass die Join-Tabelle den Namen `lectures_students` tragen muss. In den meisten Fällen sind die Rails-Konventionen sehr eingängig und helfen dabei, Software schneller und mit weniger unnötigem Aufwand zu entwickeln. Andrerseits ist es auch notwendig, sich zunächst alle Konventionen anzueignen, bevor man einen Nutzen daraus ziehen kann. Problematisch - und deshalb auch den Hinweis auf die Kontroverse um dieses Konzept - ist, dass es manchmal unverhältnismäßig aufwändig ist, bestimmte Konventionen zu umgehen oder an die eigenen Wünsche anzupassen.

Model View Controller Neben diesen eher konzeptuellen Prinzipien setzt das Rails-Framework in seiner Implementierung auf eines der bekanntesten und meist genutzten Software-Design-Patterns - das Model View Controller-Pattern. Hierbei wird das Programm nach Funktion in drei Teile geteilt. In Abbildung 2.3 ist die Aufteilung und die Kommunikation der einzelnen Teile im Kontext einer Rails-Applikation skizziert.

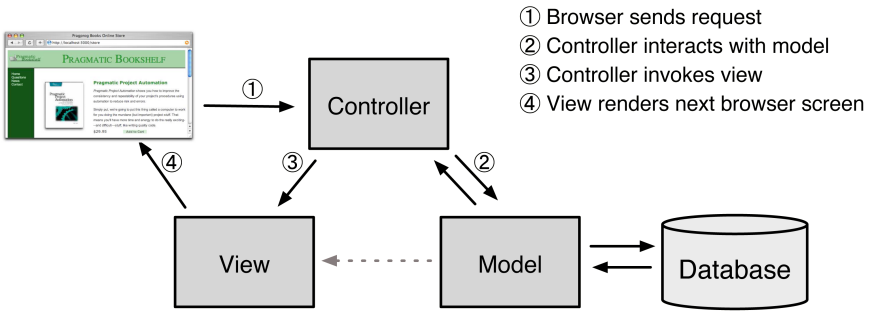


Abbildung 2.3: Rails MVC-Architektur (aus: [RTH09] S.12)

Sendet der Browser einen Request an den Applikations-Server, wird dieser in einem passenden Controller bearbeitet. Passend bedeutet hierbei, dass für verschiedene Aktionen verschiedene Controller genutzt werden können und auch sollten. In Nuntio ist beispielsweise der Teil des Systems, der die individuelle Tageszeitung zusammenstellt, in einem anderen Controller vertreten als der Teil, der für das Hinzufügen neuer Abonnements verantwortlich ist. Der Controller ist nun für die Verarbeitung des User-Requests verantwortlich und wird - sofern es sich um eine

datenbankgestützte Webapplikation handelt - Applikationsdaten aus der Datenbank lesend oder auch schreibend zugreifen müssen. Der Teil des Systems, der für die Verwaltung der Daten und damit letztlich der Kommunikation mit der Datenbank verantwortlich ist, wird dabei dem Model zugeschrieben. In Rails kann dazu ein so genanntes *ORM*, ein Object-Relational-Mapping, als Model-Schicht genutzt werden - dazu unten mehr. Sind die notwendigen Datenoperationen, die der Controller initiiert hat, durchgeführt, muss eine Antwort auf den Request gesendet werden. Im Allgemeinen muss also eine HTML-Seite generiert werden, die die Ergebnisse aus den geforderten Operationen des Requests enthält. Um die Applikationslogik, die ja nun im Controller gehalten wird, nicht mit der Daten*darstellung* vermischen zu müssen, ist diese Komponente hierfür wiederum vom Rest der Applikation getrennt. Zur Darstellung von Daten stellt Rails dazu ein dynamisches Template-System zur Verfügung. Ist die View mit den entsprechenden Daten gefüllt, kann diese beziehungsweise ihr Quelltext schließlich als Antwort auf den ursprünglichen Request des Benutzers zurück geliefert werden.

Ein konkretes Beispiel aus Nuntio sieht dabei so aus, dass ein Nutzer eine Seite anzeigen lassen kann, auf der alle seine Freunde aufgelistet werden (siehe auch Abschnitt 3.5.4). Wird der entsprechende Request gesendet, spricht Rails den passenden Controller für diese Aktion beziehungsweise Anfrage an. Danach werden aus dem zugehörigen Model alle Freunde des betreffenden Nutzers aus der Datenbank angefragt. Die zurückgelieferte Liste wird dann in einem passenden Template verarbeitet, das den grundlegenden Seitenaufbau beinhaltet, so dass nur noch entsprechende Platzhalter mit den Namen der Freunde gefüllt werden müssen. Die auf diese Weise dynamisch erzeugte Webseite wird schließlich an den Nutzer zurückgeliefert.

ORM mit Active Record Eines der wichtigsten Bestandteile des Rails-Frameworks ist das Active Record-Modul, das für das Objekt-Relationale-Mapping und den Zugriff auf die Datenbank zuständig ist. Jede Webapplikation, die für ihre Datenhaltung auf eine relationale Datenbank im Hintergrund setzt, steht dabei vor dem Problem, die - wahrscheinlich - objektorientierte Welt der eigentlichen Applikation mit der relationalen der Datenbank zu verknüpfen. Ohne ein entsprechendes Mapping müssten Antworten auf SQL-Anfragen als allgemeine, Array-ähnliche Strukturen aufgefasst werden - Vorteile der Objektorientierung würden also verloren gehen. Rails bietet ein sehr umfangreiches Modul zum Umgang mit den relationen Daten der Datenbank und stellt dies - dank Convention over Configuration - mit extrem wenig Konfigurations-Aufwand zur Verfügung.

Beim Mapping der relationalen Tabellendaten wird in Active Record jede Tabelle, also jede Entität, als Klasse aufgefasst, jede Zeile der Tabelle als Objekt dieser Klasse und jede Spalte einer Zeile als Attribut dieses Objekts. Die Rails-Konventionen besagen dabei, dass eine Tabelle - wie oben schon beschrieben - den gleichen Namen wie die Klasse jedoch im Plural und kleingeschrieben tragen muss. In Abbildung 2.4 ist dazu ein Ausschnitt aus dem ER-Diagramm zu Nuntio dargestellt. Es sind zwei Entitäten, `User` und `Feed` mit jeweils drei Attributen zu sehen. Man erkennt, dass - eine weitere Rails-Konvention - jede Entität einen synthetischen Schlüssel `id` haben muss. Weiterhin ist zwischen den beiden Entitäten eine N:M-Relation modelliert. In Nuntio kann ein `User` also eine beliebige Anzahl an `Feed`s abonnieren, während ein `Feed` von einer belieben Anzahl von `User`n abonniert werden kann. Die Übersetzung des ER-Diagramms

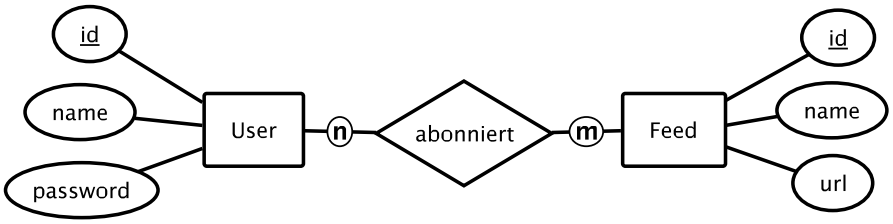


Abbildung 2.4: Beispiel ER-Diagramm aus Nuntio

in die relationale Datenbank muss also zu drei Tabellen führen. In der ersten werden alle `User` abgelegt, in der zweiten alle `Feed`s und die dritte ist als Join-Tabelle nötig, wird also in jeder Zeile einen Verweis auf den `User` und den `Feed` enthalten, die miteinander verknüpft sein sollen.

Soll nun das objektrelationale Mapping von Rails genutzt werden, reicht es im Model-Verzeichnis für jede der Entitäten eine eigene Klasse anzulegen, die den Namen der Entität trägt beziehungsweise in Rückrichtung zu obiger Konvention den Namen der Tabelle im Singular und großgeschrieben. Die angelegten Klassen müssen - um das automatische Mapping nutzen zu können - von der abstrakten Basisklasse `ActiveRecord::Base` erben. Lässt man die Assoziation der beiden Entitäten für den Moment außen vor, ist damit bereits alles Nötige getan, um die Objekte der erstellten Klassen nutzen zu können. In Listing 2.5 ist dazu ein Beispiel für den Umgang mit den Active Record-Klassen skizziert.

```
1 my_user = User.new

3 my_user.name = "Nicolas"
4 my_user.password = "geheim"

6 my_user.save

8 puts my_user.id.to_s
```

Listing 2.5: Umgang mit Active Record-Objekten

Wie oben bereits beschrieben reicht es für Rails aus, dass die erstellten Klassen den gleichen Namen tragen wie die korrespondierenden Tabellen. Bei der ersten Nutzung der entsprechenden Objekte wie in Zeile 1, wo ein neues `User`-Objekt instantiiert werden soll, erkundigt sich die Oberklasse `ActiveRecord::Base` zunächst bei der Datenbank über die Struktur der zugehörigen Tabelle, eine Query der Form `DESCRIBE users;` wird abgesetzt. Auf diese Weise erkennt Rails selbstständig und zur Laufzeit die Attribute der Klasse beziehungsweise der Objekte. Wegen der Möglichkeit zur Meta-Programmierung (siehe auch Abschnitt 2.2.1) ist Rails nun in der Lage, die Klasse der Entität dynamisch um eben diese Attribute zu erweitern sowie Getter- und Setter-Methoden zu synthetisieren. Deswegen ist es in Zeile 3 auch möglich, auf die Set-Methode des

Attributs `name` zuzugreifen und so das Attribut zu verändern. Rails verfolgt dabei die Strategie, neue Objekte nach dem Instantiieren so lange lokal im Speicher zu lassen, bis die Methode `save` aufgerufen wird. Da es sich in obigem Beispiel um ein neues Objekt handelt, wird in Zeile 6 also ein passendes `INSERT`-SQL-Statement erzeugt, dass die gesetzten Attribute persistent in die Datenbank schreibt. Bei Objekten, die bereits aus der Datenbank gelesen wurden, wird hier entsprechend ein `UPDATE`-Statement an die Datenbank gesendet. Wie bereits an diesem einfachen Beispiel deutlich wird, ist das objektrelationale Mapping in Rails ein sehr einfach zu nutzendes Werkzeug, dass dem Programmierer durch das automatische Zurverfügungstellen der grundlegenden *CRUD*-Operationen - also *Create*, *Read*, *Update* und *Delete* - viel Arbeit abnimmt und zudem extrem einfach zu handhaben ist.

Active Record bietet neben diesen einfachen Operationen aber noch einen deutlich größeren Funktionsumfang. Beispielsweise können auch Assoziationen, wie in obigem Beispiel zwischen den `Usern` und `Feeds`, einfach modelliert und in Objekt-Beziehungen überführt werden. Bei der N:M-Relation handelt es sich dabei schon um eine der komplexeren, da sie eine eigene Join-Tabelle erfordert.

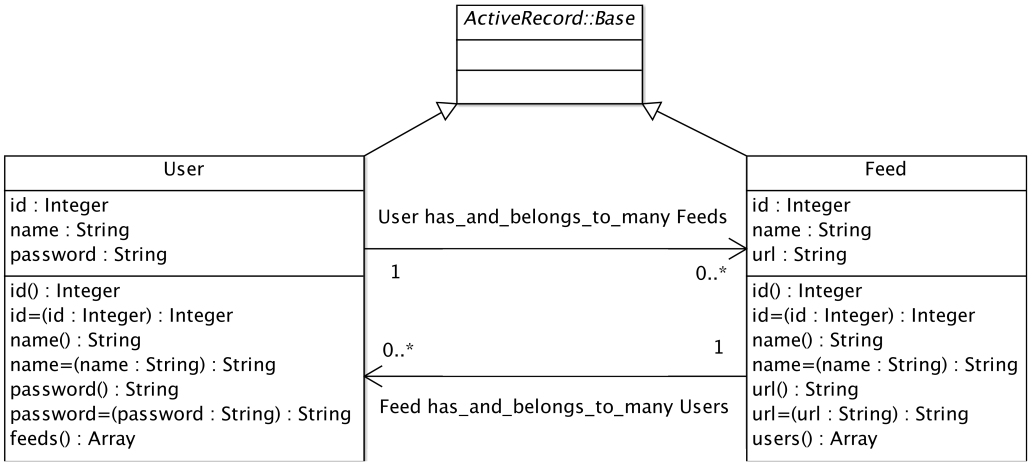


Abbildung 2.5: ORM in Rails mit Active Record (ausschnittshaft) in UML-Notation

Abbildung 2.5 zeigt den von Active Record zur Laufzeit synthetisierten Aufbau der Klassen für obiges Beispiel (vgl. auch mit Abbildung 2.4). Die N:M-Relation ist hier in zwei 1:N-Assoziationen aufgegangen, für die - genau wie für die Attribute - Hilfsmethoden erstellt wurden. Rails erkennt hierbei aber nicht selbstständig, dass die beiden Entitäten assoziiert sind, auch nicht wenn die Konventionen eingehalten wurden. Listing 2.6 zeigt deshalb den Aufbau den `User`-Klasse beispielhaft.

```
1 class User < ActiveRecord::Base

3   has_and_belongs_to_many :feeds

5 end
```

Listing 2.6: Eine Rails-Model-Klasse

Wie man in Zeile 3 sieht, ist es also nötig, die Assoziation der beiden Objekte innerhalb der Klassen anzukündigen. Im Fall einer N:M-Relation dient hierzu der Aufruf `has_and_belongs_to_many` mit dem Namen der assoziierten Klasse. Sind die Rails-Konventionen eingehalten, erwartet Rails nun die Join-Tabelle `feeds_users` mit den Attributen `feed_id` und `user_id`, die jeweils auf die beiden verknüpften Objekte hinweisen. Um einen objektorientierten Zugriff auf die assoziierten Objekte zuzulassen, werden weitere Hilfsmethoden synthetisiert. So etwa für `User` die Methode `feeds`, die ein Array mit eben den `Feed`-Objekten zurückliefert, die ein Nutzer abonniert hat. Ist die Assoziation in der Klasse `Feed` ebenso angekündigt, werden hier analog Methoden für den Zugriff auf die `User`-Objekte erzeugt. Die zugehörigen SQL-Abfragen werden wiederum innerhalb von Active Record automatisch generiert, so dass der Programmierer auch hier ohne den Einsatz eigener Queries auskommt.

Neben der ORM-Schicht bietet Active Record auch ein umfangreiches Interface zur Abfrage von Daten. Einige Möglichkeiten sind dabei in Listing 2.7 dargestellt.

```
1 all_users = User.all
2 user1 = User.find(1)
3 user_nicolas = User.where(:name => 'Nicolas').first
4 two_feeds = Feed.where(:name => ['feed1', 'feed2']).order('id DESC').includes(:users)
```

Listing 2.7: Active Record-Query-Interface-Beispiele

Die Methode `all` von Active Record fragt schlicht alle Zeilen der zugehörigen Tabelle ab und liefert ein Array von gemappten Objekten zurück, wird also in die Query `SELECT * FROM users;` übersetzt. Die Methode `find` hingegen erwartet einen Wert für den Primärschlüssel der betreffenden Tabelle, um nur den einen passenden Eintrag zurückzuliefern. Die zugehörige Query lautet also etwa `SELECT * FROM users WHERE id = 1;`. Im dritten Aufruf wird die Komplexität des Query-Interfaces deutlich. Über die Methode `where`, die eine Liste von Bedingungen als Parameter erwartet, wird der `WHERE`-Teil einer SQL-Query nachgebildet. In obigem Fall würde etwa die Query `SELECT * FROM users WHERE name = "Nicolas";` erzeugt und ausgeführt. Da das Attribute `name` nicht der Primärschlüssel und auch nicht `unique` ist, wird hier ein Array von `User`-Objekten zurückgegeben. Die auf das Array angewandte Methode `first`, gibt dabei den ersten Datensatz des Arrays zurück. Die vierte Abfrage soll einen letzten Einblick in die Komplexität von Active Record geben. Über die Methode `where` wird hier ein `IN`-Teil einer Query erzeugt, da das Attribut `name` diesmal auf ein Array von mehreren Optionen gemappt wird. Außerdem sieht man, dass verschiedene, die Query beeinflussende Methoden hintereinander geschachtelt werden können. So sorgt die `order`-Methode für einen zu erzeugenden `ORDER BY`-Teil in der resultierenden Query. Schließlich folgt der einfach anmutende `includes`-Befehl, der einen automatischen Join von `Feed`s und `User`s über die entsprechende Join-Tabelle innerhalb der Que-

ry auslöst. Zurückgeliefert wird also ein Array von `Feed`-Objekten, deren `name` *feed1* oder *feed2* ist, absteigend nach dem Wert ihrer `id` sortiert. Außerdem ist das Array, das beim Aufruf der Methode `users` an einem der Objekte alle zugehörigen `User`-Objekte zurückliefert, dank des Joins in der Query bereits gefüllt und spart damit beim Zugriff weitere, separate Datenbankabfragen.

Alle Möglichkeiten und die Funktionsweise von Active Record in ihrer Gesamtheit vorzustellen würde zweifelsohne den Rahmen dieser Arbeit, insbesondere den Rahmen eines Grundlagen-Kapitels sprengen, so dass an dieser Stelle lediglich obiges Beispiel der Anschauung dienen soll. Einen umfassenden Überblick wird [RTH09] in der zum Zeitpunkt des Verfassens dieser Arbeit noch nicht erschienenen "Fourth Edition" geben. Bis dahin bieten die offiziellen Online-Guides[7] einen guten Ansatzpunkt.

Templates, HTML und CSS Natürlich ist das objektrelationale Mapping mit Active Record nicht das einzige Feature, das Rails zur Verfügung stellt und bevor im nächsten Abschnitt ein Blick auf die clientseitige Skriptsprache JavaScript geworfen wird, soll an dieser Stelle zumindest noch die Template-Engine und das Zusammenspiel der einzelnen Applikationsteile (vgl. dazu auch den obigen Abschnitt zu Model-View-Controller) kurz beschrieben werden. Grundlegende Kenntnisse von HTML und CSS werden dabei vorausgesetzt.

Aufbauend auf obiges Beispiel, sollen nun alle abonnierten Feeds eines Nutzers angezeigt werden. Für diese Aktion wird zunächst ein Controller mit einer entsprechenden Methode benötigt, die vom Browser aus aufgerufen werden kann.

```
1  class SubscriptionController < ApplicationController

3    def show
4      @feeds = User.find(params[:id]).feeds
5    end

7  end
```

Listing 2.8: Einfacher Rails Controller

Listing 2.8 zeigt den Quelltext eines solchen Controllers, der der Member-Variablen `@feeds` über das oben beschriebene Active Record-Query-Interface alle Feeds des Users zuweist, dessen `id` als GET-Paramter übertragen wurde. Auf Fehlerbehandlung wurde der Übersicht wegen verzichtet. Ein Aufruf der Applikation, etwa über die URL `http://myurl/subscription/show?id=1`, könnte diese Methode aufrufen. Über so genannte Routen kann dabei entschieden werden, welche Methode an welchem Controller aufgerufen werden soll. Eine Konvention besagt, dass der erste Teil hinter dem Host auf den Controller verweist, hier `subscription`, der zweite Teil danach auf die aufzurufende Methode, `show`. Weiterhin fällt auf, dass zwar die nötigen Daten aus dem Model geladen werden, jedoch keine Methode für das eigentliche Anzeigen der Daten aufgerufen wird. Das liegt wiederum an einer Rails-Konvention, so dass die Applikation davon ausgeht, dass im Ordner, der für die Views oder Templates angelegt wurde, ein separater Ordner für den Controller existiert und sich darin ein Template mit dem Namen `show.html.erb` befindet. Dieses

[7] http://guides.rubyonrails.org - Abruf am 05.11.2010

Template wird - mit den in den Member-Variablen gespeicherten Daten - implizit am Ende der Methode von der Template-Engine gerendert.

Hat man die Grundlagen der Sprache Ruby verstanden, muss zur Template-Engine erfreulicherweise wenig gesagt werden, da hier auf die Einführung einer separaten, weiteren Sprache verzichtet wurde. Ein einfaches Template für die obige Aufgabe könnte dabei wie in Listing 2.9 dargestellt aussehen.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>Subscriptions</title>
5    <%= stylesheet_link_tag :all %>
6    <%= javascript_include_tag :defaults %>
7  </head>
8  <body>

10 <ul>
11 <% @feeds.each do |feed| %>
12   <li><%= feed.name %></li>
13 <% end %>
14 </ul>

16 </body>
17 </html>
```

Listing 2.9: Rails-Template

Ab Zeile 11 soll eine *Unordered List* mit den im Controller aus der Datenbank geladenen Feeds erzeugt werden. Neben der normalen HTML-Syntax erkennt man dabei Tag-ähnliche Elemente, die mit `<%` oder `<%=` eingeleitet werden. In beiden Fällen darf darin normaler Ruby-Code stehen und wird auch beim Rendern des Templates ausgeführt. Der Unterschied ist, dass im ersten Fall keine Ausgabe erfolgt, im zweiten wird der Rückgabewert des Ausdrucks in den HTML-Quellcode eingefügt. In Zeile 12 wird also über alle `Feed`-Objekte iteriert, deren Name jeweils in Zeile 13 von einem `<li>`-Tag eingeschlossen ausgegeben wird.

Im `<head>`-Bereich des Templates sind zudem noch drei Aufrufe von Rails-Helper-Methoden zu sehen. Rails stellt davon für die verschiedensten Zwecke, von der einfachen Erzeugung von Formularen bis zum Umgang mit Daten und Zeiten, eine Vielzahl zur Verfügung. In obigem Beispiel werden die passenden Tags für das Einfügen von JavaScript- und CSS-Dateien aus den entsprechenden - per Konvention festgelegten - Verzeichnissen erzeugt. An dieser Stelle soll noch darauf hingewiesen werden, dass Templates im Allgemeinen ein so genanntes Layout verwenden, so dass mehrfach genutzte Teile des Templates - häufig der Teil außerhalb des `<body>`-Bereichs - nicht in jedem Template wiederholt werden müssen (vgl. “Don’t Repeat Yourself”).

In diesem größten Abschnitt des Grundlagen-Kapitels sollte also ein kurzer aber dennoch prägnanter Überblick über den Aufbau und die wichtigsten Merkmale des Frameworks Ruby on Rails gegeben werden. Da es wie bereits zuvor bemerkt an dieser Stelle nicht möglich ist, das Framework umfassend zu behandeln, soll hier noch auf die genutzte Literatur verwiesen werden, wobei

zu bemerken ist, dass Nuntio mit dem Release-Candidate der Version 3.0 von Rails entwickelt wurde. Zum Zeitpunkt des Verfassens dieser Arbeit stand speziell für diese neue Version von Rails noch keine gedruckte Fachliteratur zur Verfügung, so dass für die Neuerungen auf die offizielle Dokumentation[8] und die Rails-Guides[9] verwiesen sein soll. Viele der Grundkonzepte von Rails sind jedoch gleich geblieben, so dass die vorhandene Literatur zu Rails 2 wie [RTH09] oder [TCH08] größtenteils noch genutzt werden kann.

2.3 JavaScript, AJAX und Bibliotheken

Nachdem nun die Grundlagen des genutzten Frameworks Ruby on Rails im letzten Abschnitt beispielhaft besprochen wurden, soll in diesem Kapitel ein Blick auf eine weitere genutzte Sprache beziehungsweise Technologie geworfen werden: JavaScript.

Die JavaScript oder auch ECMAScript genannte Sprache ist für Web-Applikationen, neben pluginbasierten Anwendungen wie Flash oder Java-Applets, die einzige weitverbreitete Möglichkeit komplexere Aufgaben auf dem jeweiligen Klienten, also im Web-Browser, auszuführen. Die Sprache wurde 1995 erstmals unter dem Namen *Mocha* vorgestellt und wird heute von so gut wie jedem aktuellen Browser unterstützt. Ein wesentliches Merkmal der Sprache ist die Möglichkeit, asynchron Daten nachladen und auf das Document-Object-Model, also den Zustand der Webseite und ihre Darstellung, zuzugreifen zu können. Da die Syntax von JavaScript C-ähnlich ist und die Sprache selbst kein Kernbestandteil von Nuntio beziehungsweise dieser Arbeit ist, soll sie an dieser Stelle nicht weiter beschrieben werden. Einen guten Überblick bietet dazu [Fla07].

AJAX Viel wichtiger ist hingegen das Konzept *AJAX*, das mit JavaScript umgesetzt werden kann und so in Nuntio auch genutzt wird. Jesse James Garrett prägte den Begriff 2005 in seinem Essay [Gar05], in dem er ihn als Abkürzung für das Zusammenspiel von asynchron-genutztem JavaScript mit XML definiert.

In Abbildung 2.6 wird der Unterschied zwischen einer klassischen Webapplikation und einer AJAX-Webapplikation verdeutlicht. In ersterem Modell klickt ein Nutzer auf einen Link, sendet ein Formular oder interagiert in anderer Form mit der Webapplikation, sodass sich der Zustand ändern muss. Dabei löst der Webbrowser einen neuen HTTP-Request aus, der serverseitig verarbeitet und mit einem entsprechenden HTTP-Response beantwortet wird. Erreicht die Antwort den Browser des Nutzers, verschwindet die vorher angezeigte HTML-Seite und die neue Seite aus dem Response wird angezeigt. Da bei jeder Interaktion die komplette Webseite neu geladen wird, ist ein dynamisches Arbeiten kaum möglich. Dieses Problem wird durch das AJAX-Modell umgangen. Hier interagiert ein Nutzer wiederum mit der Applikation, aber diesmal verarbeitet eine JavaScript beziehungsweise AJAX-Engine die Nutzereingabe und löst im Hintergrund, also asynchron, einen neuen HTTP-Request aus. Der Server soll nun, nach Garretts Modell, keine komplette Webseite, sondern nur die angeforderten Daten im XML-Format übertragen.

[8] http://rubyonrails.org/documentation - Abruf am 05.11.2010
[9] http://guides.rubyonrails.org/ - Abruf am 05.11.2010

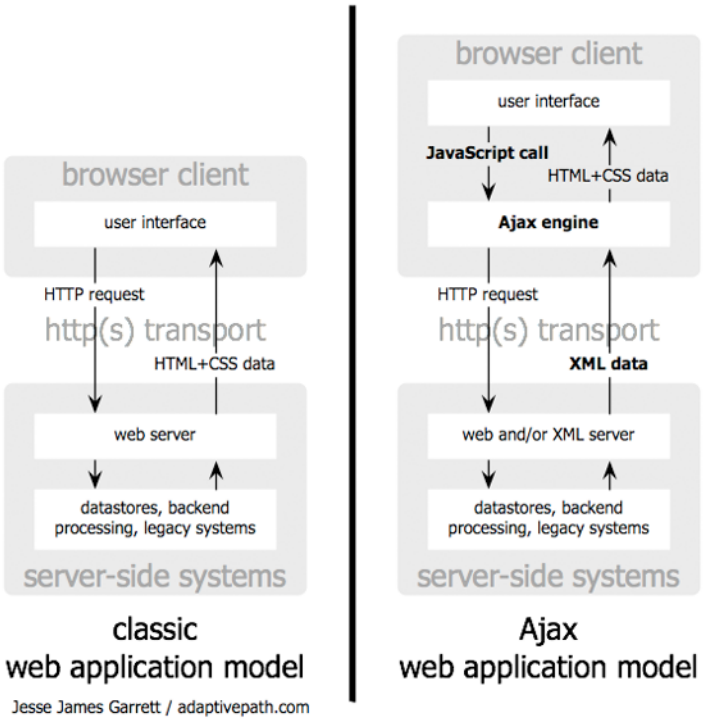


Abbildung 2.6: Klassische und AJAX-Webapplikationen im Vergleich (aus [Gar05])

Die Antwort wird wiederum innerhalb von JavaScript verarbeitet und das User-Interface, also die vorhandene Webseite, wird lediglich um die empfangenen Daten geändert. Da auch mehrere solcher AJAX-Requests gleichzeitig möglich sind, sie werden ja asynchron verarbeitet, lassen sich mit diesem Modell Webapplikationen erstellen, die extrem dynamisch sind. Ein klassisches Beispiel ist die Webseite Google Maps[10], auf der die einzelnen Kacheln der Karte im Hintergrund nachgeladen werden, während der Nutzer den Kartenausschnitt verändert.

Fünf Jahre nach Garretts Essay ist AJAX im Kontext der Webentwicklung ein mehr als bekannter Begriff, allerdings wird er heute meist etwas weiter gefasst, so dass es - obwohl das Akronym beibehalten wurde - dabei im Wesentlichen nur noch um den asynchronen Datenaustausch innerhalb einer Webapplikation geht. Ob dabei XML-Daten, JSON oder einfacher Text ausgetauscht wird, spielt für die geläufige Verwendung des Begriffs keine Rolle mehr. Wegen der starken Verbreitung von AJAX haben sich zudem einige Bibliotheken und JavaScript-Frameworks etabliert, die den Umgang damit vereinfachen. Sehr bekannt ist das so genannte Prototype-Framework[11], das die JavaScript-API um einige praktische Funktionen, nicht nur bezüglich AJAX, erweitert.

[10] http://maps.google.com/
[11] http://prototypejs.org/

```
1  var request = new XMLHttpRequest();

3  request.onreadystatechange = function() {
4    if(request.readyState == 4) {
5      document.getElementById("ausgabe").firstChild.nodeValue = request.responseText;
6    }
7  }

9  request.open("GET", "loadme.txt", true);
10 request.send(null);
```

Listing 2.10: AJAX-Aufruf in JavaScript

In Listing 2.10 ist ein AJAX-Aufruf mit JavaScript formuliert. Dabei wird das `XMLHttpRequest`-Objekt genutzt, um in Zeile 9 und 10 ein asynchrones Laden einer Textdatei durchzuführen. In einer Callback-Funktion, die in Zeile 3 definiert ist, wird - sobald die Antwort erhalten wurde - der Inhalt der Textdatei in ein vorhandenes Element der Webseite eingefügt. Da es sich bei diesem Vorgehen um ein extrem häufig auftretendes Szenario handelt, lässt sich obiger Code, der zudem der Übersicht halber um Browserweichen gekürzt wurde, bei Nutzung von Prototype auf den Aufruf `new Ajax.Updater('ausgabe', 'loadme.txt');` kürzen. Prototype ist dabei im Rails-Framework standardmäßig enthalten, genauso wie die die script.aculo.us-Bibliothek[12] die es erlaubt, mit JavaScript einfache Animationen auf der HTML-Webseite durchzuführen.

2.4 Newsfeeds

Während die Programmiersprache, Datenbank oder genutzte Frameworks für das Konzept von Nuntio problemlos austauschbar sind, ist die Möglichkeit, das System überhaupt mit geeigneten Daten füllen zu können, kritisch für die Umsetzung. Wie in der Einleitung bereits beschrieben wurde, sollen Newsfeeds als Datenquellen für Nuntio dienen, deren Aufbau in diesem Kapitel kurz besprochen werden soll.

Der Begriff Newsfeed ist zunächst eher konzeptioneller Natur, wobei der historische Hintergrund des Begriffs anklingt, nämlich die Verbreitung von Nachrichten. Häufig wird im Sprachgebrauch der Begriff RSS-Feed mit dem Begriff Newsfeed gleichgesetzt, was nicht ganz korrekt ist, da es sich bei RSS nur um *ein* mögliches Format für einen Newsfeed handelt. Unabhängig davon werden Newsfeeds häufig eingesetzt, um Nutzer auf neue Daten, seien es nun Nachrichten oder Inhalte anderer Form, aufmerksam zu machen. Unter einer bestimmten Web-Adresse wird dazu ein strukturiertes Dokument abgelegt, das - je nach Format - eine Liste der letzten veröffentlichten Daten beinhaltet. Da die Newsfeed-Formate gut maschinenlesbar sind, können Anwender die Dokumente in News-Readern abonnieren, die die Dokumente in einem bestimmten Intervall abfragen und Nutzer bei Erscheinen eines neuen Artikels darauf aufmerksam machen. Wie unten näher beschrieben ist, können Newsfeeds in den verschiedenen Formaten nicht nur einen Hinweis auf neue Daten enthalten, sondern häufig auch einen Ausschnitt der Daten selbst.

[12] http://script.aculo.us/

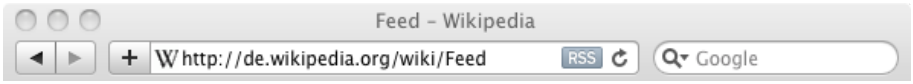


Abbildung 2.7: Hinweis auf einen RSS-Feed im Browser Safari

Webseitenbetreiber können die Existenz eines Newsfeeds im HTML-Quelltext der ausgelieferten Webseiten über ein spezielles Tag im `<head>`-Bereich der Seite ankündigen (siehe Listing 2.11). In Abbildung 2.7 sieht man, wie der Browser Safari den Nutzer auf die Existenz eines angekündigten Newsfeeds auf der rechten Seite der Adressleiste aufmerksam macht.

```
1 <head>
2   <title>Webseite mit RSS-Feed</title>
3   <link rel="alternate" type="application/rss+xml"
4    title="RSS" href="http://www.example.net/feed.rss" />
5 </head>
```

Listing 2.11: Ankündigung eines RSS-Feeds in einer HTML-Webseite (Ausschnitt)

Oben wurde bereits beschrieben, dass der Begriff Newsfeed unabhängig vom Begriff RSS-Feed zu gebrauchen ist, was daran liegt, dass es für Newsfeeds mittlerweile zwei ähnlich populäre Formate gibt, deren Aufbau im Folgenden kurz betrachtet werden soll.

2.4.1 Das RSS-Format

Das RSS-Format ist das ältere der beiden hier vorgestellten Formate und ist seit der Einführung im Jahr 1999 in verschiedenen Versionen veröffentlicht worden. Die einzelnen Versionen basieren dabei auf dem RDF-Format, einer umfangreichen Sprache zur Beschreibung von Ressourcen, unter anderem die neuste Version 2.0 jedoch auf einem einfacheren XML-Format. Auch die Abkürzung RSS hat in den verschiedenen Versionen unterschiedliche Bedeutungen, so stand es zunächst für *Rich Site Summary* später für *RDF Site Summary* und in der aktuellen Version für *Really Simple Syndication*. Da bei der Umsetzung von Nuntio eine externe Bibliothek für das Parsen der Formate genutzt wurde, soll hier nur ein Blick auf die neuste Version geworfen werden.

In Listing 2.12 ist ein Ausschnitt eines RSS-Feeds zu sehen. Man erkennt sofort, dass baumförmige XML-Format, eingeleitet durch das `rss`-Tag. Offenbar besteht ein RSS-Feed aus einem `channel`-Element, das eine Menge von `item`-Elementen beinhalten kann. Im `channel`-Element kann der Feed selbst über verschiedene Elemente beschrieben werden. Gefordert werden dabei zumindest ein Titel, ein Link zur korrespondierenden Webseite sowie eine kurze Beschreibung. Die einzelnen `item`-Elemente stehen dann jeweils für eine veröffentlichte Nachricht beziehungsweise einen neuen Datensatz, der angekündigt werden soll. Hier sind alle Elemente optional, wenn mindestens ein Titel oder eine Beschreibung vorhanden sind. Das Listing enthält dabei nur einen Ausschnitt der möglichen Angaben. Für die maschinelle Weiterverarbeitung ist jedoch zumindest die Angabe einer eindeutigen Identifikation, wie im `guid`-Tag möglich, hilfreich. Ruft ein Feed-Reader einen Feed mehrfach ab, beispielsweise in einem 10-minütigen Intervall, ist so

gewährleistet, dass erkannt werden kann, ob ein `item` bereits verarbeitet wurde oder noch nicht. Leider ist dieses Element wie bereits erwähnt optional, so dass man sich auf ein Vorhandensein bei der Verarbeitung nicht verlassen kann. Für die Nutzung in Nuntio sind vor allem die Felder `title` und `description` interessant, da diese die eigentlichen Daten beinhalten. Der Titel der Nachricht soll dem Benutzer schließlich später in der Tageszeitungs-Sicht präsentiert werden und - sofern vorhanden - auch eine Zusammenfassung der Nachricht.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <rss version="2.0">
3    <channel>
4      <title>Titel des Feeds</title>
5      <link>URL der Webpraesenz</link>
6      <description>Kurze Beschreibung des Feeds</description>

8      <item>
9        <title>Titel des Eintrags</title>
10       <description>Zusammenfassung oder Inhalt des Eintrags</description>
11       <link>Link zum vollstaendigen Eintrag</link>
12       <guid>Eindeutige Identifikation des Eintrages</guid>
13       <pubDate>Datum des Items</pubDate>
14     </item>

16     <item>
17       ...
18     </item>

20   </channel>
21 </rss>
```

Listing 2.12: Format eines RSS-Feeds (angepasst, aus Wikipedia[13])

2.4.2 Das Atom-Syndication-Format

Das Atom-Syndication-Format oder häufig - wenn auch nicht ganz korrekt - Atom-Format steht in Konkurrenz zum RSS-Format und bietet einen ähnlichen Umfang.

