Diplomarbeit

Konzeption und Implementierung eines Generators
zur Erzeugung von Activity Streams für den unternehmensinternen Einsatz

Daniel Rentsch
rentsch.daniel@gmail.com
Matrikelnummer: 3224301

Technische Universität Dresden
Fakultät Informatik
Institut für Systemarchitektur

Verantwortlicher Hochschullehrer
Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill

Betreuer
Dr. Marius Feldmann; Dipl.-Medieninf. Philipp Katz

Bearbeitungszeitraum
01.06.2011 – 30.11.2011

Selbstständigkeitserklärung

Hiermit erkläre ich, Daniel Rentsch, dass die vorliegende Arbeit zum Thema „Konzeption und Implementierung eines Generators zur Erzeugung von Activity Streams für den unternehmensinternen Einsatz“ selbständig von mir, ohne die Hilfe Dritter und ausschließlich unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt wurde.

Dresden, 30. November 2011

Aufgabenstellung

Inhaltsverzeichnis

1 Einleitung

Innerhalb eines Unternehmens kommen heutzutage zahlreiche verschiedenartige Softwarewerkzeuge zum Einsatz, welche kontinuierlich neue Informationen erhalten bzw. generieren und diese zentralisiert zur Verfügung stellen. Insbesondere in der Domäne der Softwareentwicklung lässt sich feststellen, dass der Erstellungsprozess einer Anwendung ohne Werkzeugunterstützung praktisch nicht mehr durchführbar ist. Hierein fallen unter anderem Mailinglisten, Issue-Tracking-Systeme - wie beispielsweise sogenannte Bugtracker, Anwendungen zur Versionsverwaltung bzw. verschiedenartige Repositorien sowie Foren, für eine verbesserte und zentralisierte Kommunikation. Vielen Werkzeugen gemein ist die Fähigkeit, dass einzelne Anwender bzw. Nutzer sich über neue oder geänderte Daten mittels entsprechender Nachrichtenströme informieren können, welche durch die Werkzeuge zur Verfügung gestellt werden. Zusätzlich zu weiteren Zugriffsmöglichkeiten bieten diese Informationskanäle, auch *Feeds* oder *News Feeds* genannt, dem Nutzer eine komfortable Möglichkeit, sich über aktuelle Änderungen und Neuigkeiten der jeweiligen Systeme zu informieren - in Folge dessen tritt der automatisierte Versand von E-Mails (beispielsweise in Form von Statusnachrichten) zunehmend in den Hintergrund. Im besonderem Maße tritt die Nutzung von Feeds als Kommunikationsmittel im Umfeld von Open-Source-Projekten in der Vordergrund, da eine direkte Kommunikation im Vergleich zu einem betrieblichen Umfeld, aufgrund der hochgradig verteilten Systemumgebung, teilweise unmöglich ist [72]. Des Weiteren kommen zusätzlich zu den bereits etablierten Programmen immer stärker Technologien von sozialen Netzwerken auch in einem unternehmensinternen Umfeld zum Einsatz [52], welche weitere Informationskanäle generieren werden. Aufgrund der zunehmenden Menge von Datenquellen nimmt Quantität und Frequenz der emittierten Nachrichten bzw. Informationen weiter zu, was zur Folge hat, dass ein Mitarbeiter mehr und mehr Aufwand investieren muss, um die relevanten Informationen erkennen und analysieren zu können. Diese Problematik wird gemeinhin unter dem Begriff *Information Overload* zusammengefasst und in [22] wie folgt im Unternehmenskontext charakterisiert:

> *„...managers receive more information from more sources through more channels than almost anyone else in an organisation. Unfortunately, managers find themselves bombarded with information - too much, too fast, too late. Interestingly, even with an oversupply of information, managers believe that they do not get all the information they need to do their jobs. The dilemma is clear: on the one hand, managers receive too much information, while on the other hand, they don't get enough of the right information..“*

Im Gegensatz zum öffentlichen World Wide Web kann jedoch festgehalten werden, dass die Menge an irrelevanten, respektive unerwünschten, Informationen wesentlich geringer ist, da von einer kontrollierten Umgebung ausgegangen werden kann. In [54] wird somit die Frage nach einer ausreichenden Strukturierung der Daten aufgeworfen und als eigentliche Herausforderung klassifiziert:

> *„...A lack of structure, not the amount, is the reason for our growing inability to cope with information today. ...“*

1.1 Motivation

Wie bereits im vorangegangen Abschnitt geschildert, werden in vielen unternehmensinternen Workflows, insbesondere im Umfeld der Softwareentwicklung, sehr viele verschiedenartige Informationen emittiert. Aufgrund der Quantität ist es schwierig, bis teilweise sogar unmöglich, die relevanten Daten aus diesen heraus zu filtern bzw. händisch zu erfassen und zu klassifizieren, dies wird gemeinhin unter dem Begriff **Information Overload** beschrieben. Es existieren zahlreiche Ansätze [8, 43, 50, 76] welche das gemeinsame grundsätzliche Ziel haben, die Informationskanäle zu filtern, indem Nachrichten nach deren Relevanz sortiert bzw. entfernt werden. Diese Vorgehensweise ignoriert jedoch den von [54] geschilderten Sachverhalt, dass oft nicht die Quantität (zumindest im Unternehmenskontext), sondern die fehlende Strukturierung die eigentliche Problematik darstellt. Eine der Ursachen für diesen Phänomen ist die meist sehr heterogene Systemumgebung, welche dazu führt, dass einzelne semantisch in Zusammenhang stehende Teilinformationen auf verschiedene Quellen verteilt [13] sind und erst eine Aggregation und Analyse zu einer zufriedenstellenden Strukturierung führen würde (**Information Scattering**). Erschwerend kommt hinzu, dass die Daten der jeweiligen Quellen teilweise schwierig oder nur unter erheblichen Aufwand exportierbar sind, der Begriff der sogenannten *Information Silos* [87] beschreibt diesen Sachverhalt. Betrachtet man Workflows und die damit im Zusammenhang stehenden Informationskanäle im Kontext von Unternehmensanwendungen, so wird deutlich, dass die einzelnen Nachrichten nicht nur semantisch in Beziehung zueinander stehen, sondern oft ein eindeutiges Ursache-Wirkungs-Prinzip (Kausalität) zugrunde liegt [49]. Die Nachrichten sind somit kausal voneinander abhängig, was jedoch aufgrund der separierten Kanäle schwierig bzw. nur mit manuellen Analysen für den Nutzer ersichtlich wird. Ein System, welches die genannten Punkte berücksichtigen würde, hätte einen entscheidenden Vorteil für den Nutzer: eine Nachricht wird nicht mehr isoliert betrachtet, sondern in einem (sofern vorhanden) spezifischen Kontext. Dies wird umso wichtiger, je größer die zu betrachtenden Zeiträume sind, da mit zunehmenden Umfang eine händische Analyse der Kausalität immer zeitaufwändiger wird. Im Ergebnis erhält der Nutzer somit ein verbessertes Situationsbewusstsein (**Situation Awareness**) [20, 23] da im Idealfall jede emittierte Nachricht eindeutig einem spezifischen Kontext zugeordnet werden kann, welcher wesentlich mehr Informationen zur Verfügung stellt, als die Nachricht für sich genommen anbietet. Ist ein System in der Lage, die Kausalität sowie Strukturierung aller Nachrichten korrekt zu modellieren, so ergibt sich ein weiterer Vorteil: es entstehen **Hierarchien** von Nachrichten. Berücksichtigt man die Tatsache, dass praktisch allen Umgebungen Ontologien zugrunde liegen, so können einzelne, semantisch in Relation stehende, Nachrichten auf deren übergeordnete Entität reduziert werden. Dies erlaubt eine abstraktere Systemsicht und somit eine einfachere Analyse der Umgebung bzw. Nachrichten. Oftmals treten aufgrund der heterogenen Systemumgebungen und der Tatsache, dass Informationen händisch erzeugt werden (welche in diesem Sinne natürlichsprachigen Formulierungen entsprechen und in der Regel ohne feste Konventionen bzgl. Struktur vorliegen), die emittierten Nachrichten nicht in einer einheitlichen Syntax auf - vielmehr existiert ein Konglomerat an verschiedenartigen Formulierungen, wodurch es zur Aufgabe des Nutzers wird, diese auf gleichwertige Semantiken zu untersuchen bzw. zu erkennen. Da praktisch allen Bestandteilen eines Workflows reale Aktionen zugrunde liegen, welche durch Menschen (Akteure) durchführt wurden, bietet es sich an, eine **aktivitätsbezogene Sichtweise** innerhalb eines entsprechenden Systems abzubilden.