Vergleicht man das in Listing 2.13 dargestellte Beispiel eines Atom-Feeds mit dem RSS-Format, kann man einige Parallelen feststellen. So ist das `feed`-Element hier mit dem `channel`-Element des RSS-Feeds vergleichbar und enthält eine allgemeine Beschreibung des Newsfeeds. Danach folgt eine Menge von `entry`-Elementen, die den `item`-Elementen aus RSS entsprechen. Anders als bei RSS gibt es für das Atom-Format deutlich strengere Regeln über das Vorkommen der einzelnen Elemente. In RFC 4287[14] ist das Format beschrieben. So ist beispielsweise das `id`-Element hier nicht optional sondern obligatorisch, was die maschinelle Verarbeitung deutlich erleichtert. Zu bemerken ist noch, dass das Atom-Format es erlaubt, in Elementen sowohl HTML beziehungsweise XHTML-Markup zu verwenden. Beim RSS-Format ist dies zwar auch möglich,

[13] `http://de.wikipedia.org/wiki/RSS` - Abruf 05.11.2010
[14] http://tools.ietf.org/html/rfc4287 - Abruf am 05.11.2010

war ursprünglich jedoch nicht vorgesehen. Im Atom-Format kann ein Element wie `content` mit einem `type`-Attribut versehen werden, das das Format des Inhalts ankündigt. Hier ist neben den Angaben `text`, `html` oder `xhtml` auch die Angabe eines MIME-Types möglich. Welches der Formate das bessere ist, soll an dieser Stelle nicht entschieden werden - die Forderung nach dem obligatorischen `id`-Tag innerhalb eines Eintrags ist für die Nutzung im Rahmen von Nuntio jedoch von Vorteil.

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <author>
    <name>Autor des Feeds</name>
  </author>
  <title>Titel des Feeds</title>
  <id>Eindeutige Identifikation</id>
  <updated>Datum des letzten Updates</updated>

  <entry>
    <title>Titel des Eintrags</title>
    <link href="Adresse zur korrespondierenden Webseite"/>
    <id>Eindeutige Identifikation</id>
    <updated>Datum des letzten Updates</updated>
    <summary>Zusammenfassung des Eintrags</summary>
    <content>Volltext des Eintrags</content>
  </entry>
</feed>
```

Listing 2.13: Format eines Atom-Feeds (angepasst, aus Wikipedia[15])

2.5 Soziale Netzwerke und Web 2.0

Nach Beschreibung der bei der Umsetzung von Nuntio genutzten Technologien, soll dieser und der nächste Abschnitt zwei eher konzeptuellen Themen gewidmet sein. Bei der Beschreibung der Idee hinter Nuntio in der Einleitung wurde bereits angesprochen, dass auch ein soziales Netzwerk entwickelt werden soll. Soziale Netzwerke können dabei als Bestandteil des so genannten Web 2.0 aufgefasst werden (vgl. [O'R05]): Das statische Präsentieren von Informationen aufzubrechen und das Web als eine Plattform für User zu verstehen.

Die Autoren von [BE08] definieren ein soziales Netzwerk über drei wesentliche Merkmale: Ein Benutzer soll die Möglichkeit haben, ein Profil innerhalb des Systems anzulegen, eine Liste von anderen Nutzern des Systems, mit denen er gleichwie verbunden ist, zu pflegen und schließlich diese Listen, von sich selbst und anderen, zu betrachten und durchlaufen zu können. Das derzeit wohl prominenteste soziale Netzwerk Facebook[16] ist wohl das beste Beispiel für obige Definition, allerdings keines Falls die erste Plattform, die diese Funktionalität zur Verfügung gestellt hat. [BE08] bietet ebenfalls eine ausführliche Historie sozialer Netzwerke. Wie man an obiger

[15] `http://de.wikipedia.org/wiki/Atom_(Format)` - Abruf 05.11.2010
[16] http://www.facebook.com

Definition bereits erkennt, passt ein solches System zwar ausgezeichnet in den Kontext des Web 2.0, allerdings ist die Definition so allgemein gehalten, dass keine Aussage über das Ziel oder den Zweck eines sozialen Netzwerks getroffen wird. Tatsächlich ist es so, dass es viele soziale Netzwerke gibt, deren einzige Funktionalität eben die obigen Möglichkeiten sind, es handelt sich also um reine Kommunikationsplattformen. Bei Nuntio soll der Aspekt des sozialen Netzwerks jedoch nur eine nebengeordnete Rolle spielen. Wie etwa bei der Plattform YouTube[17] steht die eigentliche Funktionalität des Systems, hier das Betrachten und Veröffentlichen von Videos, im Vordergrund, während das soziale Netzwerk die Plattform ergänzt und erweitert.

Durch die Tatsache, dass Nutzer sich miteinander verbinden und interagieren, entstehen interessante Nutzungsmöglichkeiten dieser Daten. Eine mögliche These wäre beispielsweise, dass ein Nutzer das, was seine Freunde interessant finden, häufig auch interessant findet. Stellen sich solche Aussagen als wahr heraus, lassen sie sich nutzen, um innerhalb der Plattform einen Mehrwert zu bieten. Facebook beispielsweise zeigt auf Basis der Freunde eines Nutzers und der Freunde dessen Freunde an, wen der Nutzer möglicherweise kennen könnte, aber noch nicht zu seiner eigenen Freundesliste hinzugefügt hat. Um Aussagen wie die obige zu bestätigen oder zu widerlegen, stellen soziale Netzwerke ein interessantes Forschungsfeld dar, auch hier gibt [BE08] einen Überblick.

2.6 Tablet-Computer und das iPad

Bevor nun im nächsten Kapitel die konkrete Konzeption und Entwicklung des Nuntio-Systems auf Basis der in diesem Kapitel vorgestellten Technologien beschrieben wird, soll dieser letzte Abschnitt genutzt werden, um einen kurzen Blick auf die angestrebten Zielsysteme, auf denen Nuntios Frontend dargestellt werden soll, zu werfen.

In der Einleitung wurde bereits angekündigt, dass es sich bei Nuntio um eine Web-Applikation handeln soll. So mag man zunächst vielleicht fragen, warum überhaupt von einem "Zielsystem" gesprochen wid. Natürlich ist es so, dass das Zielsystem des Frontends letztlich ein Webbrowser sein wird. Betrachtet man aber das in der Einleitung skizzierte Nutzungsszenario für Nuntio, wird deutlich, dass ein Aspekt des Systems auch der Einsatz in einem mobilen Kontext ist. Die "personalisierte Tageszeitung" kann zwar, soll aber nicht auf einem normalen Computer, sondern auf einem mobilen Gerät wie einem Tablet-Computer, etwa Apples iPad, dargestellt werden.

Der Begriff "Tablet-PC" wurde dabei bereits im Jahr 2002 durch die gleichnamige Erweiterung für das Betriebssystem Windows XP von Microsoft geprägt. Kern dieser Erweiterung war die Möglichkeit, das Betriebssystem - sofern mit einem Touch-Screen versehen - auch mit einem Stift bedienen zu können. Passenderweise wurde dazu entsprechende mobile Hardware entwickelt, die auf eine Tastatur verzichtet und nur einen Bildschirm mit Touch-Oberfläche als Eingabemethode zur Verfügung stellt. Obwohl diese Geräte dem Nutzungsszenario von Nuntio bereits nahe kommen, fehlt dazu noch der letzte Schritt in Richtung Mobilität, da die bis dahin entwickelte Hardware meist auf Laptop-Hardware aufgebaut war, sodass die Geräte noch einigermaßen

[17] http://www.youtube.com

schwer und unhandlich waren. Anfang 2010 stellte Apple dann das iPad (siehe Abb. 2.8 (a)) vor. Ein Produkt, das man als nächste Evolutionsstufe besagter Hardware bezeichnen könnte. In seinen Hardware-Spezifikationen zwar deutlich schwächer als ein aktueller Laptop, aber dafür auch deutlich mobiler, so dass das Gerät nur noch aus einem großen Bildschirm zu bestehen scheint. Zudem wurde auf die Bedienung mit einem Stift verzichtet und dafür auf ein fingergesteuertes User-Interface gesetzt. Neben den Möglichkeiten der Applikationsentwicklung für das Betriebssystem steht auf jedem Gerät zudem die mobile Variante des Webbrowsers Safari zur Verfügung, was dieses Gerät zum idealen Zielsystem für Nuntios Web-Frontend macht. Ähnliche Produkte anderer Hersteller wie das Samsung Galaxy Tab (siehe Abb. 2.8 (b)) sind natürlich ebenso geeignet, waren aber zum Zeitpunkt des Verfassens dieser Arbeit noch nicht verfügbar.

(a) Apples *iPad* (b) Samsungs *Galaxy Tab*

Abbildung 2.8: Zwei Tablet-Computer neuerer Generation (Bilder von den Herstellerwebseiten)

Kapitel 3

Konzeption und Entwicklung

In diesem Kapitel soll nun, nach Beschreibung der grundlegenden Technologien, die konkrete Umsetzung des in der Einleitung skizzierten Systems Nuntio beschrieben werden. Zunächst wird dabei die Systemarchitektur anhand des geplanten Funktionsumfangs sowie das umgesetzte Konzept zur Skalierbarkeit besprochen. Im zweiten Abschnitt werden die zur Umsetzung nötigen Datenaggregations- und -analyseprozesse des Backends sowie die genutzten Algorithmen, auch bezüglich der Einflussnahme der Nutzer, vorgestellt. Schließlich wird die Umsetzung des Frontends, also der eigentlichen Webapplikation, präsentiert.

3.1 Systemarchitektur

In der Einleitung dieser Arbeit wurde das System von Nuntio bereits durch drei Schlüsselaspekte umschrieben. Entwickelt werden soll ein *personalisierter*, *sozial-interaktiver* Nachrichten-*Aggregator*. Diese Stichpunkte sollen in diesem Abschnitt als Basis für die Beschreibung des Funktionsumfangs und die resultierende System-Architektur dienen.

3.1.1 Überblick und Funktionsumfang

Wenn der Funktionsumfang von Nuntio beschrieben werden soll, lässt sich dies am einfachsten an den Möglichkeiten erläutern, die den Nutzern zur Verfügung stehen sollen. Zunächst soll es sich bei Nuntio um eine Web-Applikation handeln, Nutzer sollen also die Möglichkeit haben, sich über eine Web-Oberfläche in das System einloggen zu können. Als Framework kommt dabei das im Grundlagen-Kapitel beschriebene Ruby on Rails sowie eine relationale MySQL-Datenbank zum Einsatz.

Abonnieren von Newsfeeds In dieser Web-Applikation sollen Nutzer die Möglichkeit haben, Newsfeeds anzugeben, die sie abonnieren möchten. Damit ein Nutzer seine abonnierten

Newsfeeds oder besser deren Einträge auch lesen kann, müssen diese von den verschiedenen angegebenen Quellen geladen, geparsed und schließlich präsentiert werden. Wegen der geplanten Analyse-Prozesse, die im Folgenden beschrieben werden, müssen die Newsfeeds in ein zentrales Datenlager integriert werden, um sie hieraus später wieder auslesen zu können. An dieser Stelle ist bereits abzusehen, dass die Applikation in ein Front- und ein Backend aufzuteilen ist, da das Aktualisieren der Newsfeeds unabhängig vom Lesen der Nutzer auf der Frontend-Oberfläche geschehen sollte. Würde das Aktualisieren *on demand*, also nutzergesteuert, ausgeführt werden, käme es in einem Szenario, bei dem viele Nutzer gleichzeitig einen Feed aktualisieren wollen, zu einer extrem großen Server-Last und einem unnötig hohen Abfrage-Intervall der Nachrichten-Quellen. Vernünftig scheint es, die Nachrichten in einem Hintergrund-Prozess zeitgesteuert zu aktualisieren und einem Nutzer immer die gerade aktuellen Daten zu präsentieren.

Finden von ähnlichen Nachrichten Hat ein Nutzer einen Eintrag eines Newsfeeds, also eine spezielle Nachricht, ausgewählt, soll ihm eine Liste von ähnlichen Nachrichten, die in Nuntios Datenbank gefunden wurden, präsentiert werden. Diese Funktion lässt sich zwar in einem einfachen Satz fordern, die Umsetzung hingegen erfordert hier einen extremem Aufwand und stellt einen Kernaspekt, nicht nur dieser Arbeit, sondern auch des gesamten Systems dar. Spätestens hier wird deutlich, dass eine integrierte, zentralisierte Sicht auf die abonnierten Newsfeeds und deren Einträge nötig ist, um diese miteinander verknüpfen zu können.

Soziale Interaktion der Nutzer Nutzern soll innerhalb von Nuntio ein eigenes soziales Netzwerk, wie im Grundlagen-Kapitel definiert, zur Verfügung stehen. Ein Ziel dieser Funktionalität ist es, den Aufbau von Nutzergruppen zuzulassen, um das Lese-Verhalten der Nutzer innerhalb einer Gruppe analysieren zu können. Für die Nutzer entsteht dabei der Mehrwert auf neue Inhalte, die andere Mitglieder der Gruppe interessant finden, hingewiesen zu werden. Zudem soll über automatische Aktivitätsinformationen und die Möglichkeit Nachrichten zu kommentieren, eine Kommunikationsplattform entstehen.

Markieren interessanter Artikel Findet ein Nutzer einen Artikel beziehungsweise eine Nachricht interessant, soll er die Möglichkeit haben, diese als solche zu markieren. Über einen interessensbasierten Sortiermechanismus sollen ihm später ähnliche Artikel, die ja über die obige Funktionalität bereits ausgemacht wurden, betonter präsentiert werden, als Artikel, die ihm nicht gefallen.

Geeignete Präsentation der Daten Für die Präsentation der Daten muss es neben einem funktionierenden Backend ein ansprechendes Frontend geben, das auch mit den Zielgeräten (vgl. Abschnitt 2.6), also den fingergesteuerten Tablets, problemlos genutzt werden kann.

Die Umsetzung dieser Kernfunktionen und -ziele soll nun in den folgenden Abschnitten sowohl konzeptuell als auch an konkreten Beispielen aus der Implementierung besprochen werden.

3.1.2 Skalierbarkeitskonzept, Master-/Slave-Konfiguration

Möchte man zu einem System, wie im vorherigen Abschnitt beschrieben, eine Systemarchitektur ableiten, ist es sinnvoll, sich zunächst Gedanken zur Verteilung des selbigen zu machen. Entwickelt man beispielsweise eine reine Desktop-Software, liegt es auf der Hand, dass *eine* Applikation auf *einem* Rechner ausgeführt wird. Bei einer Anwendung mit gleichzeitigem Zugriff mehrerer Nutzer, wie es bei einer Web-Applikation im Allgemeinen der Fall ist, muss man sich jedoch bereits bei kleineren Projekten die Frage stellen, ob die Applikation ausreichend skalierbar ist. Häufig verhält sich die Server-Last dabei etwa proportional zur Anzahl der Nutzer eines Systems. Sind also statt 100 plötzlich 1000 Nutzer gleichzeitig eingeloggt, ist auch damit zu rechnen, dass zehn mal so viele Anfragen zu bearbeiten sind. Im Vorfeld einer Projektumsetzung ist es natürlich schwer abzuschätzen, wie hoch die Nutzerzahlen im Betrieb tatsächlich liegen werden - bereits vor Implementierung ein Konzept zur Erhöhung der Kapazitäten zu haben, kann dabei aber sicher nicht schaden.

Im vorherigen Abschnitt ist bereits angeklungen, dass Nuntio in zwei Teile aufgeteilt werden soll. Ersterer, das Frontend, soll den Nutzern zur Präsentation der Daten zur Verfügung stehen. Die Daten selbst sollen in einem Backend verarbeitet werden. Während das Frontend eine Web-Oberfläche zur Verfügung stellen soll, ist es nicht nötig, das Backend ebenfalls als Webapplikation zu konzipieren, da die Nutzer nicht direkt darauf zugreifen sollen. Wegen der Vorteile von Ruby on Rails (vgl. 2.2) wurde bei der Implementierung von Nuntio jedoch in beiden Teilen auf dieses Framework aufgebaut.

Im Fall von Nuntio ist es besonders wichtig über eine solche Skalierungs-Strategie nachzudenken, da für obiges Beispiel nicht nur gilt, dass die zehnfache Nutzerzahl eine zehnfache Anfragezahl bedeutet, sondern, dass - im schlimmsten Fall - auch die zehnfache Menge an Newsfeeds im Backend verarbeitet werden muss. Erfreulicherweise wird es im Allgemeinen so sein, dass zwischen den Nutzern Schnittmengen der abonnierten Newsfeeds auftreten. Unter Umständen ist es sogar so, dass für einen neuen Nutzer alle Newsfeeds, die dieser abonnieren möchte, bereits von anderen Nutzern abonniert wurden, und so keine zusätzliche Analyse-Last erzeugt wird.

Soll ein System verteilt werden, ist es wichtig, dass Aufgaben ausgeführt werden, die parallelisiert werden können und möglichst unabhängig voneinander sind. Arbeiten zwei Prozesse konkurrierend auf Datensätzen in der Datenbank, muss ein Transaktions-Management eingesetzt werden, damit der Datenbestand konsistent bleibt. Erfreulicherweise sind die aufwändigsten Prozesse und Aufgaben innerhalb des Backends von Nuntio quasi vollständig unabhängig voneinander. Bei Nuntios Frontend handelt es sich um eine Webapplikation, wobei eine Verteilung hier ebenso unproblematisch verlaufen kann, da HTTP ein zustandsloses Protokoll ist, so dass Anfragen - vernachlässigt man die Nutzung von Sessions - von einem beliebigen Server bearbeitet werden können.

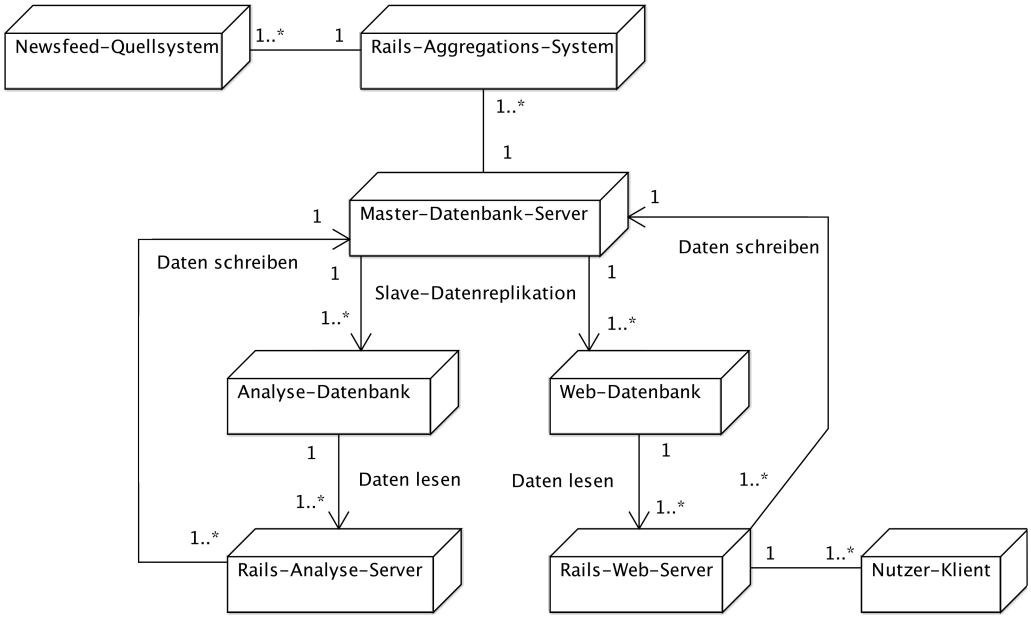


Abbildung 3.1: Die maximal-verteilte Nuntio-Architektur (vereinfacht) in UML-Notation

In Abbildung 3.1 ist die mögliche System-Architektur beziehungsweise das Skalierbarkeitskonzept von Nuntio dargestellt. Um eine möglichst gute Verteilung der Komponenten zu erreichen, wurde dabei auf eine Master-Slave-Replikationsstrategie gesetzt (vgl. auch Abschnitt 2.1.5). Im Zentrum befindet sich also der Master-Datenbank-Server, der alle Daten des Systems zentral verwaltet. In Abschnitt 2.1.1 wurde bereits erklärt, dass die Daten der einzelnen Newsfeeds zentral gespeichert werden sollen, es soll also eine materialisierte Integration der Quelldaten erfolgen. Um die Quelldaten von den verschiedenen Newsfeed-Anbietern zu sammeln und im Sinne des ETL-Prozesses in die Master-Datenbank zu laden, kann eine Menge von unabhängigen Aggregations-Systemen, die man zum Backend zählen muss, an die Datenbank angebunden werden. Dies können mehrere Rechner sein, die intervallgesteuert die Newsfeeds, die sie selbst verwalten, abfragen, die neuen Daten transformieren und schließlich in das Master-Datenbank-System speichern. Offenbar ist jedes Quellsystem unabhängig von jedem anderen und auch die Transformationsprozesse können problemlos parallelisiert werden. Eine Replikations-Strategie kommt hier noch nicht zum Einsatz, da diese Aggregations-Systeme deutlich mehr Daten in die Datenbank schreiben müssen, als sie selbst lesen. Der hier stattfindende ETL-Prozess ist in Abschnitt 3.2.1 erläutert.

Sind die verschiedenen Quelldaten, also die einzelnen Einträge der von den Nutzern abonnierten Newsfeeds, auf dem Aggregations-System vorverarbeitet und schließlich in das Master-Datenbank-System geladen worden, können diese analysiert werden. Die einzelnen Analyse-Prozesse sind in den folgenden Abschnitten erläutert. Wichtig ist nun allerdings die Tatsache, dass jeder einzelne Eintrag der Newsfeeds, also jeder Artikel, parallel zu jedem anderen analysiert werden kann. Soll beispielsweise herausgefunden werden, welche anderen Artikel inhaltlich ähnlich sind, müssen dazu zwar lesende Abfragen an die Datenbank gesendet werden, aber dies

kann parallel für alle Artikel geschehen. Da der hierzu nötige Prozess eine große Datenbank-Last erzeugt, schließlich müssen je nach Anzahl der vorhandenen Nachrichten enorm viele Daten gelesen werden, kommt hier die Master-Slave-Replikationsstrategie zum Einsatz. Der Master-Datenbank-Server verteilt also seine Daten asynchron an eine Menge von Analyse-Datenbanken. Jede Analyse-Datenbank kann nun wiederum von einer Menge von Analyse-Servern angefragt werden. Ob es dabei sinnvoll ist, eine Analyse-Datenbank für mehrere Analyse-Server zu betreiben, hängt davon ab, wie das Verhältnis der Rechenlast auf der Datenbank zur Rechenlast auf dem Analyse-System ausfällt. Wenn die Queries relativ einfach und schnell bearbeitet werden können, die Bearbeitung der Query-Ergebnisse auf den Analyse-Servern jedoch relativ aufwändig ist, kann das Betreiben einer Analyse-Datenbank mit mehreren unabhängigen Analyse-Systemen sinnvoll sein. Wegen des Mangels an großen Datenmengen während des Testbetriebs von Nuntio kann hier noch keine abschließende Aussage für das konkrete System getroffen werden. Während des Testbetriebs wurden Analyse-Server und Analyse-Datenbank nicht auf mehrere Rechner verteilt, die Software-Architektur ließe dies aber problemlos zu. Wegen der Master-Slave-Replikation ist es nun wichtig noch zu bemerken, dass die Analyse-Server lediglich lesend auf die Daten aus der Analyse-Datenbank zugreifen dürfen. Ihre Analyse-Ergebnisse müssen wiederum in den Master-Datenbank-Server geladen werden, der die Analyse-Ergebnisse für spätere Analyse-Iterationen wiederum asynchron auf die Analyse-Datenbanken verteilt. Da es hier so ist, dass der Aufwand bei den lesenden Zugriffen deutlich größer ist als der der schreibenden, bringt die Master-Slave-Architektur in diesem Kontext einen enormen Vorteil (vgl. auch Abschnitt 3.2.7 zu einigen Performance-Tests).

Auf der Seite des Frontends ist das Vorgehen ähnlich. Die nun analysierten Newsfeeds beziehungsweise deren Einträge liegen nach Verarbeitung im Backend wiederum im Master-Datenbank-Server vor und können nun auf eine Menge von Web-Datenbank-Servern repliziert werden. Jeder Web-Datenbank-Server kann also von einer Menge von Rails-Web-Servern angefragt werden. Auch hier muss die Frage gestellt werden, ob es sinnvoll ist, mehrere Web-Server an einen Datenbank-Server anzubinden. Diese lässt sich ebenfalls nur beantworten, wenn klar ist, ob die Last beim Verarbeiten der HTTP-Zugriffe der Nutzer höher ist als die dabei entstehende Last auf der Datenbank. So konnten mangels ausreichender Nutzerzahlen noch keine klaren Aussagen für Nuntios System getroffen werden. Während des Testbetriebs waren auch hier Web-Server und Web-Datenbank nicht physisch getrennt. Klar sollte allerdings sein, dass die Master-Slave-Replikation ein ausgezeichnetes Mittel ist, um viele Nutzerzugriffe auf *mehreren* verteilten Web-Servern zu verarbeiten, da die Anfragen zweier Nutzer - bis auf wenige Ausnahmen - völlig unabhängig voneinander sind. Wichtig ist aber auch hier, dass die Nutzer-Anfragen, die das Schreiben von Daten zur Folge haben, zwar auf dem Web-Server verarbeitet werden können, die neuen Daten aber wiederum in das Master-Datenbank-System geschrieben werden müssen. Möchte ein Nutzer also einen neuen Newsfeed abonnieren, muss dessen Adresse in der Master-Datenbank gespeichert werden. Ein Nachteil dieses Verfahrens ist dabei, dass der Master-Datenbank-Server die Information, dass der Nutzer einen neuen Feed abonnieren möchte, nun wiederum asynchron an die Web-Datenbank-Server verteilt. So kann es passieren, dass ein Nutzer beispielsweise anzeigen möchte, welche Newsfeeds er abonniert hat und der soeben eingetragene zwar bereits im Master-Datenbank-Server registriert ist, aber noch nicht auf der Web-Datenbank, die für diese Abfrage genutzt wird. Möglicherweise sind Änderungen von Nutzern also nicht sofort sichtbar.

Im Testbetrieb von Nuntio war die Verzögerung aber so gering, dass der Nachteil kaum ins Gewicht fiel. Zudem ließen sich solche Informationen auch temporär beispielsweise in einem Cookie auf Seite des Klienten speichern.

Zu Abbildung 3.1 muss ergänzt werden, dass die Darstellung etwas vereinfacht ist. Die gerichteten Kommunikationspfade sollen hier den Datenfluss bei der Master-Slave-Replikation verdeutlichen, da aus den Slave-Systemen nur gelesen werden darf, schreibende Zugriffe aber immer auf dem Master-Datenbank-Server erfolgen müssen. Auch fehlen Load-Balancing-Komponenten, die entscheiden, welche Anfragen von welchen Nutzern auf welchen Web-Servern bearbeitet werden. Auf das Darstellen dieser Komponenten wurde der Übersichtlichkeit halber verzichtet.

3.1.3 Rails-Integration

Neben den Vorteilen, die das Rails-Framework für die Entwicklung von Web-Applikationen bietet, besteht auch durch geringen Aufwand die Möglichkeit, in einem Master-Slave-Replikations-Szenario zu arbeiten. Wie im vorherigen Abschnitt bereits beschrieben wurde, ist es auf Seite des Frontends nötig, dass Datenänderungen auf dem Master-Datenbank-Server durchgeführt werden, während alle Leseoperationen effizient auf einer Slave-Datenbank durchgeführt werden können.

Rails stellt dazu an zentralisierter Stelle die Möglichkeit zur Verfügung, verschiedene Datenbank-Verbindungen zu konfigurieren. Im Entwicklungszustand einer Rails-Applikation wird die *Development*-Verbindung standardmäßig aufgebaut und genutzt, wenn Datenbankabfragen durchgeführt oder das objektrelationale Mapping genutzt werden sollen. Über das Active Record-Modul ist es aber auch möglich eine andere, zuvor konfigurierte Datenbankverbindung aufzubauen. Auf allen Rails-Systemen, die also auf unterschiedlichen Datenbanken lesen und schreiben, wurde dazu eine zweite Verbindung für den Master-Datenbank-Server angelegt.

```
1 ActiveRecord::Base.establish_connection(:master_development)
2 user.feeds.delete(feed) if user.feeds.include? feed
3 ActiveRecord::Base.establish_connection(:development)
```

Listing 3.1: Datenbank-Verbindungswechsel im Controller

In Listing 3.1 ist nun zu sehen, wie die derzeit aktive Datenbank-Verbindung gewechselt werden kann. Dazu ist ein Ausschnitt aus dem Controller der Webapplikation gezeigt, der einen abonnierten Feed eines Nutzers aus dessen aktiven Abonnements löscht. Da das Löschen eine schreibende Operation ist, muss dazu zunächst die Verbindung zum Master-Datenbank-Server hergestellt werden. Danach kann die gewünschte Operation ausgeführt werden und schließlich kann zur normalen Lese-Verbindung mit dem zugehörigen Slave zurückgekehrt werden. Interessant ist dabei, dass das objektrelationale Mapping von Active Record trotz Wechsel der Datenbank-Verbindung weiterarbeitet. In Zeile 2 des obigen Beispiels ist das Objekt `user` über die Lese-Verbindung, also letztlich eine komplett andere Datenbank erzeugt worden, trotzdem kann bei Aufruf der Methode `feeds` auf die abonnierten Newsfeeds des korrespondierenden Nutzers zugegriffen werden. Dazu ist es natürlich notwendig, dass auf beiden Datenbanken die gleichen Schemata für die angefragten Daten vorhanden sind, das `User`-Objekt auf der Slave-Datenbank muss also ge-

nauso aufgebaut sein und zusätzlich die gleiche `id` haben, wie auf der Master-Datenbank, damit dieser Aufruf das erwartete Verhalten zeigt. Wichtig ist auch, dass sicher gestellt ist, dass zumindest die Primärschlüssel, also hier die `id`, auf beiden Systemen während des Wechsels nicht verändert werden. Dies ist in Nuntio nur softwareseitig sicher gestellt, was einfach ist, da hier Primärschlüssel - dank ihrer synthetischen Natur - niemals geändert werden müssen.

Da durch die Master-Slave-Architektur und in Nuntio bereits sicher gestellt ist, dass ein Datensatz, der aus der Slave-Datenbank lesend bezogen wird in jedem Fall auch unter dem gleichen Primärschlüssel auf dem Master-Server vorhanden ist, kann das Wechseln der Datenbank für die einfachen *Update*- und *Delete*-Operationen auch automatisiert werden.

```
1  def before_save
2    ActiveRecord::Base.establish_connection(:master_development)
3  end

5  def before_destroy
6    ActiveRecord::Base.establish_connection(:master_development)
7  end

9  def after_save
10   ActiveRecord::Base.establish_connection(:development)
11 end

13 def after_destroy
14   ActiveRecord::Base.establish_connection(:development)
15 end
```

Listing 3.2: Datenbank-Verbindungswechsel im Model

In Listing 3.2 sind die vier Methoden angegeben, die in eine Active Record-Model-Klasse, beispielsweise die des `User`s eingefügt werden müssen, um einen automatisierten Wechsel der Datenbankverbindung zu ermöglichen. Ab Rails 3 müssen diese Methoden nicht mehr überschrieben, sondern können auch durch einen Filter beziehungsweise Hook aufgerufen werden.

```
1  user = User.find(1)
2  user.name = 'Neuer Name'
3  user.save
```

Listing 3.3: Automatischer Verbindungswechsel durch das Model

Möchte ein Nutzer beispielsweise seinen Namen ändern, kann er dies im Web-Frontend über ein entsprechendes Formular tun, das er schließlich an den Webserver sendet. Dort muss der passende Datensatz - wie in Listing 3.3 - für den Nutzer zunächst über dessen Primärschlüssel aus der Datenbank angefragt werden. Da es sich dabei um eine Lese-Operation handelt, kann dazu die bereits bestehende Verbindung zur Slave-Datenbank genutzt werden. Nun kann über die Objekt-Abstraktion des Datensatzes der Name des Nutzers geändert werden und schließlich muss der geänderte Datensatz gespeichert werden. Dies muss aber auf der Master-Datenbank erfolgen, was dank der Methoden aus Listing 3.2 auch passiert. Vor dem Aufruf der `save`-Methode wird die `before_save`-Methode des `User`-Models aufgerufen, was den Wechsel der Datenbankverbindung

zum Master-Server zur Folge hat. Active Record generiert jetzt automatisch das passende `UPDATE`-Statement, das an die Master-Datenbank gesendet wird, woraufhin die `after_save`-Methode aufgerufen wird, die die aktive Datenbankverbindung wieder auf den Slave-Server setzt, so dass die folgenden Lese-Operationen wiederum dort ausgeführt werden können. Auf diese Weise ermöglicht Rails mit wenigen Mitteln ein sehr komfortables Arbeiten in einer Master-Slave-Umgebung.

3.2 Datenaggregations- und -analyseprozesse

In Abschnitt 3.1.1 zum Überblick über den Funktionsumfang von Nuntio wurden bereits die wesentlichen Prozesse zur Datenaggregation und -analyse skizziert. Im Folgenden werden die einzelnen Schritte dieser Prozesse noch einmal genauer spezifiziert und in den weiteren Unterkapiteln ausführlich beschrieben. Wichtig ist dabei zu beachten, dass die folgenden Abschnitte sich immer auf Prozesse im Backend des Systems beziehen. Das Frontend wird später in Abschnitt 3.5 separat behandelt.

Das Kernproblem, das im Folgenden in einzelne Analyse-Schritte aufgeteilt wird, ist dabei das Finden von ähnlichen Nachrichten zu einem Eintrag eines Newsfeeds, nachdem dieser erfolgreich aus einer externen Quelle in die Datenbank geschrieben wurde. Zu diesem Problem, das sich als Text-Klassifikations-Problem auffassen lässt, gibt es in der Literatur verschiedene Lösungsansätze, die an dieser Stelle nicht umfassend diskutiert werden sollen. Einen Überblick liefert unter Anderem [Joa02]. Ziel dieses Kapitels soll es vielmehr sein, einen möglichen Ansatz im Kontext der Optimierung durch den Einsatz im Rahmen eines sozialen Netzwerks und dessen konkrete Implementierung auf Basis der zuvor beschriebenen Architektur vorzustellen.

Um die Idee des hier vorgestellten Algorithmus zu verdeutlichen, sollen die folgenden drei Nachrichten betrachtet werden:

Google Street View: Erste Bilder aus Deutschland im Netz veröffentlicht von ad-hoc-news.de am 02.11.2010 08:42:18 UTC mit dem Inhalt: “Berlin (dpa) - Google Street View hat erste Straßenbilder aus Deutschland ins Internet gestellt. Damit will Google einen ersten Vorgeschmack auf seinen Online-Straßenatlas geben. Es geht zunächst aber nur um sechs Sehenswürdigkeiten, zehn Bundesliga-Stadien und ...”

Vorgeschmack auf Street View: Erste Bilder aus Deutschland veröffentlicht von n-tv.de am 02.11.2010 06:59:43 UTC mit dem Inhalt: “Der Fotodienst des Suchmaschinenbetreibers Google öffnet seine deutschen Pforten mit einem Schnupperangebot. Google Street View zeigt erste Aufnahmen aus Deutschland im Internet.”

Google Street View startet in Oberstaufen veröffentlicht von heute.de am 02.11.2010 10:15:04 UTC mit dem Inhalt: “Seit heute ist Oberstaufen virtuell im Internet zu begehen. Google zeigt die erste deutsche Gemeinde: Oberstaufen ist jetzt über Google Street View im

Internet einsehbar. Nun kann jeder in Fußgängerperspektive durch den Allgäuer Kurort laufen - doch nicht alle Oberstaufener sind einverstanden."

In [Joa02] werden fünf Ebenen von Text-Analyse unterschieden, von denen die zweite das "Word Level", also die Klassifikation auf Basis der einzelnen Wörter ist. Man könnte mit diesem einfachen Ansatz bereits vermuten, dass die drei Artikel ähnlichen Inhalt haben, weil in allen drei Überschriften die Wörter "Street" und "View" auftauchen. In der nächsthöheren Ebene des "Multi-Word Level" können nun auch Phrasen und syntaktische Zusammenhänge berücksichtigt werden, man darf also weiterhin annehmen, dass die Artikel ähnlichen Inhalt haben, weil in jeder Überschrift der Begriff "Street View" auftaucht. In der nächsten Ebene des "Semantic Level" wird nun die Bedeutung eines Textes erfasst. So kann ein Mensch beispielsweise feststellen, dass die Phrase "Erste Bilder aus Deutschland" eine ähnliche Bedeutung wie "startet in Oberstaufen" hat, da sowohl "startet" als auch "Erste" in diesem Fall auf den Beginn von etwas hindeuten. Die letzte Ebene, das "Pragmatic Level", würde nun zusätzlich den Kontext und die aktuelle Situation, in der eine Nachricht erscheint berücksichtigen. Wüsste ein Mensch, der die drei Artikel betrachtet also, dass der Konzern Google kurz davor steht, den Dienst *Street View* in Deutschland zu starten, erhärtet das die Vermutung, dass die ersten beiden Artikel zusammen gehören. Um den letzten definitiv zuordnen zu können muss zusätzlich die Information vorhanden sein, dass der Ort "Oberstaufen" in Deutschland liegt.

Als Mensch muss man sich geradezu anstrengen, um zu versuchen, die Semantik, also die dritte Ebene der Text-Analyse zu ignorieren. Für eine Maschine oder einen Algorithmus ist es genau umgekehrt - bereits die Semantik eines Textes verstehen zu können, ist extrem kompliziert. Laut [Joa02] gibt es für die letzte Ebene bisher noch nicht einmal hinreichende Ansätze. Für ein System wie Nuntio ist es zusätzlich problematisch, dass bereits der syntaktische Zusammenhang eines Textes nicht für alle Sprachen gleich zu erschließen ist. Sollen also Nachrichten in verschiedenen Sprachen verarbeitet werden, müsste eine Syntax-Analyse auf Basis der jeweiligen Sprache stattfinden. Da Nuntio zunächst sprachunabhängig - zumindest im Rahmen von Sprachen mit lateinischen Schriftzeichen - gestaltet werden sollte, wurde für die Text-Analyse die Ebene des "Multi-Word Level" verwendet, jedoch ohne Berücksichtigung der syntaktischen Elemente. Einfach ausgedrückt soll das System die Texte auf Basis der darin vorkommenden Wörter und Phrasen klassifizieren beziehungsweise miteinander verknüpfen. Alle höheren Ebenen bleiben dabei unberücksichtigt.

Für den hier vorgestellten Algorithmus bedeutet das also, dass die Ähnlichkeit des Inhalts beispielsweise anhand des Begriffs "Street View" festgemacht wird. Allerdings ist es auch so, dass sowohl in der ersten als auch der zweiten Überschrift der Begriff "aus" vorkommt. Offensichtlich ist dies jedoch kein relevanter Begriff um festzustellen, ob zwei Nachrichten ähnlich sind oder eben nicht. Man kann also behaupten, dass es eine Reihe von relevanten Wörtern und Phrasen in den Texten gibt, die auf eine Ähnlichkeit hindeuten und eine Reihe von Wörtern, die überhaupt keine Rolle spielen. Festzustellen, welche Begriffe relevant sind, ist ein schwieriges Unterfangen, wobei es relativ einfach ist, eine Menge von Begriffen zu definieren, die sehr wahrscheinlich irrelevant sind. Sucht man einige aus den obigen Texten, fallen sofort alle Artikel wie *der*, *die* und *das* auf, genauso wie die Wörter *mit* oder *ist*. Abgesehen davon, dass diese Wörter sehr kurz sind, fällt auch auf, dass sie fast in jedem Text vorkommen. Über alle Texte - zumindest in der

gleichen Sprache - sind sie also extrem häufig. Da man also nicht ohne Weiteres sagen kann, welche Begriffe relevant sind, kann man zumindest sagen, dass extrem häufige Begriffe nicht als relevant zu betrachten sind. Möchte man diese Begriffe identifizieren, steht man aber vor dem Problem, dass diese je nach Sprache unterschiedlich sind. Während im deutschen die Artikel zu den häufigsten Wörtern gehören, müsste man für die englische Sprache das *the* als irrelevant betrachten. Für Nuntio ist dies ein neues Problem, da - neben anderen Sprachen - sowohl englische als auch deutsche Newsfeeds analysiert werden sollen. Leider sind die Newsfeeds im Allgemeinen jedoch nicht bezüglich ihrer Sprache gekennzeichnet. Soll also die oben vorgeschlagene Liste von irrelevanten Begriffen erstellt werden, müssen die Newsfeeds zunächst durch ihre Sprache klassifiziert werden. Das genaue Vorgehen innerhalb von Nuntios Backend ist hierzu in den Abschnitten 3.2.2 bis 3.2.6 erläutert.

Ist schließlich eine Liste mit vermutlich relevanten Stichwörtern und Phrasen unter anderem durch das Weglassen von irrelevanten Begriffen - der genaue Ablauf ist in Abschnitt 3.2.4 beschrieben - erstellt, kann eine Nachrichtenmeldung mit den Schlüsselwörtern indiziert werden, die in der Nachricht selbst und in der resultierenden Liste enthalten sind. Der genaue Prozess ist dabei in Abschnitt 3.2.5 dargestellt. Schließlich kann, wie in Abschnitt 3.2.6 erläutert wird, ein Ähnlichkeitsmaß definiert werden, womit festgestellt werden kann, ob zwei Artikel inhaltlich ähnlich sind oder eben nicht.

3.2.1 Datenquellen: Laden und Parsen von Newsfeeds

Bevor aber eine Analyse der einzelnen Newsfeeds beziehungsweise ihrer Einträge durchgeführt werden kann, müssen die Daten zunächst geladen und - für Nuntios Architektur - in die Master-Datenbank integriert werden. Um überhaupt einen Newsfeed laden zu können, wurde dazu eine separate Tabelle `proposed_feeds` angelegt. Möchte ein Nutzer später im Frontend einen neuen Newsfeed abonnieren, der noch nicht in Nuntios Datenbank berücksichtigt ist, also von noch keinem anderen Nutzer abonniert wurde, wird in dieser Tabelle ein neuer Eintrag angelegt, der später mit dem vorschlagenden Nutzer verknüpft werden kann. Eines der Rails-Aggregations-Systeme (vgl. Abb. 3.1 in Abschnitt 3.1.2) kann dann versuchen den Newsfeed zu laden.

In Abschnitt 2.4 wurden hierbei zwei mögliche Formate solcher Newsfeeds dargestellt. Um diese XML-Daten zu laden und parsen, so dass in geeigneter Form auf die einzelnen Einträge und Datenfelder eines Newsfeeds zugegriffen werden kann, wird die Feedzirra-Bibliothek[1] von Paul Dix eingesetzt. Die Bibliothek ist unter der MIT-Lizenz[2] veröffentlicht und damit frei zugänglich. In der Beschreibung der Bibliothek schreibt der Autor: "Feedzirra is a feed library that is designed to get and update many feeds as quickly as possible." Feedzirra selbst ist dabei ebenfalls in Ruby geschrieben und lässt sich problemlos in den bereits vorhandenen vorhandenen Ruby-Code einbinden. Die Bibliothek stellt dabei einen Parser zur Verfügung, der die einzelnen Newsfeeds als Objekte repräsentiert und ihre Datenfelder so leicht zugänglich macht.

[1] https://github.com/pauldix/feedzirra
[2] http://www.opensource.org/licenses/mit-license.php

```
1  feeds = ProposedFeed.all
2  feeds.each do |feed|

4    loaded_feed = Feedzirra::Feed.fetch_and_parse(feed.url)

6    db_feed = Feed.new
7    db_feed.title = loaded_feed.title
8    db_feed.url = loaded_feed.url
9    db_feed.feed_url = loaded_feed.feed_url
10   db_feed.last_modified = loaded_feed.last_modified
11   db_feed.save

13 end
```

Listing 3.4: Einfaches Beispiel zum Umgang mit Feedzirra

In Listing 3.4 ist ein einfaches Beispiel zum Umgang mit Feedzirra angegeben. Deutlich wird zunächst der Vorteil, der sich aus der Tatsache ergibt, dass Rails auch im Backend eingesetzt wird, da in Zeile 1 alle neu hinzuzufügenden Newsfeeds einfach über den Active Record-Befehl `ProposedFeed.all` geladen werden können. In der darauf folgenden Schleife wird Feedzirra nun jeweils aufgefordert, den über das Datenfeld `url` identifizierten Newsfeed zu laden und ihn danach direkt zu parsen. Unter der Annahme, dass dies erfolgreich war - die Fehlerbehandlung wurde oben der Übersichtlichkeit halber weggelassen - kann der `ProposedFeed` in einen `Feed`, also einen Eintrag in Nuntios Basis-Tabelle `feeds`, umgewandelt werden. Wie man an den Zeilen 7 bis 10 sieht, ist es nun einfach möglich, auf die Datenfelder des ursprünglichen XML-Dokuments mit Methoden des von Feedzirra erzeugten `Feedzirra::Feed`-Objekts zuzugreifen. Es ist sogar so, dass die Bibliothek ein gutes Stück der Datenintegration übernimmt. In dem Datenfeld `url` findet sich dabei beispielsweise die Internetadresse des Newsfeed-Anbieters wieder, also beispielsweise `http://www.tagesschau.de`. Betrachtet man aber die beiden in Abschnitt 2.4 vorgestellten Newsfeed-Formate, stellt man fest, dass im RSS-Format dazu ein eigenes Tag mit dem Namen `link` vorgesehen ist, während im Atom-Format zwar ebenfalls das `link`-Tag vorhanden ist, aber die eigentliche Adresse nicht Inhalt des Knotens sondern Inhalt des `href`-Attributs ist. Feedzirra integriert diese beiden Darstellungen je nach erkanntem Feed-Format nun einfach zugänglich in ein Datenfeld des entsprechenden Objekts.

Ist ein `ProposedFeed` also zum ersten Mal erfolgreich geladen und mit den gelesenen Meta-Daten erfolgreich in die Master-Datenbank geschrieben worden, kann er aus der `proposed_feeds`-Tabelle gelöscht werden und der oder die Nutzer, die diesen Newsfeed vorgeschlagen haben, können mit dem neu angelegten `Feed`-Objekt verknüpft werden, um ihr Abonnement zu starten. Der neu angelegte Newsfeed kann nun also periodisch abgefragt werden, damit seine einzelnen Einträge ebenfalls - zunächst zur Analyse - in die Master-Datenbank geladen werden. Hierzu kann jeder Newsfeed über eine Identifikationsnummer noch einem Aggregations-System zugewiesen werden, das später für das periodische Update zuständig ist. Da das Laden der einzelnen Newsfeeds unabhängig voneinander passieren kann, lässt sich das System auf diese Weise einfach erweitern. Ist also beispielsweise ein zehnminütiges Abfrageintervall der Newsfeeds vorgesehen und ist der eingesetzte Aggregations-Server wegen der hohen Last nicht mehr in der Lage dieses Intervall einzuhalten, kann parallel dazu ein weiterer Server eingesetzt werden.

```
1  loaded_feed.entries.each do |entry|

3    db_entry = FeedEntry.new
4    db_entry.feed = db_feed
5    db_entry.url = entry.url
6    db_entry.fid = entry.id
7    db_entry.title = strip_tags(coder.decode(entry.title))
8    db_entry.summary = sanitize(coder.decode(entry.summary))
9    db_entry.content = sanitize(coder.decode(entry.content))
10   db_entry.save

12 end
```

Listing 3.5: Bearbeiten der einzelnen Einträge mit Feedzirra

Das Laden der einzelnen Einträge im Rahmen des Newsfeed-Updates erfolgt dabei ganz ähnlich zum initialen Laden (vgl. Listing 3.4), da Feedzirra die einzelnen Einträge im Datenfeld `entries` des korrespondierenden Feed-Objektes als Array zur Verfügung stellt. Auch hier hat automatisch eine Integration der `entry`-Knoten des Atom-Formats und der `item`-Knoten des RSS-Formats stattgefunden. In Listing 3.5 ist zu sehen, wie nun einfach über alle Einträge eines Newsfeeds iteriert werden kann. Unter der Annahme, dass der Eintrag noch nicht in der Datenbank vorliegt - das Beispiel ist wiederum entsprechend verkürzt, siehe auch unten - kann ein neues `FeedEntry`-Objekt angelegt werden, das später zu einer Zeile in der zugehörigen Tabelle `feed_entries` wird. Erneut können die einzelnen Datenfelder des von Feedzirra bereitgestellten Objekts genutzt werden, um die Datenfelder, die in der Datenbank vorgesehen sind, zu füllen.

Das angegebene Beispiel ist extrem verkürzt, da Feedzirra zwar eine grundlegende Datenintegration vornimmt, aber weder die Datenqualität prüft, noch den gelieferten Inhalt. Immer wenn man externe Datenquellen zur Nutzung heranzieht, muss man sich dabei mit diesen Themen beschäftigen. Im eigentlichen Lade-Modul von Nuntio finden dazu eine ganze Reihe von Abfragen statt, wobei zunächst ein einfacher Qualitätsstandard festgelegt wird. So muss jeder Newsfeed zumindest einen Titel für sich selbst definieren, Newsfeeds ohne Titel werden beim initialen Prüfen abgelehnt. Für jeden Eintrag eines Newsfeeds muss ebenfalls mindestens ein Titel aber auch eine zugehörige URL vorhanden sein, damit er in Nuntios System eingetragen wird. Das Vorhandensein von eigentlichem Inhalt über die Felder `summary` oder `content` wird nicht gefordert, da viele Newsfeeds nur ihre Schlagzeilen zur Verfügung stellen. Diese werden in Nuntios Frontend später separat behandelt. Die URL hingegen ist nötig, um einzelne Einträge eines Newsfeeds später wieder eindeutig identifizieren zu können. Wird also ein Newsfeed beispielsweise nach einer zehnminütigen Wartezeit erneut abgefragt, liefert er zwar zunächst die neuen Einträge, da das Protokoll aber keinen Zustand hat, werden ebenfalls die Einträge erneut gesendet und verarbeitet, die bereits im vorherigen Ladevorgang in die Datenbank geschrieben wurden. Um doppelte Einträge zu vermeiden, wird nun die URL eines jeden Eintrags als Schlüssel genutzt, um den bereits vorhandenen Eintrag in der Datenbank zu finden. In diesem Fall würden die Datenfelder lediglich mit den neuen Daten gefüllt werden - schließlich kann sich der Inhalt auch geändert haben. Beim Atom-Format ist der Vorgang wie bereits in Abschnitt 2.4.2 angeklungen etwas einfacher, da jeder Eintrag mit einem `id`-Tag versehen werden muss, der noch besser als Schlüssel geeignet ist. Fehlt dieser Eintrag, wird auch hier die URL als Notlösung verwendet.

Ein weiteres Problem besteht allgemein im Umgang mit externen Daten, die aus nicht vertrauenswürdigen Quellen geladen werden. "Nicht vertrauenswürdig" klingt dabei vielleicht zunächst sehr harsch, aber da in Nuntio die Nutzer selbst die Möglichkeit haben, die Datenquellen zu spezifizieren, kann im Grunde jede Datenquelle, auch manipulierte Datenquellen möglicher Angreifer, in das System gelangen. Um verschiedene Angriffsszenarios abzuwehren, müssen die Daten vor dem Eintragen in die Datenbank von unerwünschten Inhalten gereinigt werden. Ab Zeile 7 von Listing 3.5 sieht man dabei das Vorgehen in Nuntio. Soll also der Titel eines Eintrags in die Datenbank geschrieben werden, wird dieser vorher dekodiert, so dass maskiertes HTML-Markup wieder im Klartext erscheint. Ein Titel mit dem Inhalt `&lt;i&gt;kursiv&lt;/i&gt;` würde also nach Dekodierung in der Form `<i>kursiv</i>` vorliegen. Mit dem Aufruf der Rails-Hilfsmethode `strip_tags` werden nun alle HTML-Tags entfernt, so dass nur noch der schadlose Inhalt `kursiv` übrig bleibt. Natürlich ist das Markieren von Text als kursiv nicht als Angriff zu betrachten, auf diese Weise kann jedoch auch unerwünschtes HTML entfernt werden, so dass Überschriften grundsätzlich in ihrer reinen Textform vorliegen. Das Layout selbiger kann in Nuntios Frontend einheitlich erfolgen.

Ganz ähnlich verfahren wird mit den Datenfeldern `summary` und `content`. Für die Darstellung in Nuntios-Frontend ist hier jedoch erwünscht, dass das HTML-Markup größtenteils beibehalten wird. Stellt ein Newsfeed-Betreiber im Datenfeld `content` also den kompletten Artikel seiner Nachricht beispielsweise mit Bildern und Tabellen zur Verfügung, soll dieser später so auch in Nuntios Frontend angezeigt werden können. Trotzdem darf man den Inhalt dieser Datenfelder nicht unbearbeitet in die Datenbank schreiben, da das System ansonsten beispielsweise gegen Cross-Site-Scripting-Angriffe anfällig wird. Ein Newsfeed-Betreiber könnte also neben dem gewünschten HTML-Markup auch JavaScript-Code in seinen Content einbauen, der dann innerhalb von Nuntio ausgeführt würde. Weniger kritisch aber ebenso unerwünscht ist es, wenn ein Betreiber seinen Artikel beispielsweise mit einer eigenen Schriftart oder -größe versieht, die nicht zum übrigen *Look and Feel* von Nuntios Frontend passt. Es muss also zwischen erlaubtem und unerwünschten Markup unterschieden werden. Dazu bietet Rails die Methode `sanitize` an, die über ein Whitelisting-Verfahren bestimmte Tags zulässt und alle sonstigen entfernt. Ebenso wird eingebettetes CSS geprüft und unerwünschte Eigenschaften, wie das Setzen einer anderen Schriftart, können gelöscht werden.

Bevor der Eintrag nun in die Datenbank gespeichert wird, folgt ein letzter Schritt der Datentransformation beziehungsweise Datenvorverarbeitung. Um die Inhalte der einzelnen Einträge später miteinander vergleichen zu können, müssen diese in eine geeignete Form gebracht werden. Als erster Schritt wird dabei einfach der gesamte textuelle Inhalt eines Eintrags von sämtlichem Markup sowie Interpunktion und sonstigen Nicht-Text-Zeichen befreit, so dass alle Wörter - noch in der richtigen Reihenfolge - durch jeweils ein Leerzeichen getrennt, aufeinanderfolgend in einem Datenfeld `analyzable_text` in der Datenbank zur Verfügung stehen.

Hat ein Eintrag die Qualitätssicherung überstanden und sind seine Datenfelder in die gewünschte Form transformiert worden, kann er in die Master-Datenbank gespeichert werden. Vorher wird für den Eintrag jedoch, ähnlich wie bei den einzelnen Newsfeeds, eine Identifikation für den Analyse-Rechner hinzugefügt, damit die Analyse der einzelnen Einträge später ebenfalls unabhängig voneinander erfolgen kann. Wie bereits in Abschnitt 3.1.2 beschrieben, ist es auch

hier möglich die Skalierungsstrategie aus Nuntios Architektur geeignet umzusetzen. Werden also schneller neue Einträge zur Analyse in die Datenbank geschrieben als ein bestehendes System diese verarbeiten kann, wird einfach ein weiterer Rechner zur Architektur hinzugefügt, der die überzähligen Einträge parallel verarbeitet.

3.2.2 Clustering der Datenquellen nach Sprache

Nachdem die einzelnen Einträge der Newsfeeds nun in der Master-Datenbank vorliegen, werden diese in die einzelnen Analyse-Datenbanken repliziert, wo nun die eigentliche Analyse beginnen kann. In der Einführung dieses Kapitels wurde bereits beschrieben, dass es für den gewählten Ansatz nötig ist, ein Wörterbuch mit Worthäufigkeiten für die jeweilige Sprache eines Newsfeeds aufzubauen. Das Problem, das dort ebenfalls bereits genannt wurde ist, dass die Newsfeeds im Allgemeinen nicht in ihrer Sprache gekennzeichnet sind und selbst wenn, müsste diese Information im Sinne der nicht-vertrauenswürdigen Datenquellen kritisch betrachtet werden.

Um ein solches Wörterbuch sprachabhängig aufbauen zu können, ist es also erforderlich die Sprache, in der ein Newsfeed verfasst ist, herauszufinden. Dabei ist es unerheblich, in welcher konkreten Sprache die Nachrichten veröffentlich werden, es geht einzig darum, dass zwei Newsfeeds der *gleichen* Sprache auch mit dem *gleichen* Wörterbuch berücksichtigt werden.

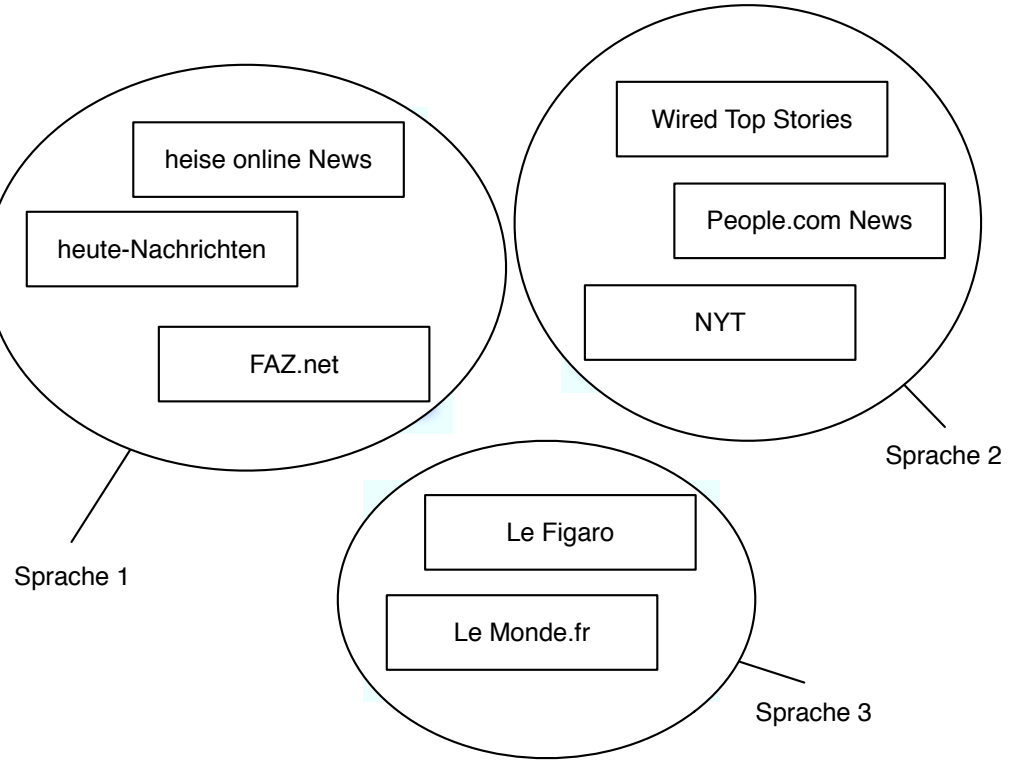


Abbildung 3.2: Visualisierung des Clustering-Problems

Um also die Frage nach der Gleichheit der Sprache einer Menge von Newsfeeds zu beantworten, ist es wiederum nötig ein Klassifikations-Problem zu lösen. Unter anderem in [DHS01] und [HMS01] sind hierzu eine Reihe von Algorithmen und Konzepten zur Lösung solcher Probleme vorgestellt. An dieser Stelle soll es als Clustering-Problem aufgefasst werden. In Abbildung 3.2 wird für einen beispielhaften Datenbestand vereinfacht davon ausgegangen, dass bereits eine geeignete Abbildung auf zwei Raum-Dimensionen gefunden wurde. Ziel des Clustering-Algorithmus ist es nun, mit Hilfe einer geeigneten Abstands- oder *Scoring*-Funktion festzustellen, dass hier drei Cluster gebildet werden können, deren Elemente untereinender einen möglichst geringen Abstand haben, deren Abstand zu den Elementen anderer Cluster aber möglichst groß ist.

Im Fall von Nuntio bestehen hierbei zwei grundsätzliche Probleme. Die meisten der klassischen Clustering-Algorithmen, wie beispielsweise *k-means* (vgl. [HMS01]) gehen von zwei Grundannahmen aus: Zunächst handelt es sich bei vielen der Algorithmen um Offline-Algorithmen, also solche, bei denen bereits zu Beginn des Algorithmus *alle* Daten zur Verfügung stehen. Weiterhin benötigen viele der Algorithmen eine zumindest geschätzte Angabe der Anzahl der Cluster, die gebildet werden sollen (eben das k in *k-means*). In Nuntio sind beide Voraussetzungen aber nicht erfüllt. Erstens können von den Nutzern ständig neue Newsfeeds eingetragen werden, die bestehenden Clustern, also Sprachen, zugeordnet werden müssen und zweitens kann es genauso sein, dass ein neuer Newsfeed in einer noch nicht berücksichtigten Sprache veröffentlicht ist, also ein neuer Cluster erzeugt werden muss. In [HMS01] wird dabei für solche Fälle unter anderem vorgeschlagen, ein "threshold" einzuführen, also ein schwellwertbasiertes Verfahren einzusetzen.

Bevor ein solcher Schwellwert aber festgelegt werden kann, muss zunächst feststehen, für welche Funktion dieser überhaupt gelten soll. Es muss also eine geeignete Abstandsfunktion gefunden werden. In der beispielhaften Abbildung oben ist dies einfach, da die Newsfeeds bereits auf einen Ort im zweidimensionalen Raum abgebildet wurden - für die tatsächliche Datenbasis existiert so ein Mapping aber zunächst nicht. Um also eine Abstandsfunktion zu definieren, wird zunächst folgende Grundannahme gemacht: Zwei Newsfeeds sind wahrscheinlich in der gleichen Sprache veröffentlicht, wenn für die einzelnen Einträge der beiden Newsfeeds gilt, dass die Schnittmenge der Wörter, die in zwei betrachteten Einträgen vorkommen, im Durchschnitt und unter Berücksichtigung der Länge der Einträge relativ groß ist.

Hierzu ein Beispiel. Auf der einen Seite sei ein deutscher Newsfeed mit allgemeinen Nachrichten gewählt, auf der anderen Seite ein englischer Newsfeed über Computer-Themen. In einem ungünstigen Fall würde nun in beiden Newsfeeds eine Nachricht gewählt werden, die zwar inhaltlich ähnlich ist - die Sprache beider Nachrichten aber nicht übereinstimmt. Ein Beispiel wären Nachrichten mit den Inhalten "Apple hat heute in Cupertino eine neue Generation von Tablet-Computern vorgestellt: das iPad." sowie "Today Apple introduced a new generation of tablet computers - the iPad.". Die Schnittmenge wäre hier (ohne Beachtung von Groß- und Kleinschreibung) mit *apple*, *generation*, *tablet* und *ipad*, relativ groß für zwei Texte mit knapp über 10 Wörtern. Um dieses Problem zu umgehen kommt nun der besagte Durchschnitt zum Tragen, dabei wird die gleiche Nachricht des deutschen Newsfeeds mit anderen Einträgen des englischen verglichen. Möglicherweise kann hier bei weiteren Apple-spezifischen Nachrichten eine Schnittmenge auftreten, diese wird aber eben im Durchschnitt deutlich kleiner sein. Noch weiter verbessert sich das Ergebnis, wenn nicht nur *ein* Eintrag aus dem deutschen Newsfeed betrach-

tet wird sondern verschiedene gewählt werden. Eine weitere Nachrichtenmeldung könnte lauten: "Die seit nunmehr Wochen verschütteten Minenarbeiter in Chile haben neuen Grund zur Hoffnung." Mit dieser Nachricht hat obige englische überhaupt keine Schnittmenge und jede andere des englischen Newsfeeds vermutlich auch höchstens über das Wort "in" oder "Chile". Auf der anderen Seite ist es so, dass bei einem Vergleich des deutschen Newsfeeds mit einem anderen deutschen Newsfeed - unabhängig von dessen thematischer Ausrichtung - im Durchschnitt eine größere Schnittmenge über die häufig vorkommenden Wörter wie "hat", "in", "eine" oder "von" zu erwarten ist. Finden also hinreichend viele Vergleiche statt, kann man behaupten, dass die durchschnittliche Größe der Schnittmenge zwischen Newsfeeds gleicher Sprachen größer sein wird als die unterschiedlicher.

Auf dieser Basis wird in Nuntio eine Abstandsfunktion implementiert. Man muss allerdings eher von einer *Scoring*-Funktion sprechen, da die Funktion - wie im Folgenden beschrieben wird - so definiert ist, dass größere Werte für ein engeres Verhältnis, also einen geringeren Abstand, sprechen. Zudem ist der Abstand von einem Eintrag zu sich selbst nicht 0, hier nicht mal konstant. Für den Schwellwert-Ansatz spielt das aber keine Rolle.