1.2 Zieldefinition

Im Rahmen dieser Arbeit soll ein System konzipiert und realisiert werden, welches in der Lage ist, heterogene Informationsquellen zu integrieren und analysieren, mit dem Ziel, Relationen zwischen semantisch zusammengehörigen Nachrichten detektieren zu können. Grundsätzliches Ziel ist hierbei, dass Detektion sowie Modellierung der Beziehungen weitestgehend automatisiert erfolgen kann. Als Basis bzw. Kriterium für die Erkennung von Relationen soll hierbei eine aktivitätsbezogene Sichtweise zur Anwendung kommen, das heißt, die einzelnen Nachrichten müssen einer Analyse unterzogen werden, welche als Ergebnis eine Abbildung auf deren zugrunde liegenden Aktivitäten ermöglicht. Grundlage hierfür soll die Activity-Stream-Spezifikation [5] darstellen, welche im einfachsten Fall einer Syntax der Form *Subjekt Prädikat Objekt* entspricht und somit dem Nutzer die Möglichkeit bietet, die emittierten Informationen unkompliziert erfassen zu können. Im Rahmen der Arbeit soll dabei untersucht werden, welche Erweiterungen notwendig und geeignet sind, um verschiedenartige Relationen zwischen Nachrichtenentitäten sowie deren Quellen modellieren und verarbeiten zu können. Aufbauend auf den detektierten Relationen, soll das System mittels zusätzlichen Metadaten aggregierte Hierarchien von Nachrichten generieren können, welche dem Nutzer eine reduzierbare Sicht auf die vorhandenen Informationen ermöglicht. Die Arbeit hat somit das Ziel, dem Nutzer ein verbessertes Situationsbewusstsein [23] zu ermöglichen, indem Nachrichten zu ihrem jeweiligen Kontext zugeordnet sowie in Relation zu anderen Entitäten gesetzt werden. Durch Bereitstellung von geeigneten Konfigurationsmechanismen soll das System zudem adaptierbar auf verschiedenartige Umgebungen sein und somit eine weitestgehende Domänenunabhängigkeit gewährleisten.

1.3 Anwendungsfälle

Anwendungsfall 1: Relationen *MyMail* ist ein mittelgroßes Open-Source-Projekt, welches sich mit der Entwicklung eines sogenannten Webmailers[1] beschäftigt. Das Projekt wird mittels eines zentralen Softwareentwicklungsportals[2] verwaltet. Dieses bietet einen Issue Tracker sowie eine Versionsverwaltung an. Für die Kommunikation unter den Entwicklung sowie den Nutzern verwendet MyMail historisch bedingt jedoch ein eigenes, extern administriertes, Forum anstelle des vom Projektportal bereitgestellten. Alle genannten Werkzeuge publizieren Veränderungen und Aktualisierungen mittels eines Web Feeds. Alice ist Projektmitglied und Entwicklerin bei MyMail, sie benötigt jede Woche mehrere Stunden, um innerhalb des Forums neue Themen zu betreuen, welche sich mit Fehlern in MyMail beschäftigen. Wird ein Thema von ihr als Bug eingestuft, was nur bei einer Minderheit der Themen der Fall ist, eröffnet sie im Bugtracker einen entsprechenden Eintrag und setzt händisch einen Link auf das Thema im Forum. Analog setzt sie auch im Forum einen Kommentar, mit einem Verweis auf den Bugtracker Eintrag – auf diese Weise wissen die Nutzer, dass das Thema nun offiziell bearbeitet wird. Nachdem der Fehler von Entwickler Steve behoben wurde, wird die entsprechende Datei im Versionsverwaltungssystem erneuert, mit einem Verweis im Log-Eintrag auf den entsprechenden Bug. Im Bugtracker setzt Alice anschließend den Fehler auf den Status „gelöst“. Damit auch Nutzer des Forums über den neuen Stand in Kenntnis gesetzt