```
1  def overall_match_rating(other_entry)
2    intersection = self.word_list.keys & other_entry.word_list.keys

4    sum1 = 0
5    sum2 = 0

7    intersection.each do |word|
8      sum1 += self.word_list[word]
9      sum2 += other_entry.word_list[word]
10   end

12   dist1 = sum1 / self.word_list.keys.length.to_f
13   dist2 = sum2 / other_entry.word_list.keys.length.to_f

15   dist1 > dist2 ? dist1 : dist2
16 end
```

Listing 3.6: Scoring-Funktion als Methode eines `FeedEntry`-Objekts (Ausschnitt)

In der Klasse `FeedEntry`, die einen Eintrag eines Newsfeeds repräsentiert, ist die in Listing 3.6 dargestellte Methode definiert, die besagte Scoring-Funktion berechnet. Dazu wird in Zeile 2 zunächst die Schnittmenge der in beiden `FeedEntry`s vorkommenden Wörter aus dem `Hash word_list` mit Wörtern als Schlüssel und Häufigkeit als Wert gebildet. Ab Zeile 7 wird dann über diese Wörter iteriert, um festzustellen, wie oft das jeweilige Wort in den beiden Einträgen vorkommt. Auf diese Weise lässt sich das Ergebnis weiter verbessern, nimmt man an, dass häufige Wörter wie *die* oder *the* ein besserer Indikator als seltenere Wörter sind. In Zeile 12 und 13 wird die Summe der Wörter unter Berücksichtigung ihrer Häufigkeit durch die Anzahl der Wörter (ohne Berücksichtigung ihrer Häufigkeit) im jeweiligen Eintrag geteilt. Das Maximum beider Ergebnisse wird dann als Ergebnis der Scoring-Funktion zurück gegeben. Die Quotientenbildung findet an dieser Stelle statt, um einen Bezug zur Länge des korrespondierenden Eintrags herzustellen, da in einem längeren Text natürlich mehr Treffer für die Wörter aus der Schnittmenge

zu erwarten sind. Hierzu erneut ein einfaches Beispiel: Die Schnittmenge zweier Einträge sei lediglich das Wort “in”. Der erste Text bestehe dabei aus wenigen Worten, beispielsweise eine der obigen Schlagzeilen. Der zweite Eintrag sei relativ lang und habe sehr viele Vorkommnisse des Wortes “in”. Um nun nicht, wegen der hohen Trefferzahl davon auszugehen, dass beide Einträge in der gleichen Sprache verfasst sind (anhand des Wortes “in” lässt sich wohl keine Aussage treffen), wird der Quotient gebildet. Ein Beispiel für den Nutzen der Maximumsbildung könnte es sein, dass ein ausreichend langer Text mit beispielsweise 100 Wörtern und ein sehr langer Text, beispielsweise mit 1000 Wörtern, miteinander verglichen werden. Da die Schnittmenge aber im Allgemeinen durch den kürzeren Text beschränkt ist, wird für die Scoring-Funktion letztlich das “bessere” Ergebnis berücksichtigt.

Um die Performance der genutzten Scoring-Funktion zu zeigen, wurde ein einfache Test-Szenario konzipiert. Dabei wurde zu einem festen Zeitpunkt jeweils eine Ausschnitt von 50 beziehungsweise 20 Einträgen der Newsfeeds “manager magazin”, “FAZ.NET”, “ComputerBase News”, “fscklog”, “MacRumors : Mac News and Rumors”, “Le Monde.fr : à la Une” sowie “Le Figaro : A la Une” gewählt. Die Einträge waren dabei immer die 20 oder 50 aktuellsten, so dass auch inhaltlich ähnliche Artikel zwischen den einzelnen Nachrichten vorhanden waren. Für die Tests wurde jeweils ein Quervergleich von jedem Beitrag aus dem ersten Ausschnitt mit jedem Beitrag des jeweils anderen Ausschnitts durchgeführt. Insgesamt wurde die Scoring-Funktion also jeweils 400 beziehungsweise 2500 mal ausgewertet.

Verglichene Newsfeeds	**Ausschnitt**	**Durchschnitt**	**Minimum**	**Maximum**
“manager magazin” und “ComputerBase News”	20 Einträge	0,193	0,0	0,667
“manager magazin” und “ComputerBase News”	50 Einträge	0,183	0,0	0,667
“Le Monde.fr : à la Une” und “Le Figaro : A la Une”	20 Einträge	0,234	0,017	1,396
“Le Monde.fr : à la Une” und “Le Figaro : A la Une”	50 Einträge	0,226	0,0	1,468
“Le Monde.fr : à la Une” und “FAZ.NET”	20 Einträge	0,005	0,0	0,189
“Le Monde.fr : à la Une” und “FAZ.NET”	50 Einträge	0,006	0,0	0,189
“fscklog” und “MacRumors : News and Rumors”	20 Einträge	0,087	0,0	0,694
“fscklog” und “MacRumors : News and Rumors”	50 Einträge	0,092	0,0	0,718

Tabelle 3.1: Performance der vorgestellten Scoring-Funktion in verschiedenen Szenarien (auf drei Nachkommastellen gerundet)

In Tabelle 3.1 sind die Ergebnisse des Testszenarios dargestellt. Ohne zu betrachten, welche Newsfeeds miteinander verglichen wurden, ist festzustellen, dass der Unterschied zwischen 20 und 50 Einträgen in den getesteten Proben bei keinem der Szenarien einen erheblichen Unterschied ausmacht. Da der Test quadratische Laufzeit hat, ist dies erfreulich, da für das eigentliche Clustering später offenbar auch kleinere Proben ausreichen. Interessant sind nun vor allem die Durchschnittsergebnisse. Zunächst wurden mit "manager magazin" und "ComputerBase News" zwei Newsfeeds miteinander verglichen, die inhaltlich quasi keine Ähnlichkeit aufweisen, aber beide in deutscher Sprache veröffentlicht sind. Die Durchschnittswerte liegen hier knapp unter 0,2. Bei dem darauffolgenden Vergleich der Newsfeeds zweier französischer Nachrichtenmagazine erhöht sich der Wert etwas, auf knapp über 0,2, und am Maximumswert lässt sich ablesen, dass hier scheinbar auch Nachrichten miteinander verglichen wurden, die neben der sprachlichen auch eine hohe inhaltliche Ähnlichkeit aufwiesen, so dass die Schnittmenge hier nicht nur sprachtypische Wörter enthielt, sondern vermutlich auch Eigennamen oder feststehende Begriffe. Schließlich wurden noch zwei Vergleiche mit Newsfeeds in unterschiedlichen Sprachen durchgeführt. Dabei wurden zunächst zwei Nachrichtenmagazine, "Le Monde.fr" in französisch und "FAZ.NET" in deutsch, miteinander verglichen. Auch hier sind Artikel mit inhaltlicher Ähnlichkeit zu erwarten, jedoch liegt der Durchschnittswert mit höchstens 0,006 deutlich unter dem Wert für sprachlich gleiche Newsfeeds. Im letzten Test wurde einer der ungünstigsten Fälle getestet. In den beiden Newsfeeds "fscklog" und "MacRumors : News and Rumors" werden inhaltlich nahezu identische Themen behandelt, nämlich nur Nachrichten mit Bezug zur Firma Apple und den zugehörigen Produkten. Doch auch hier liegt der Durchschnittswert mit unter 0,1 immer noch relativ weit von den etwa 0,2 der sprachlich passenden Newsfeeds entfernt.

Als Resultat der obigen und weiterer Testergebnisse, wurde der Schwellwert für Sprachgleichheit zweier Newsfeeds auf 0,1 festgelegt. Es wird also von folgender Annahme für das Clustering ausgegangen: Zwei Newsfeeds sind wahrscheinlich in der gleichen Sprache veröffentlicht, wenn der durchschnittliche Scoring-Wert beim Quervergleich von zwei mindestens 20-elementigen Proben größer als 0,1 ist. Um ein weiteres Problem mit der Scoring-Funktion zu umgehen, wird außerdem gefordert, dass jeder Eintrag der 20-elementigen Stichprobe aus mindestens 40 Wörtern besteht. Es ist leicht zu sehen, dass der Vergleich von zwei Schlagzeilen wie beispielsweise "New in Stores" und "Brand in Nordhorn" zu einem Score von 0,3 führt, obwohl die Nachrichten dahinter vermutlich unterschiedliche Sprachen haben. Weitere Tests haben dabei gezeigt, dass bei Einträgen mit 40 oder mehr Wörtern gute Ergebnisse erzielt werden können.

Beim Vergleich mit den obigen Werten mag der Schwellwert 0,1 fast etwas zu niedrig gewählt sein, im eigentlichen Clustering wird dabei aber zusätzlich Sorge getragen, dass nicht versehentlich falsche Newsfeeds zu einem Cluster hinzugefügt werden. Dabei wird so vorgegangen, dass zunächst ein beliebiger Newsfeed ausgewählt wird, von dem eine mindestens 20-elementige Probe genommen werden kann. Um das Problem der inhaltlichen Gleichheit etwas zu verringern, werden dabei nicht die 20 letzten, sondern immer 20 zufällige Einträge ausgewählt. Der gefundene Newsfeed stellt nun den ersten Cluster da. Danach werden alle anderen Newsfeeds betrachtet, von denen wiederum jeweils eine 20-elementige Probe genommen wird. Um festzustellen, ob ein Newsfeed zum Cluster passt, wird wie oben das durchschnittliche Scoring der beiden Proben ermittelt. Bei Überschreitung des Schwellwerts wird der Eintrag dem Cluster hinzugefügt. Wird der Schwellwert nicht erreicht, wird zunächst geprüft, ob noch andere Cluster existieren, für die

das Verfahren wiederholt wird. Dabei erfolgt der Vergleich des aktuellen Newsfeeds immer mit einem zufälligen Element des betrachteten Clusters. Konnte kein Cluster gefunden werden, zu dem der Newsfeed hinzugefügt werden kann, wird dieser als Kandidat für einen neuen Cluster gespeichert.

Durch das zufällige Wählen eines Newsfeeds in einem Cluster, mit dem ein noch unklassifizierter Newsfeed verglichen werden soll, wird bereits eine gewisse Homogenität innerhalb eines Clusters erreicht. Um diese noch weiter zu erhöhen, muss ein Newsfeed, bevor er zu einem Cluster hinzugefügt wird, noch einen weiteren Test bestehen. Hat also der Vergleich des unklassifizierten Newsfeeds mit einem zufällig ausgewählten eines bestehenden Clusters zu einem Schwellwert größer 0,1 geführt, muss dieser Test mit fünf weiteren zufällig ausgewählten Newsfeeds des Clusters (sofern bereits fünf Newsfeeds zum Cluster gehören) wiederholt bestanden werden. So ist sicher gestellt, dass ein neues Mitglied des Clusters nicht nur zu einem, sondern mindestens zu sechs anderen Mitgliedern des Clusters passt.

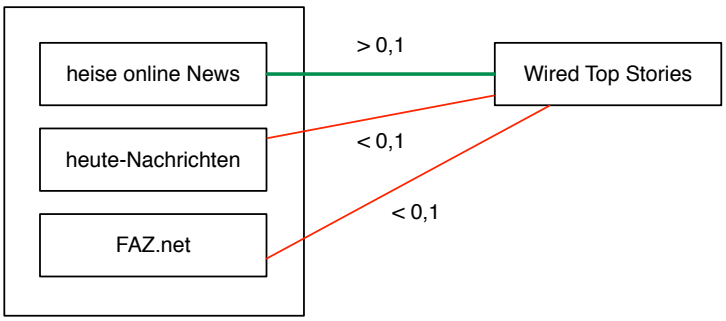


Abbildung 3.3: Visualisierung des Clustering-Algorithmus

Abbildung 3.3 visualisiert das Vorgehen. Die links dargestellten Newsfeeds sollen dabei bereits zu einem Cluster gehören. Unter Umständen kann es nun dazu kommen, dass ein neuer Feed, beispielsweise der des "Wired"-Magazins, den Schwellwert durch den Vergleich mit einem inhaltlich ähnlichen, aber sprachlich unterschiedlichen Newsfeed überschreitet, hier durch den "heise.de"-Newsfeed dargestellt. In einem solchen Fall ist es aber unwahrscheinlich, dass der Vergleich mit den übrigen Mitgliedern des Clusters auch zu einer Schwellwertüberschreitung führt, da vermutlich nicht alle Elemente des Clusters das gleiche inhaltliche Thema behandeln. Das Verfahren führt so auch zu einer frühen Fehlererkennung. Würde das Clustering beispielsweise mit dem "heise.de"-Newsfeed beginnen und der nächste betrachtete Newsfeed wäre der des "Wired"-Magazins, würde ein unerwünschter Cluster mit einem deutschen und einem englischen Newsfeed entstehen. Da aber jeder weitere Newsfeed, der zu diesem Cluster hinzugefügt werden soll, mit beiden Cluster-Mitgliedern verglichen wird, ist es relativ unwahrscheinlich, dass der Cluster sehr groß wird. Vermutlich wird er sogar nicht mehr als zwei Elemente beinhalten. Der Clustering-Prozess kann also regelmäßig sehr kleine Cluster wieder auflösen, so dass - durch die zufällige Wahl der Vergleichspartner - beide Newsfeeds des aufgelösten Clusters zu den "richtigen" Clustern hinzugefügt werden können.

Die Performance des Clustering-Algorithmus endgültig zu beurteilen, ist schwierig, da er sich auf die Anzahl der in Nuntios Datenbank eingetragenen Newsfeeds bezieht, die während des Testbetriebs lediglich 113 betrug. Davon konnten nur 79 mit dem Clustering-Algorithmus bearbeitet werden, da für die übrigen 39 zum Testzeitpunkt keine Probe mit mindestens 20 Elementen, die jeweils aus mindestens 40 Wörtern bestanden, gewählt werden konnte. Für die übrigen Newsfeeds sind drei Cluster mit 62 deutschen, 15 englischen und zwei französischen Einträgen erstellt worden, was auch exakt der Wirklichkeit entspricht. Ob der Algorithmus auch für größere Datenbestände so gut funktioniert, muss sich im weiteren Verlauf noch zeigen. Es lässt sich aber sagen, dass mit dem hier vorgestellten Ansatz ein performantes Online-Clustering implementiert wurde, das zumindest bisher sehr gute Ergebnisse bei akzeptabler Laufzeit geliefert hat. Nicht anwendbar ist die gesamte Idee des Algorithmus natürlich auf solche Newsfeeds, deren einzelnen Einträge in verschiedenen Sprachen verfasst sind, beispielsweise bei zweisprachigen Angeboten. Solche Newsfeeds sind zwar eher eine Seltenheit, würden aber wahrscheinlich - je nach zufälliger Auswahl beim Vergleich - zu einem Fehlerfaktor führen.

3.2.3 Ableiten von Wörterbüchern

Ist nun bekannt in welcher Sprache ein Newsfeed veröffentlicht wurde, ist es einfach möglich, für jeden der Cluster, die ja eine eigene Sprache repräsentieren, ein eigenes Wörterbuch anzulegen. Dazu werden in Nuntios Backend parallelisiert alle Einträge von Newsfeeds in Clustern durchlaufen und das Datenfeld `analyzable_text` (vgl. Abschnitt 3.2.1) wird genutzt, um den Inhalt eines solchen in seine Wortbestandteile zu zerlegen. Während das eigentliche Clustering nicht parallelisiert erfolgen sollte, da größere Cluster zu besseren Ergebnissen führen, ist es beim Ableiten der Wörterbücher wiederum möglich, viele Prozesse auf unterschiedlichen Systemen zu starten, die jeweils nur die Einträge der Newsfeeds bearbeiten, die über ihre Identifikationsnummer für das entsprechende System vorgesehen sind.

Bei jedem Wort aus `analyzable_text` wird nun geprüft, ob es groß oder klein geschrieben ist. Auf diese Weise sollen später Eigennamen erkannt werden. Das Wort selbst wird dann in eine klein geschriebene Form gebracht und in die Wörterbuch-Tabelle geschrieben. Trifft der Algorithmus also auf das Wort “Apple”, wird in der Datenbank zunächst geprüft, ob bereits ein Eintrag für das Wort “apple” für den Cluster, zu dem der Newsfeed des betrachteten Eintrags gehört, vorliegt. Falls nicht, wird dieser neu angelegt, wobei auch vermerkt wird, dass das Wort ursprünglich groß geschrieben war. Schließlich enthält die Tabelle ein Datenfeld für die Häufigkeit des Vorkommens des Wortes, das mit dem Wert 1 initialisiert wird. Trifft der Algorithmus später - im gleichen Cluster - auf das Wort “apple”, wobei diesmal die Frucht und nicht die Firma gemeint ist, wird die bereits vorhandene Zeile angepasst, wobei die Markierung als Eigenname gelöscht und der Zähler um eins erhöht wird. Als Eigennamen werden also nur die Wörter betrachtet, die *immer* groß geschrieben werden.

Soll der Algorithmus tatsächlich parallelisiert ausgeführt werden, muss dabei beachtet werden, dass nach Möglichkeit ein Transaktionsmanagement eingesetzt wird, da es sonst leicht zu dem klassischen *Lost Update*-Problem kommen kann, wenn konkurrierend auf den Datensatz des gleichen Wortes zugegriffen wird. Eine zu starke Parallelisierung dieses Algorithmus ist dabei sowie-

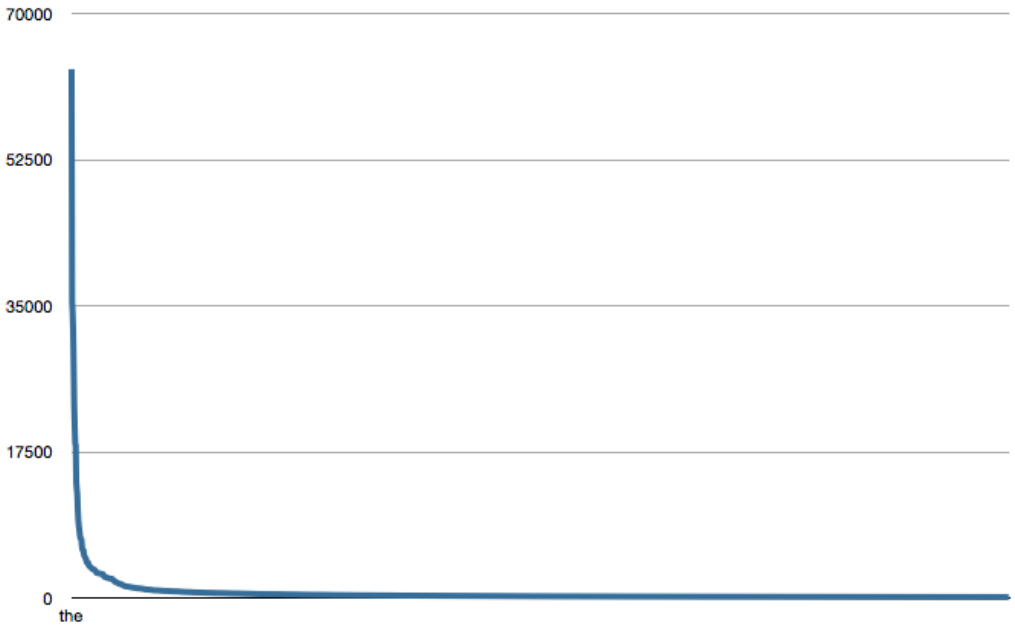
so nicht ratsam, da die Updates wegen der Master-Slave-Architektur ja immer in die Master-Datenbank geschrieben werden müssen, so dass der Schreibdurchsatz dieser leicht zum Flaschenhals werden kann.

3.2.4 Ableiten von relevanten Wörtern und Phrasen

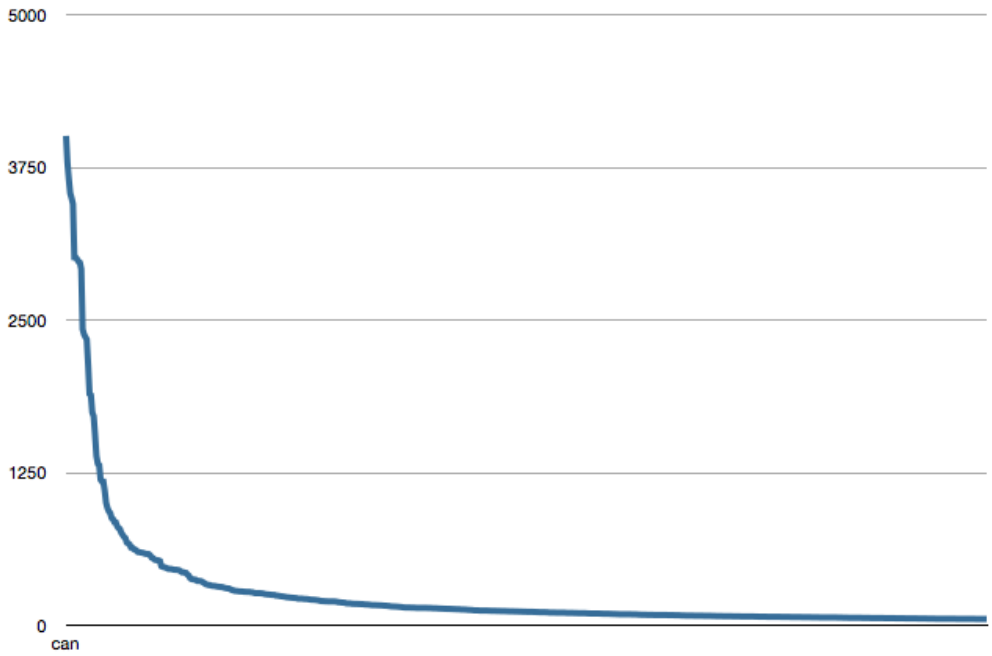
Da ständig neue Artikel in Nuntios Datenbank geladen werden, entsteht auf zuvor beschriebene Weise ein ständig aktuelles, sprachabhängiges Wörterbuch mit notierten Häufigkeiten der einzelnen Wörter. Zu einem Zeitpunkt während des Testbetriebs von Nuntio hatte das System bereits über sechs Millionen Wörter verarbeitet, wobei das Wörterbuch - über alle Sprachcluster gerechnet - über 200.000 Wörter umfasste.

Da nun bekannt ist, wie häufig ein Wort in einer Sprache vorkommt, können - wie in der Einleitung dieses Abschnitts beschrieben - Annahmen zur Relevanz eines Wortes getroffen werden. In den Abbildungen 3.4 sind dabei die Worthäufigkeiten in einem Diagramm visualisiert worden. In (a) sind alle Wörter im englischen Wörterbuch berücksichtigt und es ist deutlich zu erkennen, dass das Wort *the* mit einem absoluten Vorkommen von über 60.000 bei knapp über 1,5 Millionen registrierten Wortvorkommen das absolut häufigste Wort der Sprache darstellt. Selbst das nächsthäufigste Wort *a* hat bereits nur noch etwa die Hälfte der Registrierungen gegenüber *the* vorzuweisen. Um die Annahme, dass es eine Reihe von extrem häufigen Wörtern gibt, die für eine inhaltliche Analyse eines Textes keine Rolle spielen zu bestätigen, sind in (b) erneut die Worthäufigkeiten visualisiert, diesmal jedoch ohne die 30 häufigsten Begriffe. Zwar nicht ganz so extrem, aber immer noch deutlich zu sehen ist, dass weiterhin eine Reihe von sehr häufigen Begriffen einer deutlich größeren Menge von viel selteneren Begriffen gegenüber steht. Durch einen Blick in die Datenbank lässt sich dies bestätigen, da dort fast 30.000 unterschiedliche Wörter registriert sind, die über alle berücksichtigten englischen Newsfeeds lediglich fünfmal oder seltener registriert wurden, davon alleine 13.000, die lediglich ein einziges Mal aufgetaucht sind, was bei etwa 45.000 Wörtern im englischen Wörterbuch fast 30% ausmacht. Für das deutsche Wörterbuch gelten dabei ähnliche Verhältnisse: Hier sind knapp 155.000 unterschiedliche Wörter berücksichtigt, wobei rund 105.000 (also etwa zwei Drittel) fünfmal oder seltener vorkommen, knapp 42.000 Wörter - ebenfalls etwa 30% - sind nur einmal gezählt worden.

An dieser Stelle soll das eigentliche Ziel der hier geleisteten Vorarbeit wieder vor Augen geführt werden. Die inhaltliche Ähnlichkeit zweier Texte soll mit einem “Word Level” beziehungsweise “Multi Word Level”-Ansatz bestimmt werden. Da nun bekannt ist, dass manche Wörter extrem häufig in Texten vorkommen und manche extrem selten, kann man versuchen, auf Basis der Worthäufigkeit bestimmte Wörter eines Textes beim späteren Vergleich zu berücksichtigen und bestimmte zu vernachlässigen. Wie bereits in der Einleitung dieses Abschnitts angeklungen ist, ist es wohl wenig sinnvoll zu behaupten, dass zwei Texte inhaltlich ähnlich sind, weil in beiden das Wort “aus” vorkommt. Tatsächlich ist “aus” zum Testzeitpunkt das 25-häufigste Wort im deutschen Wörterbuch. Nun ist es relativ schwierig, harte Grenzen für die erlaubte Häufigkeit eines Wortes zu definieren, so dass dies noch als inhaltlich relevant betrachtet werden soll. Sicherlich unproblematisch wäre es zu behaupten, dass die 30 häufigsten Wörter zu vernachlässigen sind. Genauso könnte man sagen, dass Wörter, die lediglich fünfmal oder seltener registriert wurden,



(a) Worthäufigkeit im englischen Wörterbuch



(b) Worthäufigkeit im englischen Wörterbuch ohne die 30 häufigsten Begriffe

Abbildung 3.4: Grafische Darstellung der Worthäufigkeiten

besonders wichtig sind. Letztere Aussage ist dabei problematisch, da dazu beispielsweise auch das Wort "Lokomotivführertarifvertrag" zählen würde, das sogar nur ein einziges Mal gezählt wurde, aber für den zugehörigen Nachrichten-Eintrag sehr wohl relevant ist. Der Begriff *relevant* soll also so verwendet werden, dass damit die Relevanz für die Suche nach ähnlichen Einträgen gemeint ist. Kommt ein Wort also sehr selten vor, hilft es nichts es zu berücksichtigen, da kaum ein anderer Artikel gefunden werden kann, in dem das Wort auch auftaucht. Problematisch ist auch, dass die Wörterbücher ständig aktualisiert werden und so nahezu jedes Wort irgendwann die vorgeschlagene harte 5er-Grenze überschreitet. Sinnvoller wäre es also, die Häufigkeit eines Wortes in Relation zu allen anderen Wörtern zu betrachten. Hilfreich für das Extrahieren von relevanten Wörtern aus den einzelnen Wörterbüchern wäre also eine Aussage wie die Folgende: Die obersten X und die untersten Y Prozent der Wörter im Wörterbuch können vernachlässigt werden.

Wort	**Platzierung**	**Prozentual**
merkel	186	0,24%
westerwelle	633	0,82%
gabriel	1221	1,57%
özdemir	5023	6,47%
steuer	1422	1,83%
chile	622	0,8%
irak	597	0,77%
wahl	585	0,75%
google	324	0,42%
ipad	608	0,78%
pizza	10868	13,93%

Tabelle 3.2: Häufigkeiten und Platzierungen ausgewählter Begriffe

Um eine geeignete Auswahl treffen zu können, muss also betrachtet werden, in welchem Häufigkeitsbereich die relevanten Wörter vorzufinden sind. In Tabelle 3.2 sind dabei eine Reihe von Begriffen und ihre Platzierung im nach Häufigkeit sortierten deutschen Wörterbuch angegeben. Die prozentuale Angabe ist dabei als *in den obersten x Prozent* zu lesen. Dabei ist wichtig, dass für die prozentualen Angaben nur Wörter berücksichtigt wurden, die mindestens dreimal gezählt wurden. Wie oben bereits beschrieben wurde, bringt diese absolute Begrenzung auf Dauer betrachtet zwar keinen Vorteil, Fantasiewörter oder durch Tippfehler entstandene Einträge können auf diese Weise trotzdem umgangen werden. Natürlich hängt die dargestellte Platzierung nun vor allem davon ab, auf Basis welcher Newsfeeds das Wörterbuch überhaupt entstanden ist. Wären beispielsweise nur Nachrichten zu politischen Themen in der Datenbank vorhanden, würde das Wort "ipad" kaum einen derart hohen Rang haben. Je größer die Vielfalt der Newsfeeds einer Sprache ist, desto besser setzen sich die irrelevanten Begriffe von den relevanten ab. Im Testbetrieb von Nuntio wurden gute Ergebnisse erzielt, wenn die obersten 0,5% der Wörter in jeder Sprache vernachlässigt wurden. Hier muss nun entschieden werden, ob besser möglicherweise

relevante Wörter verloren gehen sollen - wie hier der Begriff “merkel” oder “google” - oder ob entsprechend mehr irrelevante Begriffe berücksichtigt werden sollen. Der häufigste Begriff, der mit obiger Grenze berücksichtigt würde, ist beispielsweise “zwar”, der ebenfalls kaum als relevant angesehen werden kann. Eine geeignete untere Grenze zu finden, ist möglicherweise noch schwieriger. Im Testbetrieb von Nuntio wurden testweise die nächsten 30% der Wörter als relevant betrachtet. Dadurch ergab sich eine absolute Grenze von 15 nötigen Vorkommnissen eines Wortes, um als relevant zu gelten, was insgesamt etwa 30.000 Wörter - über die drei vorhandenen Sprachen gerechnet - für die weiteren Betrachtungen zur Folge hatte.

In Nuntios Backend steht also ein weiteres Programm zur Verfügung, das in bestimmten Intervallen aus den einzelnen Wörterbüchern für die Sprachen ein neues Wörterbuch mit diesen relevanten Wörtern erzeugt. Für jede Sprache werden dabei separat die oben genannten 30% der Wörter ausgewählt und das vorhandene Wörterbuch relevanter Begriffe entsprechend ergänzt. Wörter, die wegen der Verlegung der Grenzen nun als nicht mehr relevant gelten, werden schließlich gelöscht. An diese Stelle sollte auch deutlich werden, warum es überhaupt nötig ist, die einzelnen Sprachcluster separat zu behandeln. Würde man dies nicht tun, könnte man keine Aussagen über die Relevanz eines Wortes auf Basis von dessen Häufigkeit treffen. In Nuntio sind während des Testbetriebs beispielsweise deutlich mehr deutsche als englische Newsfeeds vorhanden gewesen. Würde man einfach über alle Wörter die entsprechenden 30% wählen, wären darin vermutlich fast alle irrelevant häufigen Begriffe aus dem Englischen enthalten, da sie es wegen des Übergewichts der deutschen Begriffe nicht in das obere halbe Prozent geschafft hätten aber immer noch häufig genug wären, um nicht durch die Untergrenze herausgefiltert zu werden.

In der Einleitung dieses Abschnitts wurde beschrieben, dass Artikel untereinander auf Grund der vorkommenden Wörter (“Word Level”), aber auch auf Grund von Begriffen, also Phrasen (“Multi Word Level”), miteinander verknüpft werden sollen. Mit dem nun vorhandenen Wörterbuch kann im Inhalt eines Artikels nun leicht geprüft werden, welche einzelnen Wörter als relevant gelten und welche nicht, es bleibt aber das Problem der Phrasen. Da eine Phrase mindestens aus zwei Wörtern besteht, kann man sich die obige Vorauswahl der Wörter zu Nutze machen, indem man einfach annimmt, dass eine relevante Phrase aus zwei oder mehr aufeinanderfolgenden relevanten Wörtern besteht. Dabei verliert man jedoch einige relevante Phrasen, wie “Karl-Theodor zu Guttenberg”, da hier das Wort “zu” vermutlich zu häufig auftritt, um als relevant zu gelten. Der große Vorteil dieses Vorgehens ist aber, dass sich Phrasen beim späteren Verknüpfen von Artikeln relativ leicht identifizieren lassen, während eine Analyse, die auch obige Fälle abdeckt, deutlich komplizierter und mit geringerer Performance umzusetzen wäre. Trotzdem kann eine Identifikation von Phrasen mit obiger Definition erst dann erfolgen, wenn das Wörterbuch mit den relevanten Wörtern bereits besteht, in Nuntios Backend erfolgt dies während einzelne Artikel mit ihren relevanten Wörtern verknüpft werden.

3.2.5 Verknüpfung von Wörtern und Phrasen mit Artikeln

Da nach den vorherigen Schritten ein Wörterbuch mit relevanten Begriffen in Nuntios Datenbank vorliegt, kann jetzt die Vorarbeit zum eigentlichen Ziel geleistet werden - das Verknüpfen der Artikel untereinander. Dazu werden für jeden Eintrag eines Newsfeeds die Schlagwörter, auch *Tags* genannt, gewählt, die im jeweiligen Text vorkommen. Erneut wird also der `analyzable_text` eines jeden Eintrags durchlaufen, wobei nun für jedes erreichte Wort geprüft wird, ob es sich um ein gerade als relevant betrachtetes Wort handelt oder eben nicht. Durch Indizierung der Datenbank wird ein Wort dabei in logarithmischer Laufzeit gefunden, so dass der Aufwand hier vertretbar ist. Ist ein relevantes Wort gefunden, gilt es eine effizient abfragbare Datenstruktur anzulegen und das relevante Wort so mit dem Artikel zu verknüpfen. In Nuntio erfolgt dies auf Ebene der Datenbank.

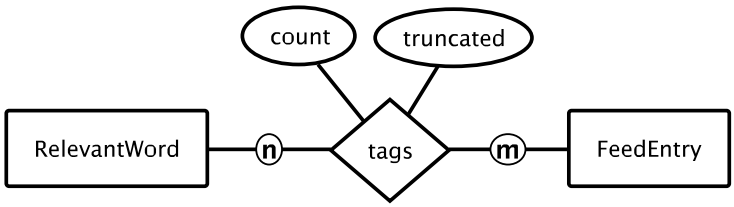


Abbildung 3.5: Verknüpfung von `RelevantWord`s und `FeedEntry`s in ER-Notation (Ausschnitt)

In Abbildung 3.5 ist dabei dargestellt, wie relevante Wörter und Einträge von Newsfeeds miteinander verbunden werden. Da es sich um eine N:M-Relation mit eigenen Attributen handelt, wird jede Verknüpfung eines Artikels mit einem relevanten Wort in einer eigenen Tabelle gespeichert. Das Attribut `count` gibt dabei an, wie häufig ein Wort in einem Artikel vorkommt. Wird also der Begriff "ipad" mehrfach gefunden, wird die Verknüpfung nicht doppelt angelegt sondern lediglich der Wert von `count` erhöht.

Um das Attribut `truncated` zu erklären, muss zunächst auf ein weiteres Problem bei dem bisherigen Ansatz eingegangen werden, denn bei einer Verknüpfung auf Wortebene werden flektierte Formen von Wörtern im Allgemeinen nicht miteinander verknüpft. Beispielsweise würde in der Schlagzeile "Merkels neuer Gesetzesentwurf" zwar das Wort "merkels" gefunden werden, aber durch das angehängte *s* würde der Algorithmus davon ausgehen, dass damit ein anderes Wort als "merkel" - beispielsweise in der Schlagzeile "Merkel stellt neues Gesetz vor" - gemeint ist. Um dieses Problem zu umgehen oder zu verringern, gibt es verschiedene Ansätze - in [Joa02] wird hierzu beispielsweise auf die n-Gram-Darstellung von Wörtern verwiesen. Dabei wird jedes Wort durch n Zeichen lange Wortfragmente repräsentiert, um auch flektierte Formen miteinander verbinden zu können. Bei der 3-Gram-Darstellung würde das Wort "merkels" beispielsweise durch "_me", "mer", "els" und "ls_" dargestellt, während "merkel" zu "_me", "mer", "kel" und "el_" würde. Immerhin die beiden Anfänge würden hier miteinander verknüpft werden. Auch

wenn sich hierdurch eine gewisse Verbesserung ergibt, entstehen zusätzliche Probleme, möchte man später ein Ähnlichkeitsmaß definieren. Beispielsweise würde hier jeder Artikel mit obigem verknüpft werden, in dem ein Wort, das mit "me" beginnt, vorkommt. Zudem muss beachtet werden, dass die Nutzer später ja noch die Möglichkeit haben sollen, die Schlagworte zu bewerten, um die Ergebnisse der Ähnlichkeitsanalyse zu verbessern. Sicher fällt es einem Nutzer dabei leichter, anzugeben, dass der Begriff "merkel" für den Artikel relevant ist, als zu sagen, dass das Fragment "_me" wichtig ist. Ein anderer, in [Por97] vorgestellter Ansatz, stellt fest, dass Wörter häufig den gleichen Wortstamm haben und durch einen Suffix in eine flektierte Form gebracht werden. Dort wird ein geeigneter Algorithmus vorgestellt, der sich allerdings an der englischen Grammatik orientiert, um Wörter auf einen gemeinsamen Wortstamm abzubilden. Während der dort vorgestellte Algorithmus dabei Buchstabenvorkommnisse in den Suffixen überprüft, wurde für Nuntio ein ähnliches, aber deutlich einfacheres Verfahren genutzt. Wird ein Wort bei der Analyse erreicht, wird nicht nur geprüft, ob dieses Wort relevant ist, es wird zusätzlich geprüft, ob das Wort aus mindestens sechs Zeichen besteht. Ist dies der Fall, wird zunächst das letzte Zeichen des Wortes abgeschnitten und für das neu entstandene Wort ebenfalls geprüft, ob es relevant ist. Falls ja, wird der Artikel nicht nur mit dem Ausgangsbegriff verknüpft, sondern auch mit der verkürzten Form. Außerdem wird das Wort erneut um ein Zeichen gekürzt, wobei auch hier gegebenenfalls eine weitere Verknüpfung hergestellt wird. Für obiges Beispiel würde die erste Schlagzeile also nicht nur mit dem Begriff "merkels" - sofern dieser relevant ist - verknüpft sondern auch mit dem Wort "merkel" (ein Buchstabe weniger) und dem Wort "merke" (zwei Buchstaben weniger). Wiederum natürlich nur, wenn diese bereits als relevant klassifiziert sind. Natürlich macht man es sich hier relativ leicht, aber das einfache Verkürzen der Begriffe sorgt zumindest im Deutschen und Englischen dafür, dass eine Vielzahl an flektierten Formen auf einen gemeinsamen Stamm gebracht werden können. Die Tatsache, dass die Verknüpfung über eine Verkürzung eines Begriffes erfolgt ist, wird dabei im Attribut `truncated` vermerkt. Hier wird aber ein Nachteil deutlich, da das Wort "merke" zwar ein sinnvolles Wort ist, aber vermutlich keine Relevanz für den Artikel mit obiger Schlagzeile hat. Es kommt also unter Umständen zu unerwünschten Verknüpfungen. Während des Testbetriebs von Nuntio hat sich aber gezeigt, dass die Vorteile in diesem Fall überwiegen.

Die Verknüpfung von Phrasen mit Artikeln erfolgt dabei während der Wort-Analyse. Werden zwei aufeinanderfolgende Wörter für relevant gehalten, wird geprüft, ob bereits eine aus diesen Wörtern bestehende Phrase in der Datenbank vorhanden ist, falls nicht, wird diese erstellt und der Artikel mit dem Eintrag verknüpft. Besteht die Phrase bereits wird der Eintrag der Phrase, nicht die Verknüpfung, in ihrem Häufigkeitswert, der in einem separaten Attribut gespeichert ist, erhöht und es erfolgt ebenfalls eine Verknüpfung. Obige Wortverkürzungen werden dabei nicht berücksichtigt, es wird aber zusätzlich vermerkt, ob alle Teilwörter groß geschrieben waren, was vermuten lässt, dass es sich um einen Namen oder feststehenden Begriff handelt.

In Tabelle 3.3 sind hierzu die zehn häufigsten Begriffe während des Testbetriebs von Nuntio aufgelistet. Insgesamt sind dabei rund 27.000 Phrasen mit knapp 190.000 Vorkommnissen in der Datenbank registriert gewesen. An diesem Beispiel wird dabei deutlich, dass auch Phrasen erkannt werden, die mehr als zwei relevante Wörter beinhalten. Stößt der Algorithmus also auf eine Schlagzeile wie "Neue Blu-Ray-Disc-Veröffentlichungen im November" werden bei der Vorverarbeitung die Trennstriche entfernt, so dass der Verknüpfungs-Algorithmus beispielsweise

Platzierung	Phrase	Anteil in Promille
1	champions league	5,2
2	blu ray	4,4
3	wall street	3,3
4	blu ray disc	2,8
5	europa league	2,8
6	los angeles	2,5
7	barack obama	2,0
8	angela merkel	1,9
9	street view	1,7
10	san francisco	1,7

Tabelle 3.3: Die zehn häufigsten Phrasen in Nuntios Datenbank (Werte gerundet)

einzeln die Wörter “blu”, “ray” und “disc” findet. Sind alle drei im Wörterbuch als relevant markiert, wird zunächst die Phrase “blu ray” eingetragen. Da aber “disc” ebenfalls relevant ist, wird nun auch “blu ray disc” eingetragen und der Artikel schließlich mit beiden Phrasen verknüpft.

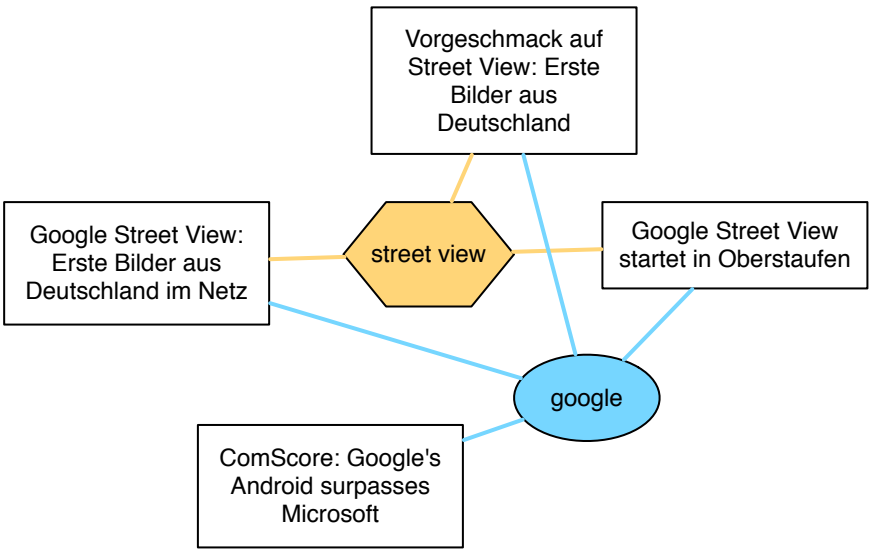


Abbildung 3.6: Verknüpfung von Artikeln über Phrasen und relevante Wörter

In 3.2 - der Einleitung dieses Abschnitts - wurde die Idee, die inhaltliche Ähnlichkeit von Artikeln auf Basis von den darin enthaltenen Wörtern und Phrasen zu bewerten, anhand von drei Beispiel-Artikeln erörtert. In Abbildung 3.6 sind die drei Schlagzeilen der Artikel sowie die eines weiteren in Form eines Graphen visualisiert. Die Darstellung lässt sich dabei als beispielhafte Ausprägung der beiden N:M-Relationen zwischen Artikeln und Phrasen beziehungsweise Wörtern auffassen. Da die Verknüpfungen persistent in der Datenbank vorliegen, ist es nun sehr einfach - und vor allem effizient - herauszufinden, welche Einträge mit der Phrase "street view" verknüpft sind. Gleiches natürlich auch für ein Wort wie "google". Betrachtet man obigen Graphen, drängt es sich quasi auf, die Vermutung zu formulieren, dass sich zwei Artikel desto mehr ähneln, je mehr verschiedene Verbindungen es zwischen ihnen gibt. In obigem Beispiel können die drei Artikel zu Google Street View über jeweils zwei Pfade, nämlich das Wort "google" und die Phrase "street view" erreicht werden, während der Artikel über Googles Android-Betriebssystem jeweils nur über einen Pfad erreichbar ist.

3.2.6 Verknüpfung von Artikeln untereinander

Nach Ablauf des im vorherigen Abschnitt erläuterten Algorithmus ist in der Datenbank des Backends eine effiziente Datenstruktur entstanden, die die einzelnen Artikel miteinander verknüpft. Der letzte Schritt, um eine Aussage über die inhaltliche Ähnlichkeit zweier Nachrichten oder Einträge im Allgemeinen zu machen, ist es nun, ein Ähnlichkeitsmaß zu definieren und letztlich anzuwenden.

In der Literatur sind hierzu verschiedene Ansätze diskutiert, wobei - bei Analyse auf Wortebene - häufig ein so genannter "bag-of-words"-Ansatz genutzt wird (vgl. [Joa02]). Einen solchen Ansatz mit der Erweiterung quasi um einen "bag-of-phrases" verfolgt auch der Algorithmus in Nuntio. Dieser besteht letztlich darin, einen Text als Menge von Wörtern aufzufassen, wobei die Reihenfolge keine Rolle spielt. Auf diese Weise lässt sich ein Text als ein n-dimensionaler Vektor betrachten, wobei jeder Eintrag des Vektors für die Häufigkeit eines bestimmten Wortes oder einer Phrase im Text steht. Im vorgestellten Datenbank-Zustand von Nuntio würde n also der Anzahl der relevanten Wörter plus der Anzahl der Phrasen entsprechen, also knapp 60.000 Einträge haben. Liegen zwei Texte in einer solchen Darstellung vor, lässt sich beispielsweise der Winkel zwischen den beiden Vektoren als Ähnlichkeitsmaß nutzen. In [HMS01] wird dies als *cosine distance* bezeichnet. Wenn also d_1 der erste und d_2 der zweite Text in Vektor-Form ist, wäre

$$sim(d_1, d_2) = cos(d_1, d_2) = \frac{d_1 \cdot d_2}{\|d_1\| * \|d_2\|}.$$

Da diese 60.000-dimensionalen Vektoren im Wesentlichen mit Nullen gefüllt wären, ist eine tatsächliche Repräsentation der Texte als Vektor wenig sinnvoll, zumal durch obige Struktur bereits sehr einfach herauszufinden ist, an welchen Stellen des Vektors überhaupt größere Zahlen stehen würden. Schließlich muss beachtet werden, dass die Aufgabe ja nicht nur darin besteht, für zwei gegebene Artikel eine Aussage über die Ähnlichkeit zu machen, sondern auch und vor allem darin, für *einen* gegebenen Artikel herauszufinden, welche Artikel überhaupt als *ähnlich* in Frage

kommen. Natürlich können auch andere Ähnlichkeitsmaße definiert werden. Die Idee, die hinter obigem Maß steht, ist allerdings durchaus nachvollziehbar. So wird durch das Skalarprodukt das Produkt der Worthäufigkeiten in der Schnittmenge der Wörter beziehungsweise Phrasen aufsummiert und durch den Quotienten in Relation zur Länge der beiden Texte gesetzt. Ein ähnlicher aber etwas umfangreicherer Ansatz wird in Nuntio verfolgt.

Um das in Nuntio genutzte Ähnlichkeitsmaß zu erklären, sollen an dieser Stelle noch einmal zwei der in der Einleitung dieses Abschnitts vorgestellten Nachrichtenmeldungen genutzt werden.

Google Street View: Erste Bilder aus Deutschland im Netz veröffentlicht von ad-hoc-news.de am 02.11.2010 08:42:18 UTC mit dem Inhalt: "Berlin (dpa) - Google Street View hat erste Straßenbilder aus Deutschland ins Internet gestellt. Damit will Google einen ersten Vorgeschmack auf seinen Online-Straßenatlas geben. Es geht zunächst aber nur um sechs Sehenswürdigkeiten, zehn Bundesliga-Stadien und ..."

Vorgeschmack auf Street View: Erste Bilder aus Deutschland veröffentlicht von n-tv.de am 02.11.2010 06:59:43 UTC mit dem Inhalt: "Der Fotodienst des Suchmaschinenbetreibers Google öffnet seine deutschen Pforten mit einem Schnupperangebot. Google Street View zeigt erste Aufnahmen aus Deutschland im Internet."

Zunächst besteht hier der Vorteil, dass nicht zwei reine Textdokumente miteinander verglichen werden müssen, sondern noch mehr Informationen zur Verfügung stehen. So ist beispielsweise bekannt, welche Wörter in der Überschrift vorkommen und welche im eigentlichen Text, zudem ist jeder Eintrag mit einem Veröffentlichungs- oder zumindest Eintragsdatum und Informationen über den Betreiber des Newsfeeds versehen. Diese Informationen können für das Ähnlichkeitsmaß genutzt werden.

Angenommen, für zwei Nachrichten wie die obigen soll bestimmt werden, wie groß ihre inhaltliche Ähnlichkeit ist. Dazu kann nun von Abbildung 3.6 ausgegangen werden, wobei zunächst die Schnittmenge der relevanten Wörter und Phrasen beider Nachrichten bestimmt wird, also bezüglich obiger Abbildung die Wort- beziehungsweise Phrasen-Knoten, die die beiden Nachrichten miteinander verbinden. Tatsächlich ist es so, dass die einzelnen Kanten dabei den Einträgen in der Datenbank entsprechen wobei jeder Datenbank-Eintrag einen Artikel mit einem Wort verknüpft und jeder Datenbank-Eintrag dabei außerdem mit den in Abschnitt 3.2.5 beschriebenen Zusatzinformationen versehen ist. So könnte man jede Kante zusätzlich mit der Information versehen, wie oft ein Wort oder eine Phrase in dem jeweiligen Artikel auftaucht und ob es tatsächlich auftaucht oder durch Verkürzung anderer Begriffe entstanden ist.

In Tabelle 3.4 ist das Ergebnis einer auf Datenbankebene effizient durchführbaren Abfrage für die beiden Beispiel-Nachrichten dargestellt. Bevor nun mit diesen Werten gerechnet werden kann, werden einige Grundannahmen gemacht. Die erste Annahme ist dabei, dass mit der Länge eines Wortes oder einer Phrase auch dessen Informationsgehalt steigt. Für diese Annahme kann man natürlich beliebige Beispiele finden, allerdings auch beliebige Gegenbeispiele. Insbesondere bei einzelnen Wörtern ist dies problematisch, bei Phrasen fällt es leichter, einen solchen Ansatz zu

verfolgen. Im Testbetrieb von Nuntio wurden mit diesem Ansatz aber gute Ergebnisse erzielt, so dass die Annahme Bestandteil des Algorithmus geblieben ist. Weiterhin wird angenommen, dass ein Wort, das durch Verkürzung entstanden ist, weniger Gewicht haben sollte, als ein Wort das nicht künstlich verkürzt wurde. Schließlich ist leicht nachzuvollziehen, dass Wörter, die auch in der Überschrift eines Artikels erscheinen oder als Eigennamen gekennzeichnet sind, ein höheres Gewicht haben sollten.

Wort	**Häufigkeit A**	**Verkürzt in A**	**Häufigkeit B**	**Verkürzt in B**
google	4	Nein	3	Nein
street	3	Nein	2	Nein
view	3	Nein	2	Nein
internet	2	Nein	1	Nein
interne	2	Ja	1	Ja
intern	2	Ja	1	Ja
vorgeschmack	1	Nein	1	Nein

Tabelle 3.4: Verknüpfungen der beiden Beispiel-Artikel

Mit diesen Annahme wird nun ein Basis-Wert für das Ähnlichkeitsmaß bestimmt. Ist w ein Wort der Schnittmenge und $l(w)$ die Länge des Wortes, dann wird für jedes Wort ein Grundwert $g(w) = l(w) + (l(w) - 4) \cdot 0{,}1$ festgelegt. Dabei wird angenommen, dass ein Wort mit 4 Zeichen eine angemessene Länge hat, kürzeren Wörtern wird pro fehlendem Zeichen 0,1 in ihrem Wert abgezogen, längeren 0,1 addiert. Folglich wäre $g('google') = 6{,}2$. Handelt es sich bei dem Begriff um einen Eigennamen, wird der Basis-Wert um 10% erhöht, also mit 1,1 multipliziert, taucht das Wort auch in der Überschrift auf, wird der Wert ebenfalls mit 1,1 multipliziert. Ist das Wort durch künstliche Verkürzung entstanden, wird der Wert mit 0,9 multipliziert, also verringert. Ob die Berechnung des Basiswerts und die Erhöhung oder Verringerung des Wertes in einem guten Verhältnis stehen, darf man hier durchaus in Frage stellen. Im Testbetrieb sind mit diesen Werten jedoch gute Ergebnisse erzielt worden.

Der errechnete Basis-Wert wird nun mit einem Gewicht für die Häufigkeit des Auftretens in beiden Artikeln in Relation zur Gesamtlänge der Artikel multipliziert. Dabei kann zunächst davon ausgegangen werden, dass ein Wort oder eine Phrase, die sehr oft in einem Artikel vorkommt, auch sehr wichtig ist. Da die Wörter und Phrasen, die dabei berücksichtigt werden, aber - wie in Abschnitt 3.2.4 beschrieben - automatisiert bestimmt wurden, kann es sein, dass beide Artikel über häufig auftretende, aber irrelevante Begriffe miteinander verknüpft sind. Um dieses Problem abzudämpfen wird der Basiswert nicht einfach der Häufigkeit des Vorkommens gewichtet. Ist h die Häufigkeit des Auftretens eines Begriffs in einem Artikel, dann ist $gew(h) = \sqrt{h} \cdot 2{,}5 - 1{,}5$. So gilt weiterhin, dass $gew(1) = 1$ ist, aber durch den Wurzel-Term wird ein unverhältnismäßig häufiges Auftreten gedämpft. $gew(4)$ - für den Begriff *google* im ersten Artikel - wäre also 3,5, $gew(3)$ wäre etwa 2,83, für den gleichen Begriff im zweiten Artikel. Die Summe beider Werte wird schließlich noch ins Verhältnis zur Länge des Textes gesetzt, da bei längeren Texten Begriffe auch öfter vorkommen können. Auch hier wird ein Wurzel-Term eingesetzt, der zur Dämpfung bei sehr

langen Texten führt. Ist len_1 die Länge des ersten Textes und len_2 die Länge des zweiten, wird das errechnete Gewicht durch $\sqrt{len_1 + len_2} \cdot 20$ geteilt. Ist die Summe von len_1 und len_2 also kleiner als 400 wird der Term größer als die Summe, bei Werten darüber wird der Term kleiner. Der Wurzel-Term wurde an dieser Stelle eingeführt, um das Ähnlichkeitsmaß beim Vergleich von sehr langen Texten mit sehr kurzen zu verbessern, da die Summe hier zwar groß ist, aber trotzdem nur wenige Verknüpfungen vorhanden sein können - diese werden schließlich durch den kürzeren Text limitiert. Der Schwellwert 400 wurde hier erneut experimentell bestimmt. Ist also $b(w_1)$ der im vorherigen Absatz berechnete Basis-Wert für die Verknüpfung des Wortes mit dem ersten Artikel und $b(w_2)$ der Basis-Wert für die Verknüpfung mit dem zweiten Artikel sowie h_1 die Häufigkeit des Wortes im ersten und h_2 die Häufigkeit im zweiten Artikel, dann ist

$$\frac{b(w_1) + b(w_2)}{2} \cdot \frac{gew(h_1) + gew(h_2)}{\sqrt{len_1 + len_2} \cdot 20}$$

der Gesamtwert für die Verknüpfung. Die Einzelwerte für jedes Wort der Schnittmenge werden dann aufsummiert, so dass für die obigen Beispielartikel ein Wert von rund 0,946 errechnet wird.

Phrase	**Häufigkeit A**	**Häufigkeit B**
google street	2	2
google street view	2	2

Tabelle 3.5: Verknüpfungen der beiden Beispiel-Artikel durch Phrasen

Für Phrasen, wie beispielhaft in Tabelle 3.5 angegeben, wird nun genauso verfahren und ein eigener Wert ermittelt, der für die beiden obigen Nachrichten bei rund 0,705 liegt. Die Länge der beiden Phrasen sorgt hier für den vergleichsweise hohen Wert.

Danach wird als Hilfswert das in Abschnitt 3.2.2 vorgestellte Maß zur Ähnlichkeit der Sprache der zwei Nachrichten angewandt. Wie in diesem Abschnitt auch anhand von Beispielen beschrieben wurde, ist das dort vorgestellte Maß deutlich besser geeignet, um festzustellen ob zwei Nachrichten in der gleichen Sprache geschrieben wurden. In Tabelle 3.1 war aber auch zu sehen, dass beim Vergleich zwei inhaltlich ähnlicher Newsfeeds deutlich höhere Werte erzielt werden konnten als bei thematisch verschiedenen - auch wenn diese in unterschiedlichen Sprachen verfasst waren. Eine Berücksichtigung dieses Maßes sorgt außerdem dafür, dass Wörter, die fälschlicherweise als nicht relevant gelten, trotzdem zumindest in gewissem Maße berücksichtigt werden. Zwischen den beiden oben angegebenen Nachrichten wird beispielsweise ein Wert von 1,47 erreicht, der im Vergleich mit den Durchschnittswerten aus Tabelle 3.1 außergewöhnlich hoch ist.

Bevor die einzelnen Werte nun zu einem Gesamtwert, der letztlich das Ähnlichkeitsmaß darstellt, verrechnet werden, wird schließlich noch die Zeitdifferenz herangezogen. Hierzu kann man die Annahme formulieren, dass Beiträge, die zeitlich dichter zusammen liegen, vermutlich eher inhaltlich ähnlich sind als Beiträge, die mit großer Zeitdifferenz veröffentlicht wurden. Die Zeit wird aber - da sie keine Information über den tatsächlichen Inhalt liefert - lediglich als Verstärkungskomponente genutzt. Dabei wird das Ähnlichkeitsmaß um maximal 5% erhöht, wenn beide

Artikel zeitgleich veröffentlicht werden. Der Prozentsatz sinkt dann über einen Zeitraum von 48 Stunden linear auf 0% ab.

Um nun den Gesamtwert zu berechnen, werden zwei Überlegungen angestellt. Zunächst wird berücksichtigt, dass der Fall auftreten kann, dass zwei Artikel zwar über wenig verschiedene Phrasen und Wörter miteinander verknüpft sind, diese aber außergewöhnlich häufig auftreten. Andererseits kann es sein, dass sich zwei Artikel extrem viele Wörter und Phrasen teilen, aber jedes Wort beziehungsweise jede Phrase selbst nur ein- oder zweimal auftaucht. Um dem ersten Fall gerecht zu werden, wird der durchschnittliche Wert einer Verknüpfung berechnet, für obigen Fall also $\frac{0{,}946}{7}$ für die relevanten Wörter und $\frac{0{,}705}{2}$ für die Phrasen. Letzterer Fall ist bereits durch die Werte selbst berücksichtigt. Die letzte Aufgabe besteht also darin, die vorhandenen Werte so miteinander zu verrechnen, dass sie in einem vernünftigen Verhältnis zueinander stehen. Besonders schwer ist dies, da für keinen der Werte ein Intervall festgelegt oder eine Normalisierung durchgeführt wurde. Letztlich wurde folgendes Gesamtmaß experimentell bestimmt: Wenn $a \in [1, 1.05]$ die Zeitkomponente ist, w der Gesamtwert der Wortverknüpfungen, p der Gesamtwert der Phrasen-Verknüpfungen, w_{avg} der durchschnittliche Wert der Wortverknüpfungen, p_{avg} der durchschnittliche Wert der Phrasen-Verknüpfungen und s das Ergebnis der eigentlich sprachbezogenen Scoring-Funktion, dann hat sich

$$sim(artikelA, artikelB) = a \cdot (w + p + 2 \cdot (w_{avg} + p_{avg})) + s$$

als geeigneter Gesamtwert herausgestellt. Für die beiden obigen Nachrichten ergibt sich hiernach ein Gesamtwert von rund 5,38. Natürlich ist dieser Zahlwert ohne einen entsprechenden Kontext wenig wert, so dass zunächst gar nicht gesagt werden kann, ob die Artikel nun inhaltlich ähnlich sind oder eben nicht.

Um den Zahlwert in einen geeigneten Kontext zu bringen, sind in Tabelle 3.6 die Ergebnisse der Ähnlichkeitsanalyse für den ersten der beiden Beispielartikel dargestellt. Es wurde dazu das Ähnlichkeitsmaß für alle Artikel in Nuntios Datenbank ermittelt, die maximal zwei Wochen vor dem verglichenen Artikel erschienen sind, insgesamt rund 35.000 während des Testbetriebs. In Tabelle 3.6 sind dabei zunächst die besten 10 Ergebnisse, sortiert nach Wert des Ähnlichkeitsmaßes dargestellt, wobei für alle Artikel ein Wert von über 4 errechnet wurde. In der Auflistung würden nun noch mehr Artikel folgen, wobei abgesetzt in der Tabelle der Eintrag vermerkt ist, der - nach subjektiver Einschätzung - noch als inhaltlich ähnlich bezeichnet werden kann, aber den niedrigsten Ähnlichkeitswert erreicht hat. Schließlich sind die Artikel, die - nach subjektiver Einschätzung - als nicht ähnlich angesehen werden müssen, aufgelistet. Was an diesem Beispiel deutlich wird ist, dass die Ähnlichkeitsanalyse mit dem oben vorgeschlagenen Ähnlichkeitsmaß - zumindest in diesem Fall - immerhin eine klare Trennung von Artikeln mit inhaltlicher Ähnlichkeit und ohne ermöglicht. Für den Algorithmus muss nun also erneut ein “threshold”, also ein Schwellwert ermittelt werden, anhand dessen eine Aussage über die inhaltliche Ähnlichkeit von zwei Artikeln getroffen werden kann. Leider zeigt dieses Beispiel auch, dass es schwierig ist, hier einen Wert festzulegen, da die relevanten Artikel mit Werten größer als 1,7 sehr nah an den irrelevanten Artikeln mit Werten kleiner oder gleich 1,7 liegen. Außerdem kann man den Schwellwert natürlich nicht auf Basis eines einzigen Tests festlegen. Während des Testbetriebs von Nuntio

Titel der Meldung	**Ähnlichkeit**
Google Street View: Erste Bilder aus Deutschland im Netz	5.38
Google Street View startet in Oberstaufen	5.2
Google Street View: Ein Vorgeschmack für Deutschland	4.71
Straßenatlas im Web: Google veröffentlicht erste Street-View-Bilder in Deutschland	4.55
10 Stadien, 7 Hotspots - Deutschland-Start für Google Street View	4.42
Google Street View - So geht es weiter in Deutschland	4.38
Google "Street View" startet Bilderdienst	4.37
Google Street View zeigt erste Bilder aus Deutschland	4.37
Google: Erste deutsche Street-View-Bilder kommen aus dem Allgäu	4.26
Wandern Sie durch Stadien der Bundesliga	4.26
Datenschützer: Thilo Weichert legt Gesetzentwurf zum Datenschutz vor	1.72
Werbung: Google verschenkt Internet über den Wolken	1.7
Sony Internet TV Blu-ray Disc Player with Google TV review	1.63
Sony Internet TV with Google TV review	1.51
Cloud Computing: United Internet steigt bei Profitbricks ein	1.5
BGH: Internetportal darf Amateurspiele zeigen	1.48
BGH: Internetportal "Hartplatzhelden" darf Amateurspiele zeigen	1.37
HOWTO explain the Internet to a Dickensian street urchin	1.0

Tabelle 3.6: Ergebnisse der Ähnlichkeitssuche in Nuntios Datenbank

wurde der Wert experimentell auf 1,5 festgelegt, wobei damit relativ gute Ergebnisse erzielt wurden. Für dieses Beispiel würden allerdings drei Nachrichten falsch klassifiziert werden. Hier muss man sich erneut entscheiden, ob lieber mehr Ergebnisse allerdings mit der Gefahr falscher Klassifizierung erreicht werden sollen oder weniger Ergebnisse mit weniger Fehlern. In [ZC08], wo ein ähnliches Thema behandelt wird, wird dies als Konflikt zwischen den wünschenswerten Eigenschaften *complete* und *accurate* der Ergebnisse bezeichnet.

Wie weiter oben bereits erwähnt wurde, besteht die Aufgabe des Algorithmus zur Ähnlichkeitsanalyse nicht nur darin, ein möglichst gutes Ähnlichkeitsmaß festzulegen, sondern auch darin, dieses in effizienter Manier auf eine große Datenbasis anwenden zu können. In Abschnitt 3.2.5 wurde dazu die Verknüpfung der Artikel mit den relevanten Wörtern auf Ebene der Datenbank

vorgestellt. Die dort angelegten Datenstrukturen können nun genutzt werden, um die in Tabelle 3.6 dargestellten Ergebnisse performant zu erlangen. Ist eine Ausgangsnachricht festgelegt, können die zugehörigen relevanten Wörter und Phrasen bei entsprechender Indizierung in logarithmischer Laufzeit, bezogen auf die Gesamtanzahl von vorhandenen Verknüpfungen in der Datenbank, erreicht werden. Für jedes gefundene relevante Wort beziehungsweise Phrase können wiederum in logarithmischer Laufzeit alle übrigen Artikel gefunden werden, die ebenfalls mit diesem Begriff verknüpft sind. Auf diese Weise kann die Tabelle 3.4 mit den Join-Ergebnissen - diesmal jedoch über alle Artikel - erzeugt werden. Von den vormals 35.000 Artikeln sind dabei bereits alle diejenigen aussortiert, die keine Schnittmenge in den relevanten Wörtern mit dem Ausgangsartikel haben. Gruppiert man nun über die Ergebnisse für je zwei Artikel, lässt sich mittels der Summen-Aggregatsfunktion einfach feststellen, wie viele Übereinstimmungen es - unter Berücksichtigung der Wort-Häufigkeit - zwischen zwei Artikeln gibt. Tabelle 3.4 würde also zu Tabelle 3.7 aggregiert werden.

Artikel A	**Artikel B**	**Verknüpfungen**
"Google Street View: [...]"	"Vorgeschmack auf Street View [...]"	33

Tabelle 3.7: Aggregation der Ergebnisse aus den Tabellen 3.4 und 3.5

Sortiert nach der Anzahl der Verknüpfungen könnte hier bereits eine wirkungsvolle Vorauswahl der Nachrichten getroffen werden, für die das tatsächliche Ähnlichkeitsmaß bestimmt wird. Würde man also in Abbildung 3.6 den Kanten jeweils den Wert der Häufigkeit hinzufügen, wäre das Ergebnis genau die Summe der abgelaufenen Kanten, wenn von Artikel A jede mögliche Verbindung zu Artikel B berücksichtigt wird. Dabei würde man beispielsweise von den entstehenden Ergebnissen die ersten 100 berücksichtigen und nur zu diesem Ausschnitt das Ähnlichkeitsmaß bestimmen. Je größer die Stichprobe dabei ist, desto besser wird das Ergebnis werden. Da der Aufwand für die Berechnung des Ähnlichkeitsmaßes aber sehr hoch ist, wäre es wünschenswert, wenn bereits auf Ebene der Datenbank eine möglichst gute Vorauswahl getroffen werden könnte. Dazu wird der bei der Berechnung des Ähnlichkeitsmaßes herangezogene Wort- beziehungsweise Phrasen-Wert teilweise bestimmt:

$$\frac{b(w_1) + b(w_2)}{2} \cdot \frac{gew(h_1) + gew(h_2)}{\sqrt{len_1 + len_2} \cdot 20}.$$

Um die Abfrage nicht unnötig kompliziert zu gestalten, wird dabei auf $b(w)$, was ja die Wortlänge, Eigennamen und Vorkommnis in der Überschrift berücksichtigt hat, verzichtet. Der resultierende Quotient kann aber ohne größere Performance-Einbußen berechnet werden, da sowieso über die Häufigkeiten (h_1 beziehungsweise h_2) summiert werden muss. Auf diese Weise wird in der Join-Tabelle (vgl. Tabelle 3.7) nicht einfach die Anzahl der Verknüpfungen unter Berücksichtigung der Häufigkeit zur Vorsortierung herangezogen, sondern bereits ein Element des späteren Ähnlichkeitsmaßes, das sich im Testbetrieb von Nuntio als guter Approximator für die spätere Sortierung herausgestellt hat. In der Implementierung von Nuntio wird dabei für die 50 höchstbewerteten Artikel dieser Vorauswahl jeweils der Wert des gesamten Ähnlichkeitsmaßes bestimmt, wobei dem Nutzer davon später nur die 10 besten Ergebnisse präsentiert werden.

Wegen der doch vergleichsweise hohen Laufzeit der Ähnlichkeitsanalyse werden die Ergebnisse in eine separate Datenbanktabelle - also eine neue N:M-Relation zwischen Einträgen von Newsfeeds - gespeichert, um im Frontend nicht *on demand* berechnet werden zu müssen. An dieser Stelle sei noch ein Nachteil des obigen Ablaufs erwähnt. Da bei der Analyse immer nur die Nachrichten mit der aktuell betrachteten verknüpft werden können, die bereits in Nuntios System vorhanden sind, treten Ketten-Effekte auf. Landen also die inhaltlich ähnlichen Nachrichten A, B und C nacheinander im System, wird A mit keiner anderen Nachricht verknüpft, da B und C noch nicht vorhanden sind. B wird mit A verknüpft, aber nicht mit C. Lediglich der letzte Eintrag eines Themas kann mit allen vorherigen Einträgen verknüpft werden. Um dieses Problem zu umgehen, werden Nachrichten, die miteinander verknüpft wurden, für eine erneute Analyse markiert. Kommt also B ins System und wird mit A verknüpft, so wird ein Datenbankattribut des Eintrags von A gesetzt, so dass ein separater Prozess A erneut analysiert und nun auch B finden kann. Um keine Endlos-Schleifen zu erzeugen, setzt der Reanalyseprozess das Attribut jedoch nicht erneut. C würde nun B und A finden, so dass beide erneut analysiert würden. Während des Testbetriebs hielt sich die Last für die erneute Analyse in Grenzen. Um die in den weiteren Abschnitten beschriebene Nutzereinflussnahme zu berücksichtigen, wurde jeder Artikel der letzten 48 Stunden sogar unabhängig von der Markierung stündlich erneut analysiert. Jedoch ist abzusehen, dass hier Probleme auftreten können. Hierzu wäre es möglich, die Nachrichten für die Reanalyse über einen Zeitraum wie etwa einer Stunde zu sammeln und dann in einem Stück abzuarbeiten. Auf diese Weise würde verhindert, dass ein Eintrag im gleichen Zeitraum ständig neu analysiert werden muss.

Um die Ergebnisse aus Tabelle 3.6 zu berechnen, benötigte das Testsystem (vgl. auch Abschnitt 3.2.7) durchschnittlich 0,61 Sekunden. Diese Zahl ist natürlich schwer zu beurteilen und von extrem vielen Faktoren abhängig, zeigt aber, dass die Ähnlichkeitsanalyse in einem angemessenen Zeitrahmen durchzuführen ist. Zudem soll an dieser Stelle noch einmal auf die Architektur von Nuntio verwiesen werden (vgl. Abschnitt 3.1.2), da gerade bei diesem letzten Schritt die Master-Slave-Architektur extreme Vorteile mit sich bringt. Für obigen Algorithmus muss dabei eine extrem aufwändige Query vom Datenbank-Server verarbeitet werden und die Berechnung des Ähnlichkeitsmaßes belastet den Rails-Server entsprechend stark. Es ist aber so, dass die Ähnlichkeitsanalyse von Artikel A völlig unabhängig davon ablaufen kann, ob gerade auch für Artikel B oder C nach ähnlichen Einträgen gesucht wird. Für die Suche relevant sind dabei immer nur die Artikel, die bereits durch die in Abschnitt 3.2.5 vorgestellte Datenstruktur miteinander verbunden sind. Der hier vorgestellte Ansatz zur Ähnlichkeitsanalyse ist also durch Parallelbearbeitung nahezu beliebig skalierbar. Sollte also der Fall eintreten, dass die durchschnittliche Zeit für die Analyse größer wird als das durchschnittliche Zeitintervall zwischen dem Erscheinen neuer Einträge, kann einfach ein weiteres Slave-System eingesetzt werden, das die Performance nahezu verdoppeln müsste.

3.2.7 Performance-Betrachtungen

Nachdem nun die grundlegenden Prozesse des Backends beschrieben wurde, stellt sich natürlich die Frage nach der Performance des Systems. Dabei kann man zwei Dinge unterscheiden: Zunächst die Frage, ob die eingesetzten Algorithmen überhaupt in einem angemessenen Zeitrahmen ablaufen und damit zeitnah Ergebnisse liefern können, sowie die Frage, ob die beschriebenen Verfahren überhaupt sinnvolle Ergebnisse liefern.

Zunächst sollen an dieser Stelle einige Ergebnisse zu ersterer Frage nach der Laufzeit auf der genutzten Hardware dargestellt werden. Hierzu muss gesagt werden, dass die einzelnen Tests jeweils direkt auf dem Master-Datenbank-Server durchgeführt wurden, so dass es nicht zu einer Verzögerung durch die Replikation kommen konnte. Außerdem war jeweils nur der angegebene Test aktiv, so dass der Server sonst nicht genutzt wurde. Bei den Tests, die die Vorteile der Parallelisierung zeigen sollen, wurden jeweils mehrere Instanzen eines Prozesses gestartet, um die spätere Verteilung auf physikalisch unterschiedliche Systeme anzunähern. Dazu wurde ein Intel Xeon-Server-System mit E5530 Prozessor, also mit vier unabhängigen Prozessorkernen genutzt.

Performance-Test	**Instanzen**	**Zeit [s]**	**Zeit pro Eintrag [s]**
Laden, Parsen und Updaten von 113 Newsfeeds mit 2328 Einträgen	1	32,2	0,014
Wortanalyse für Update der Wörterbuchhäufigkeiten für 400 Einträge	1	115,4	0,289
Wortanalyse für Update der Wörterbuchhäufigkeiten für 400 Einträge	4	46,8	0,117
Ähnlichkeitsanalyse von 400 Einträgen bei Berücksichtigung von 31729 potentiellen Partnern	1	251,5	0,629
Ähnlichkeitsanalyse von 400 Einträgen bei Berücksichtigung von 31729 potentiellen Partnern	4	63,2	0,158

Tabelle 3.8: Ergebnisse einfacher Performance-Tests

Natürlich können die Zeitangaben der in Tabelle 3.8 angegebenen Test-Ergebnisse nicht für Vergleiche herangezogen werden, da sie von extrem vielen Faktoren abhängig sind - sie sollten also eher als Beispielwerte betrachtet werden. Die Ergebnisse zeigen aber, dass die implementierten Algorithmen geeignet sind, um in einem System, das Daten in nahezu Echtzeit verarbeiten muss, eingesetzt zu werden. Das Laden und Parsen der Newsfeeds sowie das zugehörige Datenbankupdate konnte dabei so performant durchgeführt werden, dass die Verarbeitungszeit für einen Nachrichteneintrag bei nur etwas über einer hundertstel Sekunde lag. Bei der Wortanalyse, die ja jedes Wort eines Nachrichteneintrags einzeln betrachten muss, konnten immerhin Zeiten von deutlich unter einer Sekunde erreicht werden. Hier zeigt sich auch, dass die Prozesse gut parallelisierbar sind, da der Wechsel von einer auf vier parallele Instanzen etwa zu einer 2,5-fachen Verbesserung der Performance geführt hat. Bei der Betrachtung der Ähnlichkeitsanalyse ist dies noch

dramatischer. Da hier, wie oben ja bereits beschrieben wurde, deutlich mehr Lese- beziehungsweise Rechenaufwand als Schreibaufwand erzeugt wird, konnte quasi eine echte Vervierfachung der Performance erreicht werden. Dabei ist besonders erfreulich, dass die Ähnlichkeitsanalyse der (zeit-)aufwändigste Prozess innerhalb von Nuntio, aber dafür auch am besten zu parallelisieren ist.

In den Testergebnissen oben sind dabei nicht alle beschriebenen Prozesse berücksichtigt, wobei die Berechnung der nötigen Verarbeitungszeit pro Nachricht meist auch nicht sinnvoll ist. Ein Update der relevanten Wörter hat bei verschiedenen Test beispielsweise rund dreieinhalb Minuten gedauert, hing aber in den Tests hauptsächlich davon ab, wie viele Wörter gelöscht werden mussten, unabhängig davon wie viele Newsfeeds in der Datenbank vorhanden oder zuletzt analysiert waren. Das Clustering nach Sprache dauerte beispielsweise für einen Newsfeed etwa zweieinhalb Sekunden, was allerdings wiederum von der Anzahl der Cluster, hier nur drei, abhängig ist.

Da die Laufzeit der Algorithmen offenbar für den Einsatzzweck des Systems angemessen ist, kann nun noch versucht werden zu klären, ob die Analyse-Algorithmen überhaupt sinnvolle Ergebnisse liefern. In den obigen Abschnitten wurden dazu beispielsweise beim Clustering bereits Angaben gemacht, wobei man natürlich sagen muss, dass die betrachtete Stichprobe von 113 Newsfeeds keine sonderlich aussagekräftigen Ergebnisse liefern kann. Aussagen über die Qualität der Ergebnisse der Ähnlichkeitsanalyse sind dabei ähnlich schwierig zu treffen - nicht weil zu wenig Daten vorliegen, sondern weil die Daten dazu bereits vorklassifiziert sein müssten, damit eine geeignete Vergleichsbasis entsteht. Da, wie auch in Abschnitt 4.1 zu möglichen weiterführenden Arbeiten beschrieben, im engen Zeitrahmen dieser Arbeit keine händische Klassifikation einer ausreichend großen und repräsentativen Menge von Einträgen durchgeführt wurde, soll an dieser Stelle nur ein Testszenario vorgestellt werden, anhand dessen gezeigt werden soll, dass das eingesetzte Verfahren zur Ähnlichkeitsanalyse durch die Vorverarbeitung von relevanten Wörtern bereits ohne die Berücksichtigung der in den folgenden Abschnitten beschriebenen Einflussnahme der Nutzer zumindest in machen Situationen überlegen sein kann.

Es handelt sich dabei um die sprachübergreifende Ähnlichkeitsanalyse. Durch die Tatsache, dass bei der Vorverarbeitung bereits eine große Menge an häufigen aber irrelevanten Wörtern aus dem Analyse-Prozess entfernt wird, bleiben im Optimalfall noch einige sprachabhängige Wörter, die möglicherweise auch als relevant zu bewerten sind, aber auch viele Eigennamen und feststehende Begriffe übrig. Da feststehende Begriffe sprachübergreifend verknüpft werden können, ist hier eine bessere Performance als beispielsweise mit dem klassischen Kosinus-Maß (vgl. Abschnitt 3.2.6) zu erreichen. Für einen Test wurde der englische Artikel “Red Hat Enterprise Linux 6 improves scalability, virtualization” analysiert. Der Artikel bestand dabei aus 128 Wörtern, wobei Nuntio davon immerhin 50 als relevant eingestuft hat. Das sind offensichtlich deutlich zu viele, aber durch die Tatsache, dass ohne die Vorverarbeitung jeder Begriff als relevant eingestuft würde, sind die Ergebnisse trotzdem besser. Das Kosinus-Maß findet unter den knapp über 32.000 betrachteten Einträgen immerhin den deutschen Eintrag “Red Hat Enterprise Linux 6 ist fertig” mit dem besten Ergebnis, worauf 24 Einträge mit englischen inhaltlich völlig unterschiedlichen Themen folgen. Erst dann wird der (immerhin als ähnlich zu bewertende) Eintrag “Betaversion von Red Hat Enterprise Linux 5.6 freigegeben” gefunden. Der nächste tatsächlich ähnliche Eintrag landet dann auf Platz 189. Nuntios Ähnlichkeitsanalyse listet die vier - also alle - zu findenden

deutschen Ergebnisse, die als inhaltlich ähnlich zu werten sind auch auf den ersten vier Plätzen der Suchergebnisse und dies mit Ähnlichkeits-Ratings zwischen 2 und 3.

Natürlich lässt sich eine eventuelle Verbesserung nicht an einem Beispiel festmachen - obgleich hierzu mehrere gefunden werden könnten. Für eine aussagekräftige Bewertung müsste in weiterführender Arbeit eine Untersuchung mit Hilfe von Testdaten erfolgen, wie auch im späteren Abschnitt 4.1 vorgeschlagen wird.

3.3 Verbesserung der Ergebnisse durch Nutzereinflussnahme

Auf Basis des in den vorherigen Abschnitten vorgestellten Backends könnte bereits ein Frontend mit hoher Nutzbarkeit aufgebaut werden, doch in der Einleitung dieses Kapitels wurde die Möglichkeit der sozialen Interaktion der Nutzer gefordert. Die bisher vorgestellten Algorithmen arbeiten jedoch völlig automatisiert - zwar auf Basis der eingegebenen Newsfeeds der Nutzer - aber unabhängig von ihrem sonstigen Einfluss. In [GH06] beschäftigen sich die Autoren mit dem Thema "collaborative tagging", also der Verschlagwortung von Daten auf Basis der Angaben, die die einzelnen Nutzer einer Plattform machen. Viele Systeme, wie das in obigem Artikel vorgestellte Delicious-System[3], setzen dabei in ihren Analysen ausschließlich auf die Angaben, die Nutzer zu Daten machen. Wegen der extremen Schnelllebigkeit der Nachrichtenmeldungen im Internet ist es aber eher unwahrscheinlich, dass ein solcher, rein auf Nutzereingaben basierender, Ansatz für ein System wie Nuntio funktionieren würde. Um ähnliche Artikel zu finden, müssten Nutzer ständig Artikel per Hand mit Schlagworten versehen, was in einem System, das eher auf den Konsum von Informationen ausgelegt ist, vermutlich schlicht nicht in ausreichendem Maße genutzt würde. Gerade auch im Hinblick auf ein Gerät wie das iPad, ist davon auszugehen, dass die Eingabe von Text eine weitere Hürde für einen solchen Ansatz darstellt.

In Nuntio wurde deshalb ein anderer Weg verfolgt, um Nutzer ähnlich wie beim "collaborative tagging" auf die Verknüpfung von Artikeln Einfluss nehmen zu lassen. Auf diese Weise sollen die Nutzer die Möglichkeit bekommen, die bestehenden Algorithmen zu verbessern, so dass sich langfristig ein Mehrwert für alle Nutzer ergibt. Letztlich wäre es unklug, den Nutzern keine Möglichkeit einzuräumen, dem System Feedback zu den getroffenen Verknüpfungen zu geben, da es - wie bereits in der Einleitung angeklungen ist - für einen Menschen einfach ist, bereits anhand der Schlagzeilen von Artikeln eine häufig richtige Aussage über die inhaltliche Ähnlichkeit zu machen. Das "collective knowledge" [Gru08] der Nutzer soll also nicht außer Acht gelassen werden. Die beiden implementierten Möglichkeiten und zwei weitere Ansätze sollen in den folgenden vier Abschnitten beschrieben werden.

[3] http://del.icio.us

3.3.1 Tag-Matching

Durch die Vorverarbeitung in Nuntios Backend ist jeder Artikel bereits automatisiert mit einer Reihe von Schlagworten oder *Tags* versehen worden, damit der Analyse-Algorithmus auf dieser Basis die einzelnen Artikel miteinander verknüpfen kann. Da die Verschlagwortung allein auf Grund der Annahme gemacht wurde, dass sehr häufige sowie sehr seltene Begriffe irrelevant sind, ist es wahrscheinlich, dass die automatisch zugewiesenen Schlagworte für den Artikel von unterschiedlicher Relevanz sind.

Für einen der in Abschnitt 3.2.6 beispielhaft vorgestellten Artikel zu Google Street View wurden in Nuntio auch die Begriffe “vorgeschmack”, “öffnet”, “pforten” und “internet” als Schlagwörter vorgesehen, außerdem die durch Verkürzung entstandenen Begriffe “interne” und “intern”. Als Mensch würde man vermutlich keinen der Begriffe als wirklich relevant für den Artikel ansehen. In einer nicht ganz so absoluten Sicht könnte man vielleicht sagen, dass das Wort “internet” zwar relevant ist, aber nicht so relevant, wie beispielsweise das Wort “google”, das ja auch ausgewählt wurde.

In Nuntios Frontend sollen Nutzer die Möglichkeit haben, die automatisiert ausgewählten Schlagwörter zu sehen und auf möglichst einfache Art und Weise ihre Zustimmung oder eben Ablehnung für einen solchen Tag zum Ausdruck zu bringen. Auf diese Weise müssen Benutzer keine Texteingabe durchführen, aber in Nuntios Datenbank kann nun zusätzlich berücksichtigt werden, wie viele Nutzer ein Schlagwort für wichtig oder eben unwichtig halten. Jede Verknüpfung zwischen einem Schlagwort und einem Artikel wird also um zwei Attribute erweitert, in denen jeweils die Zahl der Für- beziehungsweise Gegenstimmen vermerkt ist. Um das System möglichst einfach zu halten, hat ein Nutzer dabei genau eine Stimme pro Artikel, Schlagwort und Meinung, so dass eine versehentlich falsche Angabe rückgängig gemacht, aber nicht doppelt für oder gegen einen Begriff gestimmt werden kann.

Wort	**Fürstimmen**	**Gegenstimmen**	**Gewicht**
google	26	15	1,37
internet	5	13	0,9
interne	0	5	0,5
intern	1	8	0,3
vorgeschmack	0	15	0,0
pforten	2	18	0,0
öffnet	15	19	0,87

Tabelle 3.9: Beispielhafte Nutzerwertungen ausgewählter Schlagwörter

In Tabelle 3.9 ist ein beispielhafter Ausschnitt aus Nuntios Datenbank dargestellt. Dabei sind sowohl Für- als auch Gegenstimmen vermerkt sowie ein aus diesen Werten resultierendes Gewicht. Soll nun also bestimmt werden, wie relevant ein Schlagwort auf Basis der Nutzereingaben ist, muss eine geeignete Funktion gesucht werden, die später in den Analyse-Algorithmus einfließen kann.