1 Webmailer sind Dienste im World Wide Web, welche die Verwaltung von E-Mails mittels eines Webbrowsers ermöglichen.

2 beispielsweise http://sourceforge.net/

werden, schreibt Alice in diesem einen Verweis auf den geschlossenen Bugreport sowie auf das Release im Versionsverwaltungssystem. Am Ende jeder Woche analysiert Projektleiter James den Fortschritt von MyMail. Dabei untersucht er gemeldete Fehler, Feedback von Nutzern im Forum sowie Änderungen innerhalb des Versionsverwaltungssystems. Er entdeckt als erstes die Diskussion im Forum und untersucht die Beiträge. Am Ende des Themas stößt er auf den von Alice erstellen Bugreport bzw. dessen Link. Er folgt diesem und kann somit den Fehler im Detail analysieren. Nun erkennt er auch, dass der Bug bereits gelöst wurde und navigiert im Versionsverwaltungssystem auf die entsprechende Datei. James fragt sich nun, ob der Fehler auch wirklich geschlossen wurde, er begibt sich also wieder zum Bugtracker um dies zu überprüfen und kann dies bestätigen. Da für James die Kommunikation mit den Nutzern einen hohen Stellenwert hat, möchte er nun noch wissen, ob auch im Forum das entsprechende Thema geschlossen wurde bzw. die Anwender in Kenntnis über den neuen Stand gesetzt wurden. Er navigiert wieder zum Forum und sieht den Eintrag von Alice. Nun weiß er, dass der Workflow vollständig abgeschlossen ist und der Fehler eliminiert wurde. Der geschilderte Workflow kostet James sehr viel Zeit, er muss händisch verschiedenartige Informationsquellen durchsuchen und Beziehungen zwischen einzelnen Nachrichten erkennen: Wurde ein Bug zu einem Thema eröffnet? Welche Dateien waren dabei betroffen? Ist der Fehler behoben? Wissen dies auch die Anwender und Leser des Forums und vieles mehr.

Aus diesem Grund soll das zu konzipierende System eine Möglichkeit bieten, Relationen in Nachrichten automatisiert detektieren, verarbeiten und auf geeignete Weise präsentieren zu können.

Anwendungsfall 2: Kausalität Martin ist Qualitätsbeauftragter von MyMail. Wenn neue Features in das Programm integriert werden, muss er anschließend sorgfältig verschiedenartige Informationskanäle überwachen, indem er beispielsweise analysiert, ob überdurchschnittlich viele Bugs gemeldet wurden und inwiefern dies das neue Feature verursachte. Teilweise kommt es auch vor, dass er nicht über die Veröffentlichung des Features informiert wurde und er so erst im Nachhinein erkennt, dass die Erweiterung die Fehlermeldungen ausgelöste. Hilfreich wäre für ihn demnach ein Mechanismus, welcher automatisiert die Veröffentlichung von Erweiterungen erkennt und ihn benachrichtigt, wenn zeitnah überdurchschnittlich viele Fehlermeldungen erscheinen.

Das zu konzipierende System soll daher Möglichkeiten bieten, kausale Zusammenhänge zwischen Nachrichten modellieren und definieren zu können. Auf diese Weise können spezifische Ereignisse in Nachrichtenströmen detektiert werden.

Anwendungsfall 3: Activity Streams Innerhalb der Softwareentwicklungsabteilung des Unternehmens ACME werden Softwareprojekte durch eine Vielzahl von modernen Entwicklungswerkzeugen unterstützt. Die Dokumentationsabteilung ist unter anderem für die Erstellung des Handbuches, der Pflege des internen Wikis sowie der Erstellung einer sogenannten Online-Hilfe zuständig. Viele der genannten Werkzeuge bieten Informationsströme an, beispielsweise informiert das Wiki über neue oder geänderte Artikel. Aufgrund der in ACME angewandten agilen Entwicklungsmethodik, arbeiten die einzelnen Teams intensiv mit den bereitgestellten Feeds, um so eine möglichst hohes Maß an Konsistenz gewährleisten zu können: Existiert eine neues Feature? Wenn ja, wurde dies im internen Wiki bereits beschrieben? Abteilungsleiter Steve kontrolliert regelmäßig die Arbeit seiner Angestellten. Auch er analysiert hierfür die verschiedenartigen Informati-