```
def self.calc_user_relevance(users_for, users_against)

  if(users_for + users_against < 4)
    return 1
  else
    if(users_for > users_against)
      return 1 + [(users_for - users_against) / [2.0 * users_against, 10.0].max, 1.0].min
    else
      return 1 - [(users_against - users_for) / [2.0 * users_for, 10.0].max, 1.0].min
    end
  end

end
```

Listing 3.7: Einflussfunktion für Nutzerangaben in Ruby

In Listing 3.7 ist die genutzte Funktion in der Ruby-Implementierung dargestellt. Abhängig von der Anzahl der Für- und Gegenstimmen (`users_for`, `users_against`) wird ein Wert aus dem Intervall $[0, 2]$ ermittelt. Völlig irrelevante Begriffe sollen also gar nicht mehr berücksichtigt werden, sehr wichtige Bergriffe sollen mit dem doppelten Gewicht eingehen. Außerdem soll berücksichtigt werden, dass in einem bestimmten Bereich beider Werte keine gute Aussage über die Relevanz gemacht werden kann. Das ist zunächst der Fall, wenn sehr wenig Nutzer an der Abstimmung teilgenommen haben, im Testbetrieb von Nuntio wurden mindestens vier Stimmen gefordert, damit überhaupt eine Aussage getroffen werden kann. Ist dies nicht der Fall, wird das Gewicht auf 1, also neutral, belassen. Keine gute Aussage ist außerdem möglich, wenn das Verhältnis etwa gleich ist, also ähnlich viele Stimmen für einen Begriff abgegeben wurden wie dagegen. Wie oben zu sehen ist, wurde dazu ein Quotient eingefügt, der jeweils die Gegenseite berücksichtigt. Beim Begriff "google" in Tabelle 3.9 ist beispielsweise zu sehen, dass 26 Stimmen für das Wort abgegeben wurden, aber auch 15 dagegen - vielleicht weil viele das Wort für zu allgemein hielten. Es sind nun zwar mehr Stimmen für das Wort abgegeben worden, aber auch viele dagegen. Um das Gewicht in einem solchen Fall etwas abzudämpfen, wird die obige Funktion genutzt. Letztlich sagt diese aus, dass um den maximalen beziehungsweise minimalen Wert zu erreichen, die Differenz aus Für- und Gegenstimmen mindestens doppelt so groß werden muss, wie das Doppelte der jeweiligen Gegenanzahl, mindestens aber 10. Praktisch bedeutet das also, dass der Begriff "google" bei 15 Gegenstimmen mindestens 45 Fürstimmen braucht, um ein Gewicht von 2 zu erhalten. Bei 10 oder mehr Gegenstimmen ist dies gleichbedeutend damit, dass ein Begriff mindestens drei Mal so viele Fürstimmen braucht wie Gegenstimmen vorhanden sind.

In Tabelle 3.9 sind dazu die ermittelten Gewichte angegeben, die relativ einfach in der Analyse genutzt werden können. In dem in Abschnitt 3.2.6 beschriebenen Ähnlichkeitsmaß wird der Wert einer Wortverknüpfung als

$$\frac{b(w_1) + b(w_2)}{2} \cdot \frac{gew(h_1) + gew(h_2)}{\sqrt{len_1 + len_2} \cdot 20}$$

definiert. An dieser Stelle lässt sich nun einfach ein Term für das durch Nutzereinfluss festgelegte Gewicht einbauen. Ist also $n(w)$ das mit obiger Funktion ermittelte Gewicht des Wortes w, dann

wird als Wert einer Wortverknüpfung fortan

$$\frac{b(w_1) + b(w_2)}{2} \cdot \frac{gew(h_1) + gew(h_2)}{\sqrt{len_1 + len_2} \cdot 20} \cdot n(w_1) \cdot n(w_2)$$

herangezogen. Ist ein Wort also in beiden Artikeln sehr wichtig, wird das Gewicht mit 4 multipliziert, ist er in einem Artikel als irrelevant markiert, wird der Gesamtwert der Wortverknüpfung unabhängig von der Häufigkeit auf 0 gesenkt.

Neben dieser Anpassung lässt sich weiterhin annehmen, dass sich zwei Nachrichten weniger ähneln, wenn in Artikel A ein Begriff sehr wichtig ist, aber in B gar nicht vorhanden. Bei der derzeitigen Umsetzung des Algorithmus würde einfach kein Wert berechnet werden, da das betreffende Wort nicht in der Schnittmenge beider Artikel vorhanden ist. Wenn nun aber klar ist, dass ein Begriff tatsächlich sehr wichtig ist, kann man noch weiter gehen und das Fehlen in Artikel B "bestrafen". Im Fall von Nuntio wird hierbei von der Summe der Wortwerte der Wert des fehlenden Begriffs nach obiger Gleichung (jedoch ohne das Gewicht durch die Nutzer) abgezogen.

Ein letztes Problem besteht darin, dass der Einfluss der Nutzer durch obigen Ansatz durchaus signifikante Änderungen beim Finden von ähnlichen Artikeln haben kann, was ja auch erwünscht ist. In Abschnitt 3.2.6 wurde aber erklärt, dass zur Berechnung des Ähnlichkeitsmaßes eine Vorauswahl aus der Datenbank gesucht wird. Um dem Einfluss der Nutzer hier gerecht zu werden, muss das Gewicht also auch in die dort formulierte Query einfließen, was auf Seite von MySQL erfreulicherweise durch die bedingten Konstrukte mit `if(fw1.users_for + fw1.users_against < 4, 1, ...)` möglich ist.

Für den angeführten Beispielartikel würde obige Nutzergewichtung dazu führen, dass die zuvor in Tabelle 3.6 dargestellten fehlerhaft klassifizierten Artikel ganz aus der Auflistung verschwinden und durch andere relevante Artikel ersetzt werden. Außerdem haben alle Artikel der 50-elementigen Vorauswahl nur einen Ähnlichkeitswert von 1,5 oder höher, es hat also - und das gilt nicht nur für diesen Beispielartikel - eine echte Verbesserung stattgefunden.

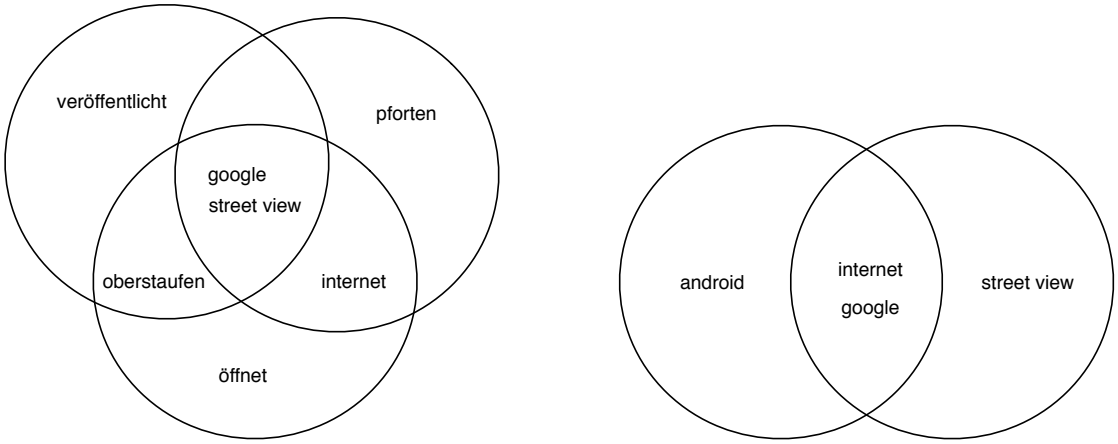
3.3.2 Artikel-Matching

Nachdem Nutzer nun Einfluss auf die Relevanz eines Schlagwortes nehmen können, stellt sich die Frage, warum Nutzer nicht auch ein Feedback zu den Ergebnissen der Gesamtanalyse geben können sollten. Werden also im Frontend später die einzelnen Artikel angezeigt, die laut Analyse inhaltlich zu dem ausgewählten Artikel passen, ist es einfach, den Nutzern die Möglichkeit zu geben, eine Aussage über die tatsächliche Ähnlichkeit der Artikel zu geben. Wie bereits erwähnt, dürfte es einem Menschen leicht fallen, bereits an der Schlagzeile eines Artikels zu erkennen, ob hier tatsächlich eine inhaltliche Ähnlichkeit vorliegt.

Die von den Nutzern an dieser Stelle gemachten Angaben beziehen sich also zunächst nicht auf die Schlagwörter eines Artikels, sondern auf die Verknüpfung des Artikels selbst. Hier gibt es nun zwei Möglichkeiten. Zunächst wäre es sehr einfach möglich, den errechneten Ähnlichkeitswert (vgl. Tabelle 3.6) auf Basis der Für- und Gegenstimmen zu gewichten. Der Nachteil dieses Verfahrens

ist aber, dass sich eine solche Gewichtung nur auf *eine* Verknüpfung beziehen würde. Stattdessen kann man aber auch - und das wäre die zweite Möglichkeit - von folgenden Aussagen ausgehen, was in Nuntio der Fall ist.

Angenommen ein Nutzer gibt an, dass zwei Artikel zueinander passen. Dann kann man annehmen, dass die Wörter und Phrasen, die in beiden Artikeln vorkommen, die also letztlich dafür gesorgt haben, dass die beiden Artikel überhaupt miteinander verknüpft wurden, wichtiger sind als die Wörter, die nicht in der Schnittmenge vorhanden sind. Bei einer negativen Bewertung einer Verknüpfung kann man ebenso annehmen, dass die Wörter der Schnittmenge weniger relevant sind, da es auf deren Basis ja zu einer falschen Verknüpfung gekommen ist.



(a) Drei paarweise zusammenpassende Artikel

(b) Zwei inhaltlich unterschiedliche Artikel

Abbildung 3.7: Schnittmengen-Ansatz bei der Bewertung von Verknüpfungen

In Abbildung 3.7 ist das Verfahren dargestellt. In (a) sollen dabei drei inhaltlich zusammenpassende Artikel miteinander verglichen werden. Für die obigen Beispielartikel würde dabei in der Schnittmenge aller drei Artikel zumindest das Wort "google" und die Phrase "street view" enthalten sein. Werden hinreichend viele Verknüpfungen bewertet, wird sich vermutlich herausstellen, dass diese beiden Begriffe die wichtigsten des Artikels sind. Andere Begriffe werden aber ebenfalls in Schnittmengen landen, so kann es sein, dass der Begriff "internet" ebenfalls für wichtig gehalten wird oder das Wort "oberstaufen". Weniger häufig in der Schnittmenge werden aber wenig relevante Begriffe wie "pforten" oder "öffnet" landen. Genauso geht es andersherum, wie in (b) dargestellt. Wird also beispielsweise ein Artikel zu Google Street View versehentlich mit einem Artikel zu Googles Android-Betriebssystem verknüpft, landen genau die Begriffe in der Schnittmenge, die vermutlich weniger relevant sind. In obigem Beispiel würde hier auch der Begriff "google" in der Schnittmenge vorkommen und tatsächlich könnte man auch behaupten, dass der Begriff "google" alleine nicht aussagekräftig genug ist, um als sehr relevant zu gelten. Praktisch könnte es mit obigem Ansatz so sein, dass "google" sehr viele Für- aber auch viele Gegenstimmen erhält, was mit der im vorherigen Abschnitt beschriebenen Bewertungsfunktion zu einem neutralen Gewicht führen würde.

Auch wenn der Ansatz ein größeres Fehlerpotential bietet, ergibt sich hierbei ein enormer Mehrwert. Bewertet also ein Nutzer die Verbindung zweier Artikel positiv, wird - wie oben beschrieben - die Schnittmenge der Wörter und Phrasen gebildet, wobei für jedes Element der Schnittmenge das `users_for`-Attribut um eins erhöht wird. Andersherum eben genauso mit dem `users_against`-Attribut. Da sich diese Attribute aber auf die Verknüpfungen des betrachteten Artikels mit den Wörtern bezieht, wird die Nutzerbewertung nun bei der nächsten Iteration der Ähnlichkeitsanalyse des Artikels auf alle Verknüpfungen bezogen. Die Nutzeraussage wurde also verallgemeinert. Außerdem ist es so, dass man bei der Bewertung von Verknüpfungen ein zweiseitiges Verhältnis betrachtet. Es können also nicht nur die Attributwerte der Wortverknüpfungen des betrachteten Artikels angepasst werden, sondern auch die des verknüpften, was im Optimalfall für eine Verbesserung der Analyseergebnisse beider Artikel sorgt.

Schließlich kann man noch einen Schritt weiter gehen und behaupten, dass Begriffe, die in Artikel A von den Nutzern bereits als wichtig markiert wurden, vermutlich in Artikel B auch wichtig sind - obwohl sie im dortigen Text vielleicht gar nicht vorkommen. Angenommen viele Nutzer sind der Meinung, dass der Begriff “startet” in einem der Beispiel-Artikel zu Google Street View wichtig ist, da es in dem Artikel nicht allgemein um Google Street View geht, sondern speziell um dessen Start. Ein Mensch würde jetzt vermutlich zwischen den Begriffen “startet” und “öffnet seine Pforten” eine Verbindung herstellen, was bei der derzeitigen Implementierung von Nuntio aber nicht der Fall ist. Um dieses Synonym-Problem zu umgehen oder zumindest etwas zu verringern, wird nun so vorgegangen: Bestätigt ein Nutzer die Ähnlichkeit solcher zwei Artikel A und B und in Artikel A ist “startet” (durch die Nutzerbewertung) als relevant markiert, in B kommt das Wort aber gar nicht vor, so wird der Artikel B ebenfalls mit dem Begriff “startet” verknüpft. Es wird also für die weitere Analyse von ausgegangen, dass der Begriff “startet” ebenfalls in Artikel B vorkommt. Auch hier kommt es natürlich unter Umständen zu Fehlern, aber auch dieses Verfahren hat während des Testbetriebs gute Ergebnisse geliefert.

3.3.3 Überlegungen zu weitergehender Einflussnahme

Über die in den beiden vorherigen Abschnitten vorgestellten Mechanismen ist es Nutzern nun möglich, auf die Ergebnisse der Ähnlichkeitsanalyse Einfluss zu nehmen, so dass an dieser Stelle der Nutzen der Einbettung des Systems in einen sozialen Kontext bereits geklärt sein sollte. Bei der Beschreibung der Bewertung von einzelnen Wörtern in Abschnitt 3.3.1 ist dabei aber vielleicht schon aufgefallen, dass hier tatsächlich nur Wörter betrachtet wurden. Natürlich ist der Ansatz problemlos auch auf die Bewertung von Phrasen anwendbar, allerdings ist es wichtig die Nutzer nicht zu überfordern. In einem längeren Artikel können dabei durchaus 20 bis 30 Schlagwörter auftauchen, die einen Nutzer unter Umständen bereits davon abhalten, überhaupt ein Schlagwort zu bewerten. Würden an gleicher Stelle nun noch alle Phrasen auftauchen, wären sicher zu viele Informationen vorhanden (vgl. dazu auch Abschnitt 3.5.2 zum User-Interface). Bei der Bewertung von Artikelverknüpfungen ist es hingegen so, dass Phrasen problemlos in den Schnittmengen betrachtet werden können (vgl. auch Abb. 3.7 (a)), da der Nutzer selbst davon ja gar nichts merkt.

Stellt man also weitere Überlegungen zur Einflussnahme von Nutzern an, muss dabei beachtet werden, dass die Nutzer durch die Möglichkeiten nicht überfordert werden. Trotzdem soll an dieser Stelle zumindest eine weitere Möglichkeit besprochen werden, die in Nuntio nicht umgesetzt, aber bereits erwähnt wurde: Das Hinzufügen von nutzerspezifischen Tags. Mit anderen Worten also, es den Nutzern zu ermöglichen, selbst definierte Schlagworte für einen Artikel zu vergeben - das eigentliche "collaborative tagging" [GH06]. Dies wurde aus zwei Gründen nicht in Nuntios derzeitige Implementierung aufgenommen. Zunächst kann hier wieder das Problem der Texteingabe angeführt werden, die auf den Zielplattformen am besten minimiert werden sollte. Das viel gewichtigere Problem ist aber Nuntios Architektur selbst. Würde man also zulassen, dass ein Nutzer einen Artikel mit einem Schlagwort versieht, müsste dies konsequenter Weise auch in das Wörterbuch der relevanten Begriffe aufgenommen werden, damit weitere Artikel automatisiert damit verknüpft werden können - was zu weiteren Problemen führen würde. Man stelle sich nur vor, jemand würde das Schlagwort "the" vergeben. Würde man auf das Einfügen in das Wörterbuch verzichten, müssten die Nutzer extrem viele Artikel mit Schlagworten versehen, damit überhaupt Verknüpfungen zwischen Artikeln auf dieser Basis zustande kommen können. Während des Testbetriebs waren dafür jedoch deutlich zu wenig Nutzer vorhanden, so dass eine solche Option gegebenenfalls für einen späteren, größer angelegten Betrieb überdacht werden könnte.

3.3.4 Überlegungen zur Verbesserung des Wörterbuchs

Obwohl der Einfluss der Nutzer die Ergebnisse der Artikelverknüpfungen bereits signifikant verbessern kann, fehlt hier noch ein Schritt, um einen globalen Lerneffekt für das System aus den einzelnen Bewertungen zu gewinnen. Ein Ansatz hierzu kann dabei einfach mit den vorherigen Überlegungen formuliert werden. Betrachtet man ein als relevant geltendes Wort, kann über die Einzelbewertungen der Wortverknüpfungen dieses Begriffs mit *allen* Artikeln, eine Annahme über die globale Relevanz eines Wortes gemacht werden.

Wort	**Fürstimmen**	**Gegenstimmen**
apple	49	36
microsoft	47	54
macbook	21	2
tot	26	0
or	22	210
one	14	172
not	24	134

Tabelle 3.10: Aggregierte Für- und Gegenstimmen ausgewählter Begriffe

In Tabelle 3.10 wurden dafür während des Testbetriebs von Nuntio die Einzelbewertungen aggregiert. Dabei sind einige interessante Effekte zu beobachten. Zunächst erkennt man, dass die eigentlich durchaus als relevant zu betrachtenden Begriffe "apple" und "microsoft" ein eher aus-

gewogenen Verhältnis von Für- und Gegenstimmen haben. Dies ist wohl vor allem durch die Bewertung über die Schnittmengen zu erklären, da diese Begriffe zwar relevant sind, aber für sich genommen zu allgemein, um aussagekräftig zu sein. Genauso können aber Begriffe wie "macbook" oder "tot" gefunden werden, die ein Themengebiet stärker einschränken und deswegen vermutlich mit einem deutlichen Übergewicht an positiven Bewertungen versehen sind. Vielleicht noch erfreulicher ist aber, dass es eine ganze Reihe von Begriffe gibt, bei denen die negativen Bewertungen mehr als deutlich überwiegen. Dabei handelt es sich ausnahmslos um Begriffe, die wie "or", "one" oder "not" quasi keinen Informationswert für die inhaltliche Beschreibung eines Artikels haben.

Betrachtet man die obigen Zusammenhänge, lässt sich also die Aussage vertreten, dass sich an den aggregierten Für- und Gegenstimmen ablesen lässt, wie groß der Informationswert eines Wortes oder einer Phrase ist. Man könnte also nach Überschreitung eines Schwellwertes behaupten, dass ein Wort wie "or" in keinem Artikel als relevant gelten sollte und deshalb - unabhängig von der Position nach Worthäufigkeit (vgl. Abschnitt 3.2.4) - nicht im Wörterbuch der relevanten Begriffe berücksichtigt sein sollte. Weiterhin könnte man die Begriffe mit vielen positiven Bewertungen vor dem Löschen aus dem Wörterbuch schützen, obwohl sie vielleicht irgendwann zu häufig auftreten würden. Klar sollte aber sein, dass ein solches Vorgehen das System als Ganzes beeinflusst und entsprechende Schwellwerte entsprechend hoch liegen sollten. In Nuntio ist ein solches Verhalten bisher nicht berücksichtigt, da - wie die obigen Zahlen bereits zeigen - schlicht zu wenig Nutzerbewertungen vorliegen, um sinnvolle Schwellwerte erreichen oder wenigstens formulieren zu können.

Trotzdem sollte in diesem Abschnitt darstellt werden, wie Nuntio bereits in seiner jetzigen Implementierung aus dem Feedback der Nutzer lernt und die Analyseergebnisse verbessert werden können. Insbesondere durch den in diesem Abschnitt darstellten Ansatz lässt sich dies auf Basis der vorangegangenen Überlegungen und gesammelten Daten bei entsprechenden Nutzerzahlen noch weiter verbessern.

3.4 Interessensprofile

Neben den bisher beschriebenen Funktionen von Nuntio wurde in der Einleitung weiterhin die Möglichkeit zum "Markieren interessanter Artikel" gefordert. In Nuntio soll also nicht nur durch die persönliche Auswahl von abonnierten Newsfeeds personalisiert werden, sondern auch über Angaben der Nutzer zu den gelesenen Nachrichten. Wünschenswert ist es dabei, wenn sich die einzelnen Artikel so sortieren ließen, dass die - für den aktuellen Nutzer - interessantesten ganz oben erscheinen, während als uninteressant bewertete Artikel weiter nach unten verschoben werden. Es soll also ein Interessensprofil eines Nutzers angelegt werden, auf dessen Basis die einzelnen Artikel bewertet werden können.

In manchen anderen Systemen ist eine ähnliche Funktionalität umgesetzt, wobei der Nutzer hier eine Reihe von Themen angeben kann, die ihn interessieren und das System entsprechend klassifizierte Daten hervorhebt. Ein ähnliches Verhalten wurde dabei für Nuntio umgesetzt, wobei das Interessensprofil hier anhand von Beispielen aufgebaut werden soll, so dass ein Nutzer erneut

nicht zur Eingabe von Text gezwungen ist. Durch die in den vorherigen Abschnitten bereits beschriebene Architektur und das Layout der Datenbank, ist es nun fast einfach, ein solches Interessensprofil aufzubauen. Das Problem vor dem man hier steht lässt sich dabei einfach so formulieren, dass nun nicht mehr zwei Artikel miteinander verknüpft werden müssen, sondern ein Nutzer mit einem Artikel. Da die Artikel aber bereits mit Schlüsselwörtern und Phrasen versehen sind, können analoge Methoden zu den bisher vorgestellten genutzt werden, um festzustellen, an welchen dieser Schlüsselwörter der Nutzer interessiert ist und welche ein Zeichen dafür sind, dass ein Artikel für ihn uninteressant ist.

Um das Interessensprofil über Beispiele aufzubauen, kann ein Nutzer zu jedem Artikel angeben, ob dieser ihn inhaltlich interessiert oder eben nicht. In einer separaten Tabelle der Datenbank wird dies vermerkt, wobei ein weiterer Prozess im Backend für jeden dieser Datenbank-Einträge nun alle relevanten Wörter und Phrasen sucht und, wieder auf Grundlage eines Für- und eines Gegenstimmen-Attributs, vermerkt, ob Interesse an diesem Begriff besteht oder nicht.

Wort	**Fürstimmen**	**Gegenstimmen**
apple	9	0
iphone	8	0
google	3	0
zusätzlich	3	0
fifa	0	3
tabellenplatz	0	3

Tabelle 3.11: Ausgewählte Begriffe aus dem Interessensprofil eines Nutzers

Obige Tabelle 3.11 zeigt dabei beispielhaft einige Begriffe aus dem Interessensprofil eines Nutzers während der Testphase. Auf Basis der Angaben könnte man die Vermutung aufstellen, der Nutzer sei besonders an computernahen Themen, die mit den Schlagwörtern “apple”, “iphone” oder “google” versehen sind, interessiert. Weniger scheinen den Nutzer fußballzentrische Themen mit den Schlagwörtern “fifa” oder “tabellenplatz” zu interessieren. Es fällt aber auch auf, dass Begriffe, die wegen ihres geringen oder nicht vorhandenen Informationsgehalt wie hier “zusätzlich” eigentlich nicht berücksichtigt werden sollten. Bei den vorliegenden Angaben müsste der Sortier-Algorithmus davon ausgehen, dass der Nutzer an einem Artikel mit dem Schlagwort “google” ebenso interessiert ist, wie an einem Artikel, in dem das Wort “zusätzlich” vorkommt, was sicher nicht der Realität entspricht. Der Ansatz aus dem vorherigen Abschnitt 3.3.4 könnte hier helfen. Würde dieser Begriff als global irrelevant identifiziert werden, könnte er auch im Interessensprofil unberücksichtigt bleiben. Grundsätzlich würde es an dieser Stelle aber auch helfen, wenn der Nutzer mehr Artikel bewerten würde - die Zahlen in obigem Beispiel sind ja relativ niedrig - da bei den irrelevanten Begriffen wiederum zu erwarten ist, dass sich in etwa ein Gleichgewicht zwischen positiven und negativen Bewertungen einstellt.

Um nun festzustellen, wie groß das Interesse eines Nutzers an einem Artikel ist, kann dies komplett datenbankseitig und damit entsprechend performant in einer einzigen Query abgefragt werden. Dazu wird analog zu Listing 3.7 für jedes Wort ein Wert, diesmal jedoch aus dem In-

tervall $[-1, 1]$ ermittelt, der über alle Wörter aufsummiert und ins Verhältnis zur Wortanzahl des Artikels gesetzt, den Grad des Interesses angibt. Begriffe, die bisher nicht im Interessensprofil eines Nutzers vorhanden sind, erhalten dabei den Wert 0, genauso wie Wörter, die mit ähnlich vielen Für- und Gegenstimmen vertreten sind. Auf Basis der errechneten Werte können die einzelnen Artikel nun "nach Interesse" des Nutzers sortiert werden. Die Berechnung ist dabei so performant, dass die Sortierung während des Testbetriebs *on demand*, also online, erfolgen konnte. Bei einer größeren Nutzerzahl müsste hier sicher ein geeigneter Caching-Mechanismus eingesetzt werden.

3.5 Webapplikations-Frontend

In den vorherigen Abschnitten dieses Kapitels sind nun die einzelnen Algorithmen und Analyse-Prozesse des Backends ausführlich vorgestellt worden. Dabei wurde den angekündigten Kernthemen des Abonnierens von Newsfeeds, Finden von ähnlichen Nachrichten, Markieren von interessanten Artikeln und durch die Nutzereinflussnahme teilweise auch der sozialen Interaktion der Nutzer Rechnung getragen. Um letzterem, aber insbesondere dem Wunsch nach einer geeigneten Präsentation der Daten gerecht werden zu können, wird in den folgenden Abschnitten das entwickelte Frontend, wie bereits angekündigt, auf Basis einer Rails-Webapplikation vorgestellt.

Dabei ist zu beachten, dass bereits in den vorherigen Abschnitten immer mit Daten gearbeitet wurde, die durch die einzelnen Nutzer in das System gebracht worden sind, wobei bisher darauf verzichtet wurde, jeweils das zugehörige User-Interface zu zeigen. Das hat vor allem den Grund, dass hierdurch betont werden sollte, dass die Algorithmen des Backends vollständig unabhängig vom Frontend implementiert sind, so dass es problemlos möglich wäre, die im Backend erzeugten Daten in einer nativen Desktop-Applikation oder gar einer Konsolen-Applikation zu nutzen. Wenn also in den folgenden bilderreichen Abschnitten die Analyse-Ergebnisse aus dem Backend quasi visualisiert werden, kann man sich wiederum die Architektur aus Abschnitt 3.1 (insbesondere Abbildung 3.1) in Erinnerung rufen. Dabei wird deutlich, dass die einzelnen Nutzer-Aktionen, mit der Master-Datenbank als Quasi-Schnittstelle, im Backend asynchron verarbeitet und die Ergebnisse der Aktionen durch die Replikations-Architektur später im Frontend dargestellt werden.

3.5.1 Tagesansicht

Die Tagesansicht oder *Today-View* ist die Hauptansicht von Nuntio. In Abbildung 3.8 ist diese für einen Nutzer des Systems ausschnitthaft dargestellt. Beim Layout der Webseite wurde dabei die Idee einer gedruckten Tageszeitung aufgegriffen. Außerdem wurde darauf geachtet, dass die Darstellung für die Betrachtung auf einem Tablet-Computer wie dem iPad geeignet ist.

In Abbildung 3.8 ist zu erkennen, dass die Ansicht in drei Abschnitte aufgeteilt ist: Es gibt einen Kopfbereich, wo zwischen den einzelnen Rubriken von Nuntio gewechselt werden kann ("Today", "Yesterday", ...) sowie einen Hauptteil, der aus einer Seitenleiste rechts und einer Artikel-Ansicht

Abbildung 3.8: Today-Ansicht (Ausschnitt)

links besteht. Das Navigations-Konzept sieht dabei vor, dass zunächst einer der großen Menüpunkte gewählt wird und die davon abhängigen Optionen jeweils in der Seitenleiste dargestellt werden. Auf der Hauptseite hat man beispielsweise die Möglichkeit, die Sortierung der Artikel auf der linken Seite zu verändern - mehr dazu in den folgenden Absätzen. Außerdem wird ein Überblick über die Aktionen der Freunde eines Nutzers gegeben, aufgeteilt in Kommentare zu Artikeln und Nutzeraktivitäten.

Kern dieser Ansicht ist aber die Darstellung der Artikel auf der linken Seite. In der *Today-View* werden dem Benutzer dabei alle Artikel, die in den letzten 24 Stunden erschienen sind, angezeigt. Die Ansicht unter dem Menüpunkt “Yesterday” würde dabei alle Nachrichten der letzten 48 Stunden ohne die letzten 24 Stunden anzeigen und der Menüpunkt “This Week”

eben alle Artikel der letzten 7 Tage. Wie man sieht, werden die Artikel ähnlich wie in einer gedruckten Tageszeitung in verschiedenen Formen dargestellt. Der erste Artikel wird als die wichtigste Schlagzeile am größten, mit einem Bild und einem Ausschnitt des Inhalts präsentiert. Darauf folgen vier weitere Artikel, ebenfalls mit Bild, aber bereits kleineren Schlagzeilen, wonach sich die Anzeige erneut teilt. Auf der linken Seite werden dann - sofern vorhanden - Artikel aufgelistet, für die in der Datenbank kein Inhalt vorliegt. Im dargestellten Beispiel ist dies beim Newsfeed von heise.de[4] der Fall. Der Betreiber liefert hier lediglich die Schlagzeile, was bei einer Darstellung wie bei den übrigen Artikeln zu "Löchern" in der Seitendarstellung führen würde, da kein Auszug angezeigt werden kann. Auf der rechten Seite wird eine passende Anzahl von weiteren Nachrichten mit vorhandenem Auszug dargestellt. Weiterhin ist zu sehen, dass jeder Artikel mit einer Quellenangabe versehen ist, einer Information über die Veröffentlichungszeit beziehungsweise das Alter der Nachricht und - sofern vorhanden - eine Anzahl an Kommentaren, die zu dem Artikel vorhanden sind. Schließlich fällt auf, dass zu jedem Artikel ein Plus- und ein Minus-Symbol angezeigt wird. Bei beiden handelt es sich dabei um Schaltflächen, die im Backend die Interessens-Bewertung auslösen (vgl. Abschnitt 3.4).

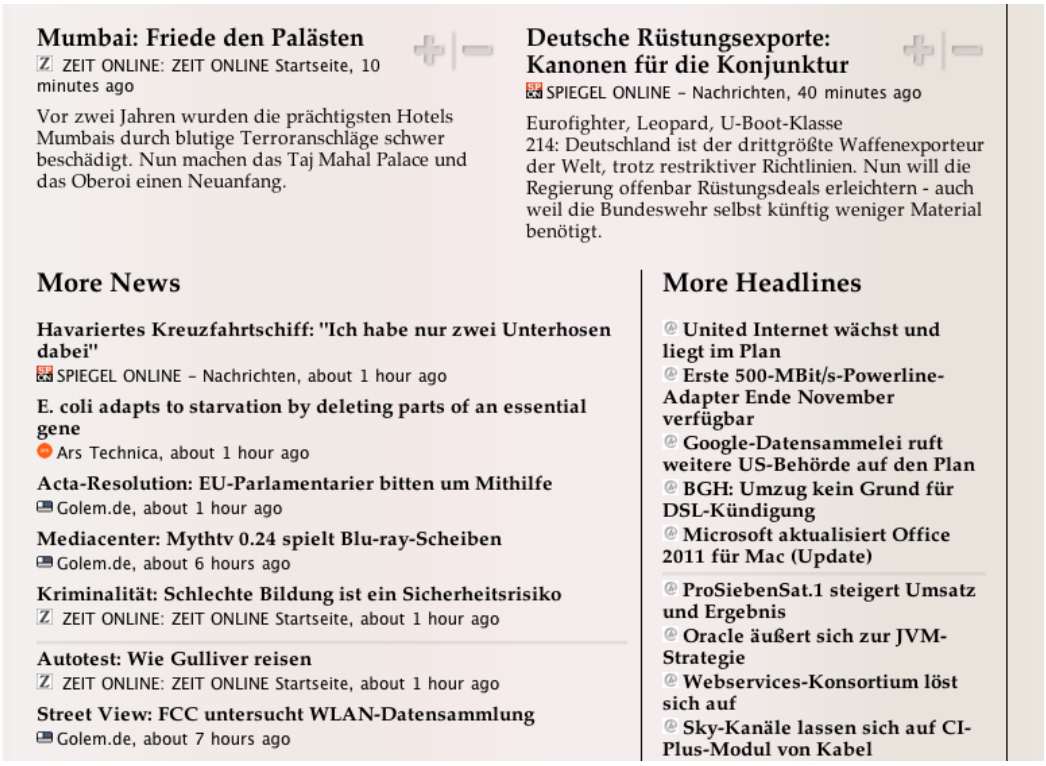


Abbildung 3.9: Unterer Teil der Today-Ansicht (Ausschnitt)

In Abbildung 3.9 ist ein Ausschnitt des weiter unten befindlichen Teils der Seite zu sehen. Insgesamt ist die Seite dabei so aufgebaut, dass auf die erste Schlagzeile vier Artikel in einem zweispaltigen Layout mit Bild folgen, darauf folgen auf der linken Seite maximal die 15 - je nach

[4] http://www.heise.de - Abruf 05.11.2010

Sortierung - obersten “Headlines”, sowie in der rechten Spalte daneben eine entsprechende Anzahl von weiteren Artikeln - im Normalfall maximal drei. Danach folgen acht Artikel im Layout der vier zweispaltigen, worauf wiederum acht Artikel in diesem Layout - jedoch ohne Bilddarstellung - dargestellt werden. Insgesamt sind so 24 Artikel mit Angabe eines textuellen Auszugs auf der Seite dargestellt sowie die obersten 15 Artikel, die keinen Auszug zur Verfügung stellen. Alle weiteren Artikel und “Headlines” sind wie in Abbildung 3.9 mit kleineren Überschriften und ohne Angabe eines Auszuges listenförmig dargestellt. Für die bessere Übersicht wird hier nach fünf Artikeln jeweils ein Trennstrich eingefügt.

Ziele dieser Hauptansicht von Nuntio waren also Übersichtlichkeit, sowohl über die Nachrichten als auch die Aktivität im sozialen Umfeld des Nutzers, Lesbarkeit bei der Anzeige auf einem Tablet-Computer und schließlich auch Bedienbarkeit, also ausreichend große Schaltflächen und Links, damit diese problemlos mit einer Fingersteuerung genutzt werden können. Besonders letzteres ist für eine Ausrichtung auf mobile Endgeräte mit Touch-Screen nicht zu unterschätzen.

Die Today-View ist auch ein gutes Beispiel für die Nutzung des in Rails beinhalteten Template-Systems. Es fällt nämlich auf, dass zumindest die vier dargestellten Nachrichten unterhalb der größten Schlagzeile im Wesentlichen gleich aufgebaut sind. Um im Template, mit dem diese Seite erzeugt wird, nicht viermal das gleiche HTML-Markup nutzen zu müssen, werden hier so genannte Partials, quasi Sub-Templates, genutzt.

```
1  <div class="half-size-item-<%= left_or_right %>">

3    <%= render :partial => 'helper/plus_minus_buttons', :locals => {:item => item} %>

5    <h4><a href="#" onclick="showNewsItem(<%=item.id%>); return false;">
6        <%= item.title %></a></h4>

8    <div class="subtitle"><%= item.favicon_tag %><%= item.feed.title %>,
9      <%= time_ago_in_words(item.sort_date) %> ago
10     <% if item.count_friends_comments(user) > 0 %>,
11       <%= pluralize(item.count_friends_comments(user), 'comment')%> by friends
12     <% end %>
13   </div>

15   <p class="summary"><%= item.summary_clean %></p>
16 </div>
```

Listing 3.8: Template einer Artikelvorschau (gekürzt)

In Listing 3.8 ist ein solches Partial-Template dargestellt. Dabei ist zu sehen, wie beim Rendern des Templates auf die übergebenen Variablen-Werte - beispielsweise durch `<%= item.title %>` - zugegriffen werden kann. Sogar die Darstellung der Plus- und Minus-Schaltflächen wurde in ein weiteres Partial ausgelagert, da es nicht nur von dieser, sondern von nahezu allen Artikel-Vorschau-Varianten genutzt wird. Gerade im Bereich des Layouts einer Seite liegt der Vorteil auf der Hand: Sollen beispielsweise die Symbole von Plus und Minus geändert werden, etwa auf einen nach oben oder nach unten zeigenden Daumen, reicht es, dies an *einer* Stelle zu tun und alle anderen Ansichten werden ebenfalls aktualisiert.

Weiterhin ist in Abbildung 3.8 zu sehen, wie die verschiedenen Artikel der Newsfeeds eines Nutzers in dieser Ansicht aggregiert, also zusammengefasst werden. Zunächst ist dabei ersichtlich, dass der Nutzer der Beispieldarstellung mindestens vier verschiedene Newsfeed abonniert hat, die in der Today-View dargestellt werden. Auf der linken Seite kann ein Nutzer dabei wählen, wie die einzelnen Nachrichten sortiert werden sollen. Dazu stehen an dieser Stelle drei Optionen zu Verfügung: "Hybrid", "Time" und "Interest". Die am einfachsten zu erläuternde Variante ist dabei die Sortierung nach "Time", also Erscheinungsdatum, der einzelnen Nachrichten. Ganz oben erscheint hier die neuste, ganz unten die älteste. Dabei wird allerdings nicht nach Newsfeed gruppiert, so dass auf eine Nachricht aus Newsfeed A auch eine Nachricht aus Newsfeed B folgen kann, wenn dies das Ergebnis der Sortierung ist. Für eine Darstellung der Nachrichten, gruppiert oder gefiltert nach einzelnen Newsfeeds, steht die in Abschnitt 3.5.3 beschriebene Feeds-Ansicht zur Verfügung.

Eine weitere Möglichkeit die einzelnen Nachrichten zu sortieren, ist nach dem Interessensprofil eines Nutzers. In Abschnitt 3.4 wurde dazu erläutert, wie jedem Artikel ein Interessens-Rating auf Basis der Bewertungen eines Nutzers zugewiesen werden kann. Bei Auswahl dieses Menüpunktes wird dieser Wert also für jeden anzuzeigenden Artikel ermittelt und schließlich danach sortiert. Im Optimalfall steht hier also der für den Nutzer interessanteste Artikel ganz oben und der am wenigsten interessante ganz unten.

Die im obigen Beispiel gewählte Sortierung ist die "Hybrid"-Sortierung. Hierbei wird versucht, sowohl das Alter einer Nachricht als auch das Interesse eines Nutzer zu berücksichtigen. Außerdem wird bei dieser Methode ebenfalls berücksichtigt, dass Nachrichtenmagazine wie zeit.de[5] täglich Nachrichtenmengen im zweistelligen Bereich veröffentlichen, während in einem privaten Blog vielleicht nur eine Nachricht pro Woche erscheint. Damit diese "seltenen" Nachrichten nicht untergehen sollten sie weiter oben angezeigt werden, als die Nachrichten von Newsfeeds, die deutlich häufiger Nachrichten veröffentlichen.

Nachricht	**Zeitwert**	**Interessenswert**	**Seltenheitswert**	**Gesamt**
"FC Bayern gewinnt erneut"	4	-1	0	3
"Merkel gibt Erklärung ab"	3	1	0	4
"Neues iPhone von Apple"	2	2	0	4
"Ich habe einen neuen Hund!"	1	0	2,96	3,96

Tabelle 3.12: Beispiel der Hybrid-Sortierung

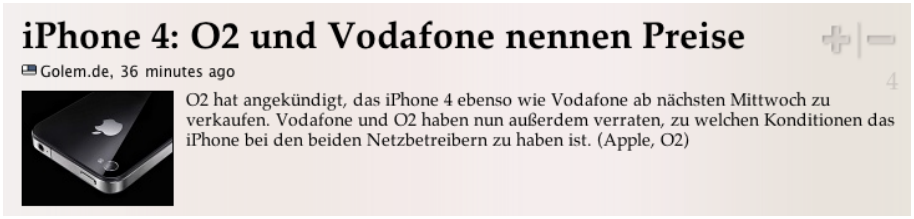
Das Vorgehen ist dabei in Tabelle 3.12 dargestellt. Problem dieser Methode ist es, ein möglichst ausgewogenes Verhältnis zwischen den einzelnen Werten zu schaffen. In Nuntio wird dabei zunächst die zeitliche Sortierung als Basis genommen. Die neuste der fiktiven Nachrichten ist also "FC Bayern gewinnt erneut". Jede Nachricht bekommt nun, ausgehend von der Länge der Liste, einen Grundwert unter Berücksichtigung des Alters des Artikels. Der zeitliche Abstand zwischen zwei Nachrichten soll dabei keine Rolle spielen. Danach werden die Nachrichten nach ihrem Interessens-Rating sortiert. Wenn k die Platzierung (der interessanteste Artikel hat die

[5] http://www.zeit.de - Abruf 05.11.2010

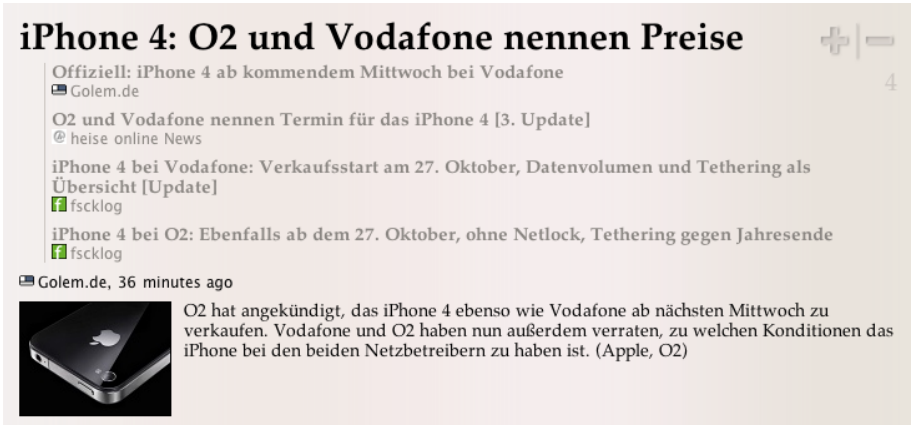
Platzierung 0) eines Artikels unter n Elementen ist und der Artikel als interessant betrachtet wird (das ermittelte Interessens-Rating also positiv ist), wird ihm der Wert $\frac{n}{2} - k$ als Interessenswert zugewiesen. Dahinter steht die Überlegung, dass der interessanteste Artikel mindestens in den oberen 50% der Artikel zu finden sein sollte, auch wenn er der älteste ist. Ist das Interessens-Rating negativ, wird dem Artikel der Wert $-1 \cdot (\frac{n}{4} - (n - (k + 1)))$ zugewiesen. Unter der Annahme, dass der gleiche Nutzer wie in Abschnitt 3.4 Basis für die Bewertung ist, könnten die Werte also wie oben angegeben aussehen. Der Fußball-Artikel erhält das schlechteste Rating, da er den Nutzer nicht interessiert, gefolgt von dem politischen Artikel und dem Artikel aus einem privaten Blog sowie schließlich dem iPhone-Artikel, der das höchste Rating erhält. Zum Schluss wird noch der Seltenheitswert eines Artikels berücksichtigt. Dazu wird berechnet, wie hoch die durchschnittliche Anzahl an veröffentlichten Nachrichten eines Newsfeeds pro Woche ist. Ist dieser Wert mehr als zehn Mal so niedrig wie der maximale Wert unter den anzeigten Newsfeeds, wird auf Basis dessen ein Wert ermittelt, der zwischen $\frac{n}{1{,}35} \cdot 0{,}1$ und $\frac{n}{1{,}35}$ liegt, je nach dem wie viel häufiger die übrigen Newsfeeds, im Vergleich zum betrachteten, Nachrichten veröffentlichen. In obigen Beispiel ist dabei vom Maximalwert ausgegangen. Die Summe aller Einzelwerte ergibt nun einen Gesamtwert nach dem wiederum sortiert werden kann, so dass sich letztlich ergibt, dass der politische oder der iPhone-spezifische Artikel ganz oben erscheint, danach der seltene Artikel aus dem privaten Blog und schließlich der für den Nutzer uninteressanteste, aber auch neuste Artikel. Auf diese Weise kann eine Sortierung erreicht werden, die deutlich dynamischer ist als die Sortierung nur nach dem Interesse eines Nutzers aber auch nicht so fluktuierend wie die Sortierung allein nach dem Erscheinungsdatum. Letztlich ist auch die Berücksichtigung der Seltenheit eines Artikels hier sicher ein Vorteil.

In der Today-View findet außerdem ein in Abbildung 3.8 nicht dargestellter Prozess zur Verbesserung der Ansicht statt. Dabei wird die Annahme gemacht, dass viele inhaltlich ähnliche oder sogar nahezu gleiche Artikel von verschiedenen Newsfeed-Betreibern veröffentlicht werden. In der Einleitung dieser Arbeit wurde dazu das Beispiel einer Bundestagswahl angeführt. An einem Wahlabend würde hierzu sicher jedes Nachrichtenmagazin mindestens eine - vermutlich aber deutlich mehr - Meldungen veröffentlichen. Die Today-View würde hier also von den Einzelmeldungen dominiert werden. Das ist in gewissem Maße sicherlich auch gerechtfertigt, allerdings wäre es auch wünschenswert, wenn beispielsweise die Artikel von Nachrichtenmagazin A, die nun wirklich nahezu die gleichen Meldungen wie die Artikel in Nachrichtenmagazin B enthalten, nicht doppelt in der Ansicht vorhanden wären, damit andere Meldungen in der Masse nicht untergehen. Da nach der Analyse im Backend für das System aber bekannt ist, welche Artikel zu anderen inhaltlich ähnlich sind, lässt sich dies für die Darstellung berücksichtigen.

In Abbildung 3.10 ist hierzu ein Beispiel dargestellt. Wie in (a) ersichtlich ist, erscheint auf der rechten Seite der Nachricht eine Nummer mit der Anzahl der ähnlichen Nachrichten, die eigentlich auch in der Tagesansicht angezeigt würden, aber eben eine so hohe Ähnlichkeit mit anderen Artikeln haben, dass sie zu einem Artikel gruppiert werden. Die Gruppierung wird unabhängig von der Sortierung als nachgestellter Verarbeitungsschritt durchgeführt. Dabei wird mit dem obersten Artikel beginnend festgestellt, welche Artikel zu diesem als ähnlich zu betrachten sind (also in der in Abschnitt 3.2.6 beschriebenen Ähnlichkeitsanalyse erkannt wurden) und ob einer dieser Artikel ebenfalls - weiter unten - angezeigt würde. Ist dies der Fall wird der Artikel aus der Liste der anzuzeigenden entfernt und eben dem weiter oben dargestellten Artikel zugewiesen.



(a) Vier ähnliche Nachrichten wurden gruppiert



(b) Detailanzeige der gruppierten Nachrichten

Abbildung 3.10: Gruppierung von inhaltlich gleichen Nachrichten

Im Fall von Abbildung 3.10 sind also vier Artikel gefunden worden, die als *so* ähnlich betrachtet werden, dass sie nicht separat angezeigt werden sollten. Zur Darstellung in (b) gelangt man nun, wenn der Nutzer die "4" auswählt, also anklickt oder mit dem Finger berührt. Das Erscheinen der Liste ist dabei zusätzlich mit der in Rails integrierten script.aculo.us-Bibliothek per JavaScript animiert, so dass die Liste der gruppierten Nachrichten quasi nach unten "ausgefahren" wird.

Zusammenfassend kann man also sagen, dass in der Today-View bereits alle Ergebnisse der Analyse-Algorithmen des Backends genutzt werden, um den Nutzern eine personalisierte, permanent aktuelle Tageszeitung zu liefern. Auf die Integration in das soziale Netzwerk und die weiteren Ansichten in Nuntio soll in den folgenden Abschnitten eingegangen werden.

3.5.2 Nachrichtenansicht

Klickt ein Nutzer nun eine beliebige Überschrift einer Nachricht an, erscheint die in Abbildung 3.11 dargestellte Nachrichtenansicht, die - wie man im Hintergrund erkennen kann - wiederum in einem papierähnlichen Layout über die darunter liegende Ansicht gelegt wird.

Auch die Detailansicht einer Nachricht ist zweigeteilt, so dass sich auf der linken Seite die Nachricht selbst findet und in der rechten Seitenleiste Informationen zu den Ergebnissen der Analyse dieser Nachricht betrachtet werden können. Auf der linken Seite findet sich nun erneut die Über-

Abbildung 3.11: Darstellung der Nachrichtenansicht (Ausschnitt)

schrift des Artikels, die Quelle sowie das Alter, worauf der eigentliche Inhalt der Nachricht folgt. An dieser Stelle wird dabei auch zugelassen, dass eigenes HTML-Markup des Newsfeed-Betreibers eingebunden wird. So können hier auch Tabellen oder Bilder in der Darstellung berücksichtigt werden. Möglicherweise schadhafter HTML-Code oder gar eigenes JavaScript wurde ja bereits bei der Integration in die Datenbank (vgl. Abschnitt 3.2.1) eliminiert. Schließlich folgt ein Link, um die Quelle des Artikels zu erreichen ("Read the full article at [...]"). Darunter sind nun die Schlagwörter visualisiert, mit denen der Artikel auf Basis der Analyseprozesse im Backend verknüpft wurde. Die Darstellung erfolgt in einer so genannten *Tag-Cloud*. Begriffe, die von mehr Nutzern als tatsächlich relevant markiert wurden, sind hier größer dargestellt (beispielsweise "google") als Begriffe, von denen Nutzer gesagt haben, dass sie weniger relevant sind (beispielsweise "hier" oder "fritz").

Darunter findet sich ein Abschnitt mit einer weiteren sozialen Funktion - einem Kommentarsystem. Viele Betreiber von Newsfeeds stellen auf ihren Webseiten zwar selbst eine Möglichkeit zur Verfügung, Artikel zu kommentieren, aber da die Seiten im Allgemeinen keine sozialen Netzwerke sind, bestehen auch keine Informationen über die Freundesbeziehungen von Nutzern. In Nuntio ist aber bekannt, welche Freunde ein Nutzer hat, so dass die Kommentare über einen Filter auf

diese beschränkt werden können. Es ist so einfach möglich, eine Diskussion über ein Thema innerhalb einer geschlossenen Nutzergruppe zu führen. Natürlich kann der Filter auch aufgehoben werden, so dass alle im System vorhandenen Kommentare aufgelistet werden. Außerdem ist es möglich, beim Senden eines Kommentars zusätzlich anzugeben, ob man den betreffenden Artikel gut oder schlecht findet.

An dieser Stelle sollte auch klar werden, warum Nutzern hier unter Tags nur die relevanten Wörter aber keine Phrasen angezeigt werden. Bei der hier beispielhaft dargestellten, sehr kurzen Nachricht wirkt das User-Interface bereits fast ein wenig überfrachtet, da die einzelnen Tags teilweise bereits sehr groß dargestellt werden - dazu kommen die Kommentare und die Seitenleiste, so dass die Nachrichtenmeldung hier höchstens ein Fünftel des genutzten Raumes einnimmt. Zudem ist es für Nutzer unter Umständen schwer nachvollziehbar, wenn in den Phrasen sowohl "google street" als auch "google street view" auftauchen würden. Genau das wäre beispielsweise für diese Nachrichtenmeldung der Fall.

In der rechten Seitenleiste sind nun einige Informationen zu der Nachrichtenmeldung dargestellt. Zunächst wird angezeigt, wie viele und welche Nutzer die Nachricht positiv bewertet haben, also an ihrem Thema interessiert waren. Dabei wird - hier wird wiederum die Integration in das soziale Netzwerk deutlich - zwischen Freunden und anderen Nutzern unterschieden. Ebenso erfolgt die Anzeige der Anzahl der Kommentare, worauf bereits die eigentlichen Ergebnisse der Analysealgorithmen folgen - die in Nuntios Datenbank gefundenen ähnlichen Nachrichten. Hierbei werden immer die besten zehn Ergebnisse aufgelistet, unabhängig davon, ob ein Nutzer den Newsfeed der Nachrichten-Quelle abonniert hat oder nicht. Bestenfalls kann dies den Nutzern nicht nur dabei helfen, ähnliche Nachrichten, die natürlich auch ähnlich interessant sein können, zu finden, sondern auch neue Newsfeeds zu entdecken.

Auffällig ist noch, dass jede der ähnlichen Nachrichten mit einem Haken sowie einem Verbots-Zeichen versehen ist. Da die Symbole für Plus und Minus ja bereits für die Bekundung von Interesse eines Nutzers vergeben sind, werden an dieser Stelle die beiden neuen Symbole genutzt, damit ein Nutzer angeben kann, ob der jeweilige auf der rechten Seite dargestellte Artikel zu dem auf der linken Seite betrachteten passt oder eben nicht.

```
1  function rateFeedEntryConnection(feed_entry_id, target_feed_entry_id, matches) {

3    new Ajax.Request('/ajax/set_matches_feed_entry', {
4      parameters: { feed_entry_id: feed_entry_id,
5                    target_feed_entry_id: target_feed_entry_id,
6                    matches: matches,
7                  }
8    });

10   $(feed_entry_id + '-' + target_feed_entry_id + '-match').setStyle({'opacity': '0.2'});