onsströme welche die Werkzeuge bereitstellen. Im Wesentlichen interessieren ihn dabei die einzelnen Aktivitäten der Angestellten, die grundsätzliche Frage, welche sich Steve somit oft stellt lautet: Wer hat wann was gemacht? Die Nachrichtenströme der einzelnen Programme liefern ihm jedoch meist nur vorwiegend technische Formulierungen in der Art „issue 934 created 'SOAP Header contains typo'“ und sind aufgrund der heterogenen Systemumgebung teilweise sehr unterschiedlich in der Art der Formulierung (Syntax), jedoch identisch in Hinblick auf den Informationsgehalt bzw. Typ (Semantik). Bei der großen Menge an eintreffenden Nachrichten ist es demnach für Steve schwierig, aus diesen die zugrunde liegenden Aktivitäten abzuleiten – beispielsweise die Frage, wann Bob zuletzt einen Fehler gemeldet hat.

Das zu konzipierende System soll deshalb die den Nachrichten zugrunde liegenden Aktivitäten erkennen können und diese einheitlich mittels einer geeigneten Syntax modellieren sowie darstellen. Der Nutzer erhält auf diese Weise eine aktivitätsbezogene Sicht auf das System bzw. dessen Nachrichtenströme.

1.4 Fragestellungen

Aufgrund der im vorangegangenen Abschnitt geschilderten Anwendungsfälle, lassen sich nachfolgend wesentliche Fragestellungen dieser Arbeit konkretisieren. Mittels dieser können die zentralen Anforderungen an das System abgeleitet werden, welches im Rahmen dieser Arbeit konzipiert werden soll.

1. Welche Strategien können angewendet werden, um die den Nachrichten zugrunde liegenden **Aktivitäten** extrahieren und modellieren zu können?
2. Wie können **Relationen** zwischen einzelnen Nachrichten detektiert werden?
3. Welche **Arten von Relationen** existieren?
4. Welche Möglichkeiten existieren, um domänenspezifische **Relationsstrukturen** beschreiben zu können?
5. Welche Voraussetzungen müssen geschaffen werden, damit das System in verschiedenartigen Umgebungen und deren Strukturen **domänenunabhängig** instantiiert werden kann?
6. Sind die von den Nachrichtenquellen gelieferten Informationen für ein **automatisiertes System** ausreichend oder müssen zusätzlichen (Meta-) Daten bereitgestellt werden?

1.5 Anforderungen

Nachfolgend werden aus den vorangegangenen Fragestellungen abgeleitete Anforderungen an das zu konzipierenden System beschrieben. Zusätzlich ergeben sich noch weitere nicht-funktionale Anforderungen wie beispielsweise Portierbarkeit, Zuverlässigkeit, Fehlertoleranz und Benutzbarkeit.

- Dem zu konzipierenden System werden heterogene **Informationsquellen** in geeigneten maschinenlesbaren Formaten bereitgestellt, eine eigenständige Suche und Integration von Ausgangsdaten ist nicht Bestandteil der Arbeit.
- Das System besitzt eine **aktivitätsbezogene Sicht** auf die Daten bzw. Domäne. In Folge dessen muss es in der Lage sein, aus den ihm zur Verfügung gestellten Informationsquellen bzw. Datenströmen, einzelne Nachrichtenentitäten zu extrahieren, um aus diesen die zugrunde liegenden Aktivitäten ableiten zu können. Die so gewonnenen Informationen sollen persistent in einem einheitlichen Format in einer zentralen Datenbasis verwaltet werden.
- Die Detektion von Aktivitäten erfolgt aufbauend auf einer **Analyse** der Nachrichten, welche eine Transformation in ein geeignetes Format ermöglicht. Dieser Prozess soll weitestgehend automatisiert erfolgen und möglichst wenig manuelle Eingaben durch den Nutzer bzw. Administrator erfordern.
- Das System soll **Relationen** zwischen einzelnen Nachrichten selbständig erkennen, analysieren und verarbeiten, mit dem Ziel, die so gewonnenen Informationen dem Nutzer mittels geeigneter User-Interface-Metaphern aufbereitet präsentieren zu können. Hierbei sollen verschiedenartige Typen von Relationen berücksichtigt werden, um diese entsprechend zu modellieren und weiterverarbeiten zu können.
- Aufbauend auf durch den Nutzer zu definierenden **Relationsstrukturen**, soll das System in der Lage sein, Muster innerhalb der Nachrichten beziehungsweise Aktivitäten erkennen und abbilden zu können.
- Die Realisierung des zu konzipierenden Systems soll **domänenunabhängig** erfolgen und erst durch Adaption von entsprechenden Konfigurationsparametern auf den jeweiligen Einsatz angepasst werden. Dies betrifft insbesondere die erwähnten Relationsstrukturen sowie Teile der Aktivitätsanalyse.
- Grundsätzliches Ziel des Systems ist eine **aggregierte Sicht** auf die transformierten Quelldaten sowie der Export in einem geeigneten Format.

1.6 Aufbau

Die vorliegende Arbeit gliedert sich wie nachfolgend beschrieben:

Kapitel 2 liefert zunächst ein Überblick über verschiedene *Web-Feed-Formate* und deren Merkmale sowie eine historische Betrachtung. Im darauf folgenden Abschnitt wird das aus diesen entstandene *Activity-Stream-Konzept* im Detail erläutert um im Anschluss daran grundlegende Begrifflichkeiten des *Information Retrieval* vorzustellen. Die darauf folgenden Kapitel *Informationsquellen* sowie *Relationen* beschäftigen sich mit Struktur und Merkmalen relevanter Quelldaten sowie deren potentielle Nachrichtenbeziehungen. Abschließend erfolgt eine Analyse von vorhandenen Systemen (State of the Art) um eine Einordnung des zu konzipierenden Systems zu ermöglichen.

Kapitel 3 schildert die Konzeption des zu erstellenden Systems, indem zunächst die Systemarchitektur und daran anschließend die wesentlichen Mechanismen sowie deren Komponenten im Detail diskutiert werden.

Kapitel 4 beschreibt die prototypische Umsetzung der zuvor beschriebenen Konzepte. Dabei wird die zugrunde liegende Technologiebasis vorgestellt und erläutert, wie die einzelnen Komponenten auf diese aufbauen bzw. realisiert wurden. Neben einzelnen Designentscheidungen wird hierbei auch auf aufgetretene Probleme und deren Lösungen eingegangen.

Kapitel 5 beinhaltet eine Evaluation des entstandenen Systems aufbauend auf einer durchgeführten Nutzerstudie sowie deren Auswertung.

Kapitel 6 fasst die Ergebnisse der Arbeit zusammen, dabei wird auf notwendige Kompromisse während der Realisierung eingegangen und ein Ausblick auf zukünftige Erweiterungen gegeben. Des Weiteren erfolgt eine Analyse der in diesem Kapitel aufgestellten zentralen Fragestellungen und wie diese realisiert, respektive beantwortet, werden konnten.

2 Grundlagen

In diesem Kapitel soll zunächst ein Überblick über verschiedene Web-Feed-Formate und deren Merkmale gegeben werden sowie eine historischer Betrachtung erfolgen. Im darauf folgenden Abschnitt wird das aus diesen entstandene Activity-Stream-Konzept im Detail erläutert. Anschließend werden im Kapitel 2.3 grundlegende und zum Verständnis der Arbeit erforderliche Begrifflichkeiten des *Information Retrieval* vorgestellt. Die darauf folgenden Abschnitte 2.4 *Informationsquellen* sowie 2.5 *Relationen* beschäftigen sich mit Struktur und Merkmalen der zu verarbeitenden Quelldaten sowie deren potentiell zu analysierenden Beziehungen zueinander. Abschließend erfolgt eine Analyse von vorhandenen Systemen *(State of the Art)* um eine Einordnung des zu konzipierenden Systems zu ermöglichen.