12 }
```

Listing 3.9: AJAX-Aufruf für die Bewertung der Ähnlichkeit

Sowohl beim Bewerten von Artikeln bezüglich des Interesses des Nutzers, als auch beim Bewerten der Ähnlichkeit wird dabei, wie in Listing 3.9 dargestellt, ein AJAX-Aufruf mit Hilfe des in Rails integrierten Prototype-Frameworks gestartet. Es ist klar, dass kaum ein Nutzer diese Funktion überhaupt verwenden würde, wenn bei jeder Bewertung die komplette Seite neu geladen würde und der Nutzer hierauf warten müsste. Stattdessen wird im Hintergrund ein passender Controller des Frontends aufgerufen, der die Aktion des Nutzers registriert und als Antwort nur Statusmeldungen wie "OK" verschickt. Im obigen Beispiel sieht man dabei, dass die Antwort letztlich überhaupt nicht berücksichtigt wird, damit der Nutzer möglichst nicht gestört wird. An dieser Stelle würde es auch wohl kaum zu einer angenehmen Nutzung beitragen, würde der Nutzer mit einer Fehlermeldung konfrontiert, sollte das Senden der Bewertung nicht geklappt haben. So oder so werden die Symbole nach dem Klick nur noch halbtransparent dargestellt (vgl. auch Abbildung 3.11), damit der Nutzer zumindest eine Rückmeldung zu seiner Aktion erhält.

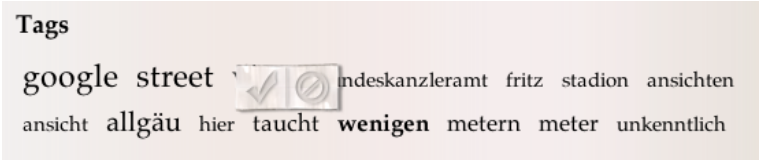


Abbildung 3.12: Bewertungsmöglichkeit von Tags

Mit dem gleichen Hintergedanken ist dabei die Möglichkeit der Bewertung der einzelnen Tags wie in Abbildung 3.12 dargestellt umgesetzt. Wählt ein Nutzer eines der Tags aus, erscheint sofort ein weiteres Menüelement, das es erlaubt mit der gleichen Symbolik zu bewerten, ob ein Tag für diese Nachricht tatsächlich relevant ist, oder eben nicht. Hier könnte ein Nutzer also zum Ausdruck bringen, dass der Begriff "wenigen" für den dargestellten Artikel irrelevant ist, was später im Backend verarbeitet und in der nächsten Iteration der Analyse der Artikel berücksichtigt wird.

In Abbildung 3.13 ist noch einmal das Kommentarsystem, das sich unter den Tags einer Nachricht befindet, dargestellt. Man sieht, dass Nutzer über ein Textfeld die Möglichkeit haben, den angezeigten Artikel zu kommentieren, wobei - wie oben bereits angesprochen - zusätzlich die Möglichkeit besteht einen Artikel als gut ("interesting") oder schlecht ("poor") zu bewerten. Um Diskussionen besser strukturieren zu können, kann dabei jeder Kommentar wiederum mit einer Menge von Antworten, "replies", versehen werden. Um das System nicht unnötig zu verkomplizieren, wurde hier auf eine Baumstruktur verzichtet, so dass eine Antwort nicht erneut mit eigenen Antworten versehen werden kann. Natürlich kann ein Kommentar aber mehrfach beantwortet werden. Um eine Antwort zu verfassen kann ein Nutzer die "replies" auswählen, woraufhin ein weiteres Textfeld und die bereits vorhandenen Antworten erscheinen.

Die Bewertung des Artikels durch einen Kommentar wird dabei von der Bewertung durch Auswahl des Plus- oder Minus-Symbols unterschieden. Hier besteht sowieso ein Problem mit der genutzten Symbolik, da eigentlich (mindestens) drei Ebenen der Bewertung eines Artikels unterschieden werden müssen. Angenommen es wird ein Artikel mit dem beispielhaften Inhalt "Selbstmordanschlag im Irak: Weit über 100 Tote." veröffentlicht. Ein Leser kann nun durch Wählen des Plus-Symbols angeben, dass er an dem Artikel beziehungsweise dem Thema *interessiert* ist. Problematisch ist dabei die positive Wirkung des Symbols, wobei das Plus hier vermutlich

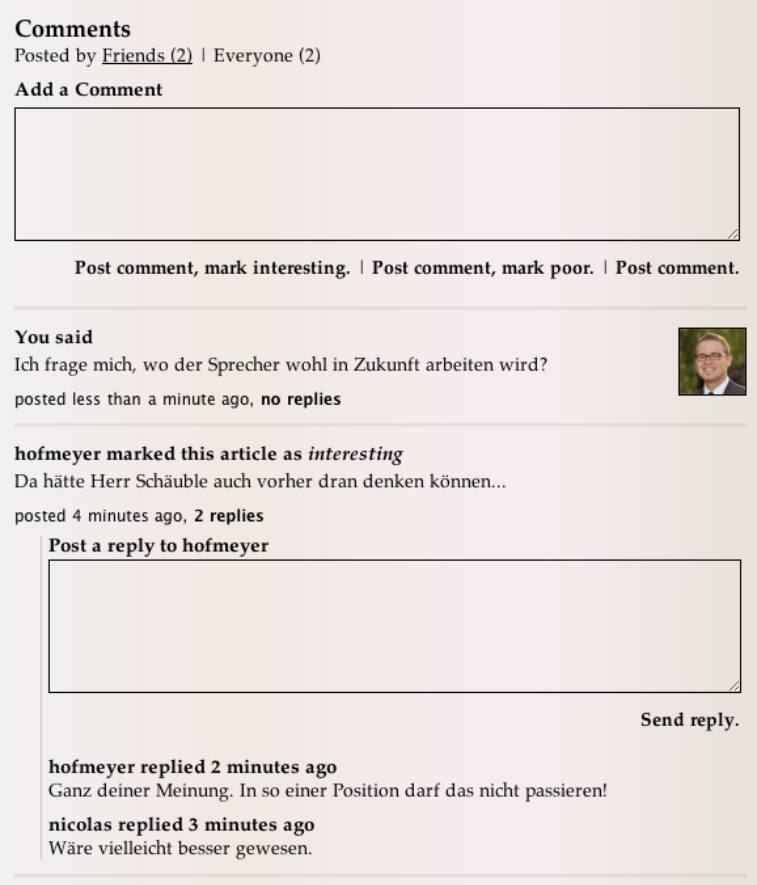


Abbildung 3.13: Kommentarsystem

noch relativ neutral wirkt. Bei der Bewertungsfunktionen anderer Plattformen, beispielsweise YouTube[6], wird hier ein Daumen-nach-oben beziehungsweise Daumen-nach-unten-Zeichen verwendet, wobei man als Nutzer damit vermutlich eher zum Ausdruck bringen möchte, dass der Inhalt *gefällt*. Bei einem Artikel wie dem obigen würde es den Nutzern also vermutlich schwer fallen, auf ein Daumen-nach-oben-Symbol zu klicken - selbst wenn sie wissen, dass damit ihr Interesse an dem Artikel, nicht aber ihr Gefallen an dem Inhalt zum Ausdruck gebracht werden soll. Die dritte Ebene der Bewertung einer Nachricht wäre die journalistische beziehungsweise inhaltliche Qualität. Zusammengefasst kann man also über einen Artikel wie den obigen sagen, dass er einen *interessiert*, dass die Aussage oder der Inhalt des Artikels *gefällt* und, dass der Artikel von einer hohen *Qualität*, also lesenswert, ist. In Nuntio wird der ersten Ebene des Interesses über die Plus- und Minus-Symbole Ausdruck verliehen, die zweite Ebene des Gefallens kann der Nutzer über einen Kommentar selbst artikulieren und die letzte Ebene der Qualität eines Artikels kann über die Bewertungsfunktion beim Senden eines Kommentars erfolgen.

[6] http://www.youtube.de - Abruf 05.11.2010

Auch die Funktionalität des Kommentar- oder Antworten-Sendens ist dabei mit AJAX umgesetzt - wie auch das Anzeigen der kompletten Nachrichtenansicht. Dies hat, wie oben bereits beschrieben, für den Nutzer den Vorteil, dass Aktionen schnell durchgeführt werden, ohne dass die komplette *Today-View* ständig erneut geladen werden muss. Auf Seite des Webservers ist dies besonders praktisch, da die Sortier-Funktionen, die im obigen Abschnitt beschrieben wurden, eine vergleichsweise große Server-Last erzeugen, die sonst bei jeder Nutzeraktion zu verarbeiten wäre.

Um die Nachrichtenansicht letztlich zu schließen, gibt es - wie in Abbildung 3.11 zu sehen - drei Möglichkeiten. Das "X"-Symbol auf der linken Seite blendet die Nachrichtensicht - erneut mit einem script.aculo.us-Effekt - einfach aus, während sich das Schließen mit den beiden Schaltflächen links oben auch direkt mit einer Interessensbewertung des Artikels verbinden lässt. Durch die Position am rechten Rand, können diese Schaltflächen bei Betrachtung von Nuntio auf dem iPad mit dem rechten Daumen erreicht werden. Unter der Annahme, dass die Mehrheit der Nutzer "Rechtshänder" sind (vgl. beispielsweise [GW92]), besteht hier die Hoffnung, dass so mehr Artikelbewertungen durchgeführt werden, was letztlich die Funktionalität von Nuntio verbessert.

3.5.3 Feeds-Ansicht

Abbildung 3.14: Feeds-View (Ausschnitt)

Während die Tagesansicht oder *Today-View* wie in Abschnitt 3.5.1 beschrieben, alle abonnierten Newsfeeds eines Nutzers in eine einheitliche Sicht integriert, ist es durchaus wünschenswert nur die Nachrichten eines bestimmten Newsfeeds anzeigen lassen zu können.

In Abbildung 3.14 ist dazu die Feeds-Ansicht in Nuntio dargestellt. In der rechten Seitenleiste finden sich dazu diesmal alle Newsfeeds, die der jeweilige Benutzer abonniert hat. Auf der linken Seite werden nach Auswahl dann nur die Nachrichten des gewählten Newsfeeds angezeigt. Außerdem ist es möglich die angezeigten Nachrichten nach ihrem Alter zu filtern ("Today", "Yesterday", "This Week") und die Sortierung zu beeinflussen ("Time", "Interest"). Da diese Funktionalität bereits für die Tagesansicht entwickelt wurde, ist der Aufbau der Feeds-Ansicht in seiner Implementierung relativ einfach. Auf eine Gruppierung von Artikeln wie in der *Today-View* wird hier verzichtet. Die Feeds-Ansicht kommt damit wohl der Darstellung eines klassischen News-Readers am nächsten, wobei auch hier die Analyse-Ergebnisse aus dem Backend - insbesondere bei der Sortierung nach dem Interesse des jeweiligen Nutzers - Anwendung finden. Es werden die obersten drei Nachrichten der Feeds-Ansicht am größten dargestellt, worauf zehn Nachrichten in etwas kleinerer Darstellung jedoch immer noch mit Bild und Textauszug folgen. Schließlich folgt eine Auflistung der übrigen Nachrichten analog zur Tagesansicht (vgl. Abb. 3.9). Eine Auswahl eines Artikels führt dabei zur gleichen Nachrichtenansicht, wie im vorherigen Abschnitt beschrieben.

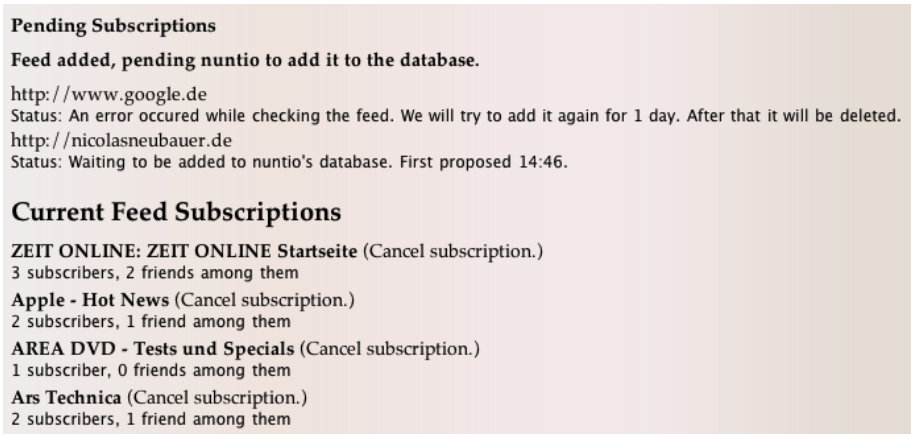


Abbildung 3.15: Feed-Management (Ausschnitt)

Mit den Links zum Hinzufügen oder Löschen eines Newsfeeds in der Seitenleiste, gelangt man in den Feed-Management-Bereich von Nuntio. In Abbildung 3.15 ist ein Ausschnitt daraus zu sehen, wobei erneut die Trennung zwischen Front- und Backend in Nuntio deutlich wird. Möchte ein Nutzer also einen neuen Newsfeed abonnieren, der noch nicht in Nuntios Datenbank vorhanden ist, wird dieser Wunsch (vgl. auch Abschnitt 3.2.1 zum Laden und Parsen von Newsfeeds) in die Datenbank eingetragen, damit - wiederum asynchron - ein Prozess des Backends in dessen nächster Iteration versucht den Newsfeed zu laden und für ein permanentes Update zu kennzeichnen. Man sieht in der obigen Abbildung, dass der Nutzer hier zwei neue Newsfeeds abonnieren wollte, wobei zu erkennen ist, dass versucht wurde "http://www.google.de" als Newsfeed anzugeben. Da Googles Startseite aber keinen Newsfeed zur Verfügung stellt, hat der Prozess des Backends hier einen Fehler festgestellt. Der Newsfeed wird nun als fehlerhaft markiert, was dem Nutzer im

Frontend mitgeteilt wird. Da es ja aber durchaus sein kann, dass der gewünschte Newsfeed nur temporär nicht erreichbar ist, wird in den nächsten 24 Stunden periodisch erneut versucht, aus der angegebenen Seite einen Newsfeed zu laden. Danach wird er - sollte dies nicht zum Erfolg geführt haben - aus dem System gelöscht.

Im unteren Bereich der Abbildung sind die Newsfeeds dargestellt, die bereits im System vorhanden und vom Benutzer abonniert sind, also Newsfeeds, die mindestens ein Mal erfolgreich geladen werden konnten. Auch diese Ansicht ist mit den Informationen aus dem sozialen Netzwerk angereichert, so dass außerdem ersichtlich ist, wie viele andere Nutzer und wie viele der Freunde des Nutzers einen Newsfeed abonniert haben.

3.5.4 Friends-Ansicht

Wenn in den vorherigen Abschnitten von der Nutzung sozialer Informationen die Rede war, dann bezog sich dies auf die Tatsache, dass Nuntio ein eigenes soziales Netzwerk darstellt.

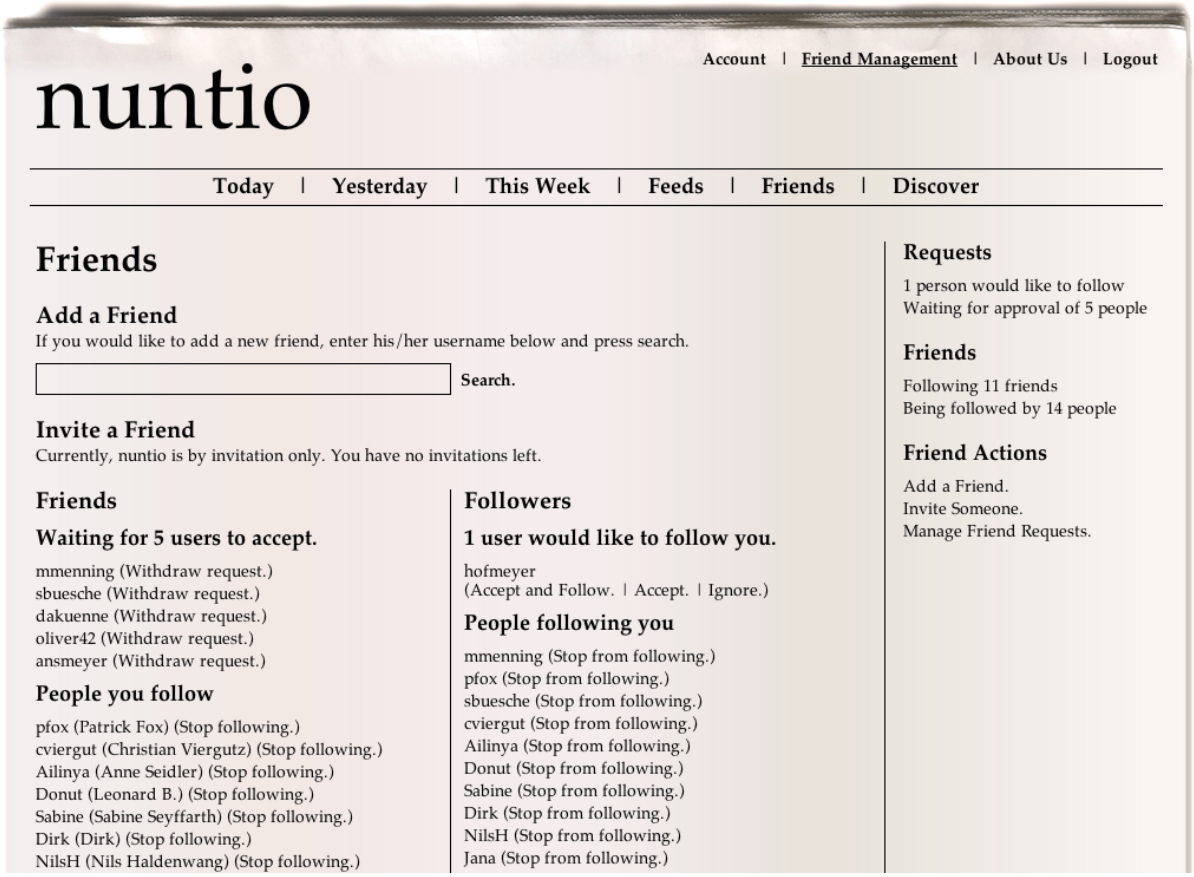


Abbildung 3.16: Friend-Management (Ausschnitt)

Um zu zeigen, dass Nuntio hier auch der in Abschnitt 2.5 vorgestellten Definition genügt, reicht es fast Abbildung 3.16 zu betrachten. Auf dieser Seite kann ein Nutzer seine Freundesliste anzeigen und verwalten. Dabei ist ein eigenes Authorisationskonzept umgesetzt, dass in etwa dem des sozialen Netzwerks Twitter[7] entspricht. Wie in der obigen Abbildung zu sehen ist, wird dabei unterschieden, ob die soziale Verbindung von dem aktuellen Nutzer ausgeht oder von der jeweils anderen Person. In Nuntio wird hierfür der Begriff der "Freundschaft" verwendet, so dass in der obigen Abbildung zwischen "Friends" und "Followers" unterschieden wird. Gebe ich als Nutzer also an, dass ich mit einem anderen Nutzer befreundet sein möchte, geht die soziale Verknüpfung von mir aus. Mit der dargestellten Suchfunktion kann ich den entsprechenden Nutzer finden und ihm eine Kontakt- oder Freundschafts-Anfrage senden. Der Nutzer bekommt daraufhin einen Hinweis, wie oben mit dem Benutzer "hofmeyer" dargestellt ist. Die Anfrage kann dann akzeptiert oder ignoriert werden, wobei dem ignorierten Nutzer dabei - ähnlich wie im sozialen Netzwerk Facebook - keine Information zukommt - die Anfrage wird lediglich nicht mehr angezeigt. Bestätigt der Nutzer die Anfrage wird die - immer noch gerichtete - Verknüpfung materialisiert, so dass ich als anfragender Nutzer nun einen neuen "Freund" habe, der in der Freundesliste angezeigt wird.

Abbildung 3.17: Darstellung des Profils eines Freundes (Ausschnitt)

Das Authorisationskonzept in Nuntio sieht dabei vor, dass für mich von Nutzern, mit denen ich nicht befreundet bin, keine Informationen außer dem Nutzernamen und den Daten aus der rechten Seitenleiste von Abbildung 3.17 sichtbar sind. Bin ich hingegen mit einem Nutzer befreundet, erfolgt die Darstellung des Profils wie in der obigen Abbildung, so dass nun auch der ganze Name (wie angegeben), die abonnierten Newsfeeds und die letzten Kommentare angezeigt werden.

Dabei ist zu beachten, dass die Verbindung immer noch nur von mir ausgeht. Das bedeutet, dass der Nutzer, der mir nun als "Friend" angezeigt wird, auf der anderen Seite lediglich weiß, dass ich ihm folge, ich also ein "Follower" bin. Damit diesem Nutzer auch die erweiterten Informationen zu mir angezeigt werden, muss er selbst eine Freundschaftsanfrage stellen. Es wird dann eine weitere Anfrage an mich gerichtet, die ich wiederum bestätigen muss, damit die Verbindung nun in beide Richtungen geht. Auf diese Weise ist sicher gestellt, dass kein Nutzer ohne meine Zustimmung einen erweiterten Zugriff auf mein Profil erlangt.

[7] http://www.twitter.com - Abruf 05.11.2010

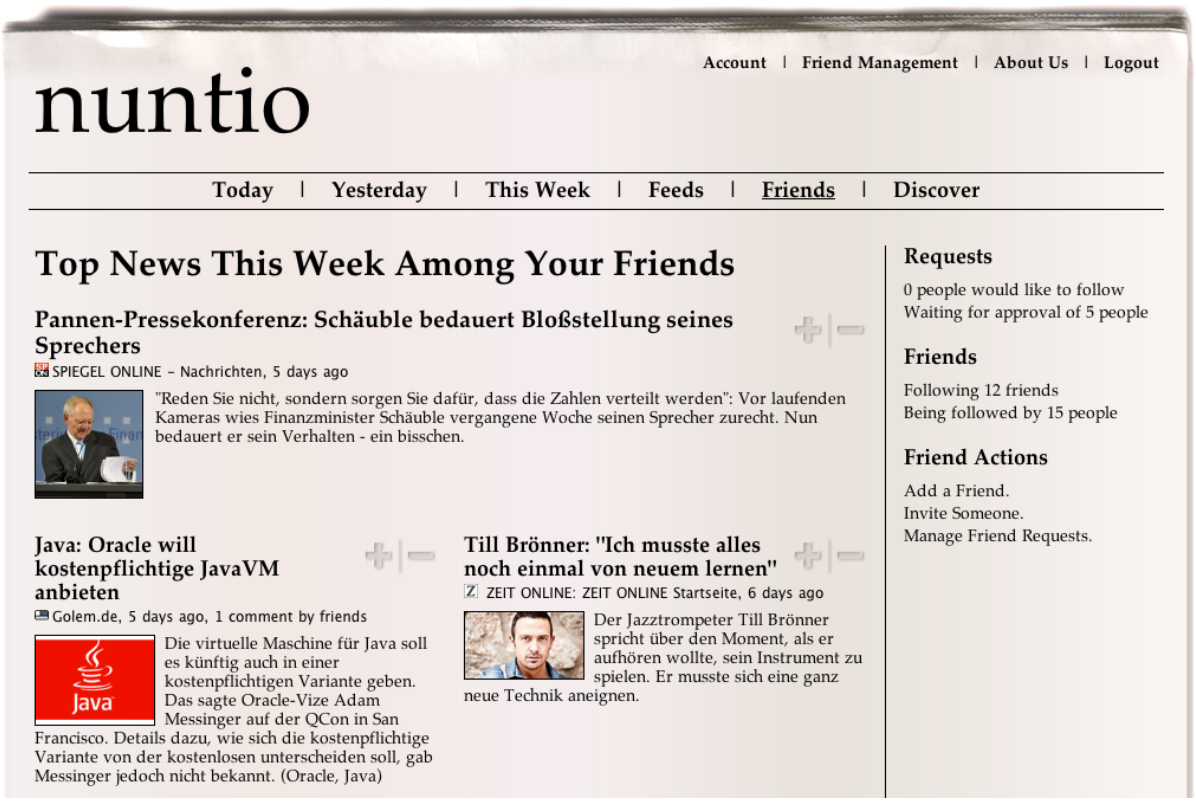


Abbildung 3.18: Darstellung der Freundes-Ansicht (Ausschnitt, oberer Teil)

Auf Basis der Freundesliste eines Nutzers werden nun nicht nur die sozialen Informationen in den anderen Ansichten, wie beispielsweise die Information, wie viele der Freunde eines Nutzers an einem Artikel interessiert sind, erzeugt. In Abbildung 3.18 ist dazu die Friends-Ansicht dargestellt, in der die Informationen aus dem sozialen Umfeld eines Nutzer aggregiert werden. Der Seitenaufbau ähnelt dabei auf den ersten Blick der Tagesansicht, da im oberen Bereich wiederum fünf Nachrichtenmeldungen im gleichen Layout angezeigt werden. Da ja nun bekannt ist, welche Freunde ein Nutzer hat und zudem bekannt ist, welche Artikel die Freunde dieses Nutzers interessieren, werden in der Friends-Ansicht immer die fünf "interessantesten" Nachrichten aus dem sozialen Umfeld eines Nutzers angezeigt. Auf diese Weise kann das "soziale" Erlebnis beim Konsum der Newsfeeds weiter gesteigert werden.

In Abbildung 3.19 ist der untere Teil der Friends-Ansicht zu sehen. Auf der linken Seite sind dabei erneut die Kommentare - wieder der letzten Woche - zeitlich sortiert aufgelistet. Natürlich geht es hier erneut um die Kommentare der Freunde eines Nutzers und da für die Darstellung das gleiche Template wie in den anderen Ansichten genutzt wurde, können auch hier per AJAX-Aufruf sofort Antworten auf Kommentare verfasst werden. Weiterhin ist auf der rechten Seite ein automatisch generierter "Activity-Stream" der Aktivitäten der Freunde eines Nutzers zu sehen. Bestimmte Aktionen innerhalb von Nuntio, beispielsweise das Hinzufügen eines Freundes, lösen dabei das automatische Publizieren einer Aktivität aus. Im dargestellten Beispiel ist etwa

Abbildung 3.19: Darstellung der Freundes-Ansicht (Ausschnitt, unterer Teil)

zu sehen, wie ein Nutzer einen neuen Newsfeed abonniert hat. Das Bewerten von Nachrichten in ihrer Ähnlichkeit, nach dem Interesse der Nutzer sowie die Bewertung von Tags löst dabei keine sofortige Veröffentlichung einer Aktivität aus, da hier sehr schnell sehr viele Informationen zusammen kommen können. Vielmehr aggregiert ein Prozess des Backends alle Aktivitäten eines Nutzers über einen bestimmten Zeitraum, hier beispielsweise 48 Stunden, und erzeugt daraus eine Aktivitätsmeldung, in der diese Informationen zusammengefasst werden.

3.5.5 Discover-Ansicht

Um interessante, neue Inhalte zu entdecken, ist es nicht unbedingt nötig, die Auswahl auf die Nachrichten zu beschränken, die die Freunde eines Nutzers interessant finden. Zwar könnte man behaupten, dass zwischen Freunden häufig auch Ähnlichkeiten im Interesse für verschiedene Themen vorherrschen, dies muss aber nicht heißen, dass relevante Inhalte nicht auch durch eine Betrachtung des “globalen” Geschmacks gefunden werden können.

In Abbildung 3.20 ist dazu die Discover-Ansicht in Nuntio dargestellt. Was in der Friends-Ansicht auf den Freundeskreis eines Nutzers beschränkt ist, wird hier über die komplette Datenbank von Nuntio betrachtet. Auf der linken Seite werden dabei die - derzeit 20 - Nachrichten aufgelistet, die in einem bestimmten Zeitraum - hier der letzten Woche - am häufigsten als interessant markiert wurden. Unter der Überschrift ist dabei zu erkennen, dass die Nachrichten sich nach der daraus folgenden Ordnung anzeigen lassen, aber auch sortiert nach dem Interesse des betrachtenden Nutzers. Dazu werden die Nachrichten einfach nach der Bewertung sortiert, die bereits in der Tages- und Feeds-Ansicht genutzt wurde.

Abbildung 3.20: Darstellung der Discover-Ansicht (Ausschnitt)

In der rechten Seitenleiste sind hierzu noch einige Zusatzinformationen dargestellt. Zunächst werden die Newsfeeds nach dem Interesse an den einzelnen Nachrichten dieser in eine Rangfolge gebracht. Darunter werden die am häufigsten genutzten Tags aufgelistet und schließlich ist zu sehen, auf welcher Zahlenbasis die Zusammenstellung überhaupt erfolgt ist.

Mit der Discover-Ansicht wurde damit in diesem Abschnitt die letzte der in Nuntio implementierten (für diese Arbeit relevanten) Sichten vorgestellt. Gemeinsam mit dem bereits zuvor beschriebenen Backend ist somit das implementierte System Nuntio in seiner Gesamtheit beschrieben worden.

Kapitel 4

Weiterführende Überlegungen, Ausblick und Fazit

Nach Betrachtung der technischen beziehungsweise technologischen Grundlagen und der Umsetzung der in der Einleitung vorgestellten Idee hinter Nuntio, soll dieses Kapitel abschließend Raum für einen Ausblick auf weiterführende Arbeiten und Überlegungen sowie ein letztes Fazit geben.

4.1 Ausblick

Während der Beschreibung der einzelnen Komponenten von Nuntio ist immer wieder angeklungen, dass es - insbesondere bei der Einbeziehung der Nutzer - schön gewesen wäre, mehr Datensätze zur Verfügung zu haben. Bei einem zeitlich so begrenzten Projekt wie einer Masterarbeit ist es andererseits auch schwierig, eine groß angelegte Testphase durchzuführen, wenn eine größere Menge von Funktionen implementiert werden soll. In der Testphase von Nuntio haben dabei insgesamt 21 Personen das System zumindest zeitweise getestet. Natürlich war es hier hilfreich zu sehen, wie sich die implementierten Algorithmen verhalten, wenn nicht nur die Testdaten desjenigen verarbeitet werden, der das System entwickelt hat. Andererseits ist die Anzahl der Nutzer deutlich zu klein um eine aussagekräftige Befragung durchführen zu können. Zwar haben die meisten der Tester bei informellen Befragungen angegeben, dass die Sortierung auf Basis der Angaben, die sie zu ihrem Interesse gemacht haben, nachvollziehbar funktioniert. Für weiterführende Arbeiten wäre es aber sicherlich reizvoll in einer kleineren Studie festzustellen, ob dies tatsächlich der Fall ist.

Insbesondere weil viele der implementierten Techniken auf einen experimentell ermittelten Schwellwert setzen, wäre es für weiterführende Arbeiten auch interessant, das System für eine deutlich größere Nutzerzahl freizugeben und dabei zu untersuchen, ob und wie die einzelnen Schwellwerte angepasst werden müssen. Ein gutes Beispiel hierzu sind die bereits in Abschnitt 3.3.4 angeführten Überlegungen zur Verbesserung des Wörterbuchs. Dort wurde beschrieben, wie es erreicht

werden könnte, auf Basis der Nutzermeinungen irrelevante Begriffe aus dem genutzten Wörterbuch zu streichen. Werden aber weniger Wörter verknüpft, ändert sich auch der errechnete Wert der Ähnlichkeitsanalyse. Zwar ist zu erwarten, dass der Wert dadurch "besser" - also differenzierter - wird, aber er wird - rein zahlmäßig - auch kleiner werden, so dass der Schwellwert hier gegebenenfalls angepasst werden müsste.

Eine deutliche Erhöhung der Nutzerzahl würde es auch ermöglichen, bessere Aussagen über die Skalierbarkeit der Systemarchitektur zu treffen und zu prüfen, ob sich die Annahmen über die Performance-Verbesserungen beispielsweise durch die Master-Slave-Replikation in diesem speziellen Anwendungsfall bestätigen.

Weiterhin wäre es reizvoll, die implementierten Algorithmen zum Clustering und der Suche nach ähnlichen Artikeln in größerem Rahmen auf vorklassifizierten Daten arbeiten zu lassen. Zwar wurden die Schwellwerte anhand von mehreren Testläufen verschiedener klassifizierter Datensätze ermittelt, da aber keine größeren Mengen vorklassifizierter Daten vorlagen, war es im gegebenen Zeitrahmen nicht möglich, eine entsprechende Datenbasis aufzubauen. Würde man die Algorithmen beispielsweise gegen mehrere tausend Nachrichten testen, wäre es sicher möglich, die Schwellwerte weiter zu optimieren. Für die Ähnlichkeitsanalyse wäre es sogar denkbar, einen weiteren Prozess des Backends dafür zu verwenden, den Schwellwert auf Basis von vorklassifizierten Daten dynamisch an das derzeit vorliegende Wörterbuch anzupassen.

Ein weiteres Problem, das noch weitgehend ungelöst geblieben ist, ist die Verknüpfung von Synonymen. Da aber bereits viele hilfreiche Informationen vorliegen - beispielsweise in welchen Artikeln ähnliche Themen behandelt werden -, könnte man sicherlich Ansätze entwickeln, um mit anderen Data-Mining-Techniken zu versuchen, ein Synonym-Wörterbuch aufzubauen.

Verbessert oder ausgeweitet werden könnte auch die Nutzung der sozialen Daten für die Präsentation von Inhalten. Beispielsweise wäre es mit nicht allzu großem Aufwand möglich, auf Basis der Daten für das Interesse zweier Nutzer ein Ähnlichkeitsmaß unter Nutzern zu definieren. Man könnte also versuchen herauszufinden, ob zwei Nutzer den gleichen "Geschmack" haben.

Auch für das Frontend wäre es wünschenswert durch Einsatz eines Usability-Tests festzustellen, ob die Darstellung insbesondere im Hinblick auf Geräte wie das iPad noch verbessert werden kann oder die gemachten Annahmen, beispielsweise zur Erhöhung der Anzahl von bewerteten Artikeln durch geschickte Positionierung der Bewertungsschaltflächen, überhaupt haltbar sind.

Neben sicher vielen weiteren Möglichkeiten von weiterführenden Arbeiten könnte man sich schließlich Gedanken darüber machen, wie das fertige System beziehungsweise die den Nutzer zu Verfügung gestellten Services außerhalb des akademischen Forschungsinteresses zu finanzieren wären. Durch die Tatsache, dass dem System bereits bekannt ist, was einen Nutzer interessiert oder eben nicht, drängt sich der Gedanke nach einer personalisierten Werbestrategie quasi auf. Für den Nutzer aus dem Beispiel in Abschnitt 3.4 wäre es sicher sinnvoller Werbung für iPhone-Zubehör zu schalten, als Karten für ein Fußballspiel anzubieten. Ist ein solches personalisiertes Werbe-System umgesetzt, wäre es wiederum interessant festzustellen, ob durch die Personalisierung die Akzeptanz von Werbung für die Plattform höher liegt als bei vergleichbaren Web-Angeboten. Außerdem könnte festgestellt werden, inwiefern sich Klickraten auf Werbung und dadurch resultierende Verkäufe verbessern beziehungsweise verändern.

4.2 Fazit

Blickt man an dieser Stelle auf die Einleitung dieser Arbeit zurück, so wurde dort die Entwicklung eines Nachrichten *aggregierenden*, *personalisierten* und in ein *soziales Netzwerk* integrierten Feed-Readers angekündigt. In den einzelnen Abschnitten wurden dabei sowohl die Grundlagen und Implementierungsdetails der hierzu entstandenen Webapplikation, als auch die verschiedenen Ansätze zur automatisierten Datenanalyse im Hintergrund ausführlich besprochen. Weiterhin wurde gezeigt, wie die Mitwirkung der Nutzer einer Plattform zu einer Verbesserung der Ergebnisse automatisierter Analyse-Prozesse in einem konkret umgesetzten System führen kann.

Zusammenfassend kann man also sagen, dass mit Nuntio ein in sich abgeschlossenes, funktionierendes System geschaffen wurde, dessen Daten nicht nur die Umsetzbarkeit, Effektivität und Effizienz der vorgestellten Konzepte zeigen, sondern auch Grundlage verschiedenster weiterführender Arbeiten im Bereich der Datenaggregation und -analyse sein können.

Literaturverzeichnis

[BE08] Boyd, Danah M. ; Ellison, Nicole B.: Social Network Sites: Definition, History, and Scholarship. In: *Journal of Computer-Mediated Communication* 13 (2008), Nr. 1, S. 210 – 230

[DHS01] Duda, Richard O. ; Hart, Peter E. ; Stock, David G.: *Pattern Classification.* New York : John Wiley Sons, 2001

[Fit07] Fitzgerald, Michael: *Ruby kurz und gut.* Köln : O'Reilly Verlag GmbH Co. KG, 2007

[Fla07] Flanagan, David: *JavaScript - Das umfassende Referenzwerk.* Köln : O'Reilly Verlag, 2007

[Gar05] Garrett, Jesse J.: *Ajax: A New Approach to Web Applications.* Webseite, Abruf 05.11.2010, Februar 2005. – `http://www.adaptivepath.com/ideas/essays/archives/000385.php`

[GH06] Golder, Scott A. ; Huberman, Bernardo A.: Usage patterns of collaborative tagging systems. In: *Journal of Information Science* 32 (2006), Nr. 2, S. 198–208

[Gru08] Gruber, Tom: Collective knowledge systems: Where the Social Web meets the Semantic Web. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 6 (2008), Nr. 1, S. 4 – 13

[GW92] Gilbert, Avery N. ; Wysocki, Charles J.: Hand preference and age in the United States. In: *Neuropsychologia* 30 (1992), Nr. 7, S. 601 – 608

[HK01] Han, Jiawei ; Kamber, Micheline: *Data Mining - Concepts and Techniques.* San Francisco : Morgan Kaufmann, 2001

[HMS01] Hand, David ; Mannila, Heikki ; Smyth, Padhraic: *Principles of Data Mining.* Cambridge : The MIT Press, 2001

[Inm96] Inmon, William H.: *Building the Data Warehouse.* New York : John Wiley & Sons, 1996

[Inm99] Inmon, William H.: *Building the Operational Data Store.* New York : John Wiley & Sons, 1999

[Joa02] Joachims, Thorsten: *Learning to Classify Text Using Support Vector Machines.* Bosten : Kluwer Academic Publishers, 2002

[KE01] Kemper, Alfons ; Eickler, Andé: *Datenbanksysteme - Eine Einführung.* München, Wien : Oldenbourg Verlag, 2001

[Mat02] Matsumoto, Yukihiro: *Ruby in a Nutshell.* Sebastopol : O'Reilly & Associates, Inc., 2002

[O'R05] O'Reilly, Tim: *What is Web 2.0.* Webseite, Abruf 05.11.2010, 2005. – `http://oreilly.com/pub/a/web2/archive/what-is-web-20.htm`

[Ora10] Oracle (Hrsg.): *MySQL 5.1 Referenzhandbuch.* Oracle, Oktober 2010

[Por97] Porter, M. F.: An algorithm for suffix stripping. In: *Readings in information retrieval.* San Francisco : Morgan Kaufmann Publishers Inc., 1997, S. 313–316

[Qua10] Quantcast Corporation: *Quantcast Mobile Web Trends, 2009 Report.* Webseite, Abruf 05.11.2010, Januar 2010. – `http://www.quantcast.com/docs/display/info/Mobile+Report`

[RTH09] Ruby, Sam ; Thomas, Dave ; Hansson, David H.: *Agile Web Development with Rails - Third Edition.* Raleigh, Dallas : The Pragmatic Bookshelf, 2009

[Smi10] Smith, Aaron: *Mobile Access 2010.* Webseite, Abruf 05.11.2010, Juli 2010. – `http://www.pewinternet.org/~/media//Files/Reports/2010/PIP_Mobile_Access_2010.pdf`

[TCH08] Tate, Bruce A. ; Carson, Lance ; Hibbs, Curt: *Ruby on Rails: Up and Running, Second Edition.* Sebastopol : O'Reilly Media, Inc., 2008

[Tho05] Thomas, Dave: *Programming Ruby.* Raleigh, Dallas : The Pragmatic Bookshelf, 2005

[ZC08] Zanardi, Valentina ; Capra, Licia: Social ranking: uncovering relevant content using tag-based recommender systems. In: *Proceedings of the 2008 ACM conference on Recommender systems.* New York, NY, USA : ACM, 2008 (RecSys '08), S. 51–58

Danksagung

Nicht nur für die Betreuung dieser Arbeit möchte ich Herrn Prof. Dr. Oliver Vornberger danken. Außerdem gilt mein Dank Friedhelm Hofmeyer für die Unterstützung bei der Konfiguration und Einrichtung der Hard- und Software, auf die Nuntio aufbaut.

Ich danke außerdem Michael Bäumer, Leonard Bulgrin, Patrick Fox, Nils Haldenwang, Julian Kniephoff, Daniel Künne, Mathias Menninghaus, Ansgar Meyer, Philipp Middendorf, Ilja Muhl, Kerstin Neubauer, Anne Seidler, Sabine Seyffarth, Dirk Stürzekarn und Christian Viergutz für die Teilnahme am Testbetrieb von Nuntio.

Ganz besonders möchte ich schließlich Mareike und Sebastian Büscher sowie Jana Lehnfeld für ihr ausführliches Feedback zu dieser Arbeit danken.

Erklärung

Ich versichere, dass ich die eingereichte Masterarbeit selbstständig und ohne unerlaubte Hilfe verfasst habe. Anderer als der von mir angegebenen Hilfsmittel und Schriften habe ich mich nicht bedient. Alle wörtlich oder sinngemäß den Schriften anderer Autoren entnommenen Stellen habe ich kenntlich gemacht.

Osnabrück, 15. November 2010