2.1 Content Syndication und Feeds

Grundsätzlich dient ein *Feed* dazu, den Nutzer über sich häufig ändernde Inhalte einer Webseite zu informieren. In Folge dessen besteht ein Feed üblicherweise aus chronologisch absteigend geordneten Beiträgen, welche neben dem eigentlichen Text bzw. dem Inhalt einer Nachricht, auch zusätzliche Metadaten wie beispielsweise den Autor und einen Zeitstempel beinhalten. Eine Webseite kann einen oder mehrere Feeds bereitstellen, welche von den Nutzern mittels eines entsprechenden Programms, den sogenannten *Feedreader* oder auch *Aggregator*, abonniert werden können. Diese Programme senden in spezifischen Zeitintervallen Anfragen an den Server, welcher den entsprechenden Feed bereit stellt, um so neue bzw. geänderte Beiträge erhalten zu können. In Folge dessen entspricht dieses System aus Clients (den Feedreadern bzw. Nutzern) und Servern (den Bereitstellern der Feeds) einer sogenannten *Publish-Subscribe-Architektur*. Aus technischer Sicht wird ein Feed durch eine XML-Datei repräsentiert, welche per HTTP-Protokoll geladen und durch einen entsprechenden Feedreader verarbeitet und ausgegeben werden kann. Die grundsätzliche Aufgabe des Readers bzw. des Aggregators ist somit die Visualisierung bzw. das Parsen der XML-Datei. Da dieser nicht aktiv durch den Server des Feeds über neue Daten bzw. Beiträge informiert werden kann, muss dieser selbständig in regelmäßigen Abständen Anfragen an den Server senden, bzw. den Feed (die XML-Datei) abfragen und auf Änderungen überprüfen. In diesem Sinne handelt es sich um eine *Pull-Technologie*. Diese permanentem Abfragen erzeugen insbesondere bei sehr populären Anwendungen/Webseiten (mit mehreren tausend bis Millionen Nutzern) enorme Zugriffslasten, in Folge wurden Ansätze wie beispielsweise *PubSubHubbub*[3] entwickelt, welche das Ziel haben, die Nachrichten aktiv (Push) an den Client bzw. Subscriber zu senden. Im Vergleich zur Webseite selbst, welche ebenfalls in XML (HTML bzw. XHTML) codiert wird, ist ein Feed meist auf die reinen Nutzdaten reduziert, jegliche präsentationsbezogenen Informationen, welche der Visualisierung bzw. Darstellung dienen, entfallen hierbei. Der wesentliche Unterschied in Hinblick auf das Format zwischen einem Feed und einer Webseite besteht darin, dass die Struktur des Feeds durch entsprechende Formate (beispielsweise *Atom* oder *RSS*, vgl. Abschnitt 2.1.1 und 2.1.2) vorgegeben ist, wohingegen eine Webseite nahezu wahlfrei konstruiert werden kann. In Folge dessen kann ein Feed weitaus einfacher und konfliktfreier automatisiert weiterver-

3 http://code.google.com/p/pubsubhubbub/ [11.07.11]

arbeitet bzw. analysiert werden. Ursprünglich wurden Feeds nicht primär als Abonnement für User konzipiert, sondern damit Webseiten gegenseitig fremde Inhalte integrieren können. Beispielsweise indem Nachrichtenbeiträge einer Tageszeitung in externe Webseiten eingebunden werden. Dieser Vorgang ist nicht erst durch das Internet bekannt geworden, unter dem Begriff *Content-Syndication* wird generell der Austausch bzw. eine mehrfache Verwendung von medialen Inhalten verstanden [40]. Seit dem Aufkommen von Feeds sind diese zu einem allgegenwärtigen Werkzeug im Internet geworden, wobei die ursprüngliche Nutzung der Syndikation von fremden Inhalten, nach und nach in den Hintergrund trat und mittlerweile primär die direkte Verwendung durch den Endnutzer im Vordergrund steht. Des Weiteren finden Feeds heutzutage in einer Vielzahl von Szenarien Anwendung, was unter anderem dazu führt, dass eine Webseite nicht mehr nur einen einzelnen Feed bereitstellt. Zahlreiche Anwendungen generieren dynamische Feeds, welche mittels entsprechender Parameter innerhalb der URL gesteuert werden können. Beispielsweise könnte sich ein Nutzer nur für Beiträge mit spezifischen Schlagworten interessieren, diese werden exemplarisch innerhalb der Feed URL codiert, vom Server verarbeitet und als Ergebnis ein individueller Feed generiert. In diesem Sinne hat sich auch die Domäne bzw. das Einsatzgebiet von Feeds stark erweitert: früher wurden diese primär zur Syndikation von Nachrichteninhalten, Blog-Einträgen bzw. News im Allgemeinen eingesetzt. Mittlerweile generieren jedoch eine Vielzahl von Anwendungen in unterschiedlichsten Domänen dynamische Feeds, eine Webseite als Bereitsteller des Feeds im klassischen Sinne entfällt hierbei. Seit dem Aufkommen von Feeds im Jahre 2000 [40] haben sich im Wesentlichen zwei Formate etabliert: *RSS* und *Atom*. Neben diesen existieren noch zahlreiche weitere Vertreter, welche jedoch aufgrund ihrer geringen Verbreitung nur eine sehr begrenze Relevanz besitzen.

2.1.1 RSS

RSS bezeichnet eine Familie von Formaten, welche erstmalig im Jahr 1999 in Erscheinung trat. Die Entwicklung erfolgte jedoch nicht kontinuierlich, sondern in unterschiedlichen Versionszweigen, aus diesem Grund existieren mittlerweile eine Vielzahl von unterschiedlichen Versionsnummern: 0.9, 0.91 bis 0.94, 1.0, 2.0 sowie noch einige Zwischenversionen. Dabei stehen die einzelnen Versionsnummern zwar in Bezug zueinander, da man durchaus die Absicht hatte, eine Weiterentwickelung auch als solche zu kennzeichnen, jedoch bauen die einzelnen Versionen nicht direkt aufeinander auf. In Folge dessen sind die einzelnen Versionen nur bedingt zueinander kompatibel, eine Transformation untereinander ist zwar prinzipiell möglich, je nach Umfang bzw. Ausdrucksstärke fehlen jedoch einzelne Daten. Dies hat historische Gründe, welche nachfolgend kurz beschrieben werden sollen: Version 0.90 stellt das älteste Format dar, es wurde im Jahr 1999 durch *Netscape* entwickelt und basierte zumindest in Ansätzen auf RDF (Resource Description Framework), wodurch sich die Bezeichnung *RDF Site Summary* für RSS ergab. Unabhängig hiervon entwickelte die Firma *UserLand Software* Version 0.91 eigenständig weiter, verzichtete jedoch auf RDF, indem einzelne Elemente entfernt bzw. vereinfacht wurden. Die Spezifikation wurde im Jahr 2000 erstmalig veröffentlicht, RSS steht bei dieser für *Rich Site Summary*. Parallel zu diesen wurde mit Version 1.0 ein weiterer Versionszweig geschaffen, welcher durch eine unabhängige Entwicklergruppe, die sogenannte *RSS-DEV Working Group*, entstandenen ist. Im Gegensatz zu 0.9x basierte 1.0 wieder auf dem RDF Standard. Die mittlerweile am weitesten verbreitete Version stellt jedoch 2.0 dar, welche im Jahr 2002 durch die bereits erwähnte Firma *UserLand* veröffentlicht wurde. Sie verzichtet ebenfalls wieder auf RDF, das Akronym RSS steht hierbei für

Really Simple Syndication. Rückwirkend lassen sich somit zwei Hauptzweige erkennen: die auf RDF basierenden Versionen 0.9 und 1.0 (bzw. deren Zwischenversionen) sowie 0.9x bzw. 2.0. In Anhang A.1 befindet in einfaches Beispiel einer RSS 2.0 Datei, da diese Version laut [50] mit Abstand die größte Verbreitung aufweist, wird auf eine detaillierte Betrachtung der Unterschiede verzichtet. Jeder Feed enthält bei dieser Version ein *channel* Element, welches Metadaten sowie die eigentlichen Beiträge *(item)* beinhaltet. Deutlich erkennbar ist zudem, dass jeder Beitrag einen Titel, Link, Datum sowie den eigentlichen Inhalt beschreibt. Laut Spezifikation sind diese Elemente jedoch optional, lediglich die Angaben über den Feed selbst (*title*, *link* und *description*) sind obligatorisch. Da RSS 2.0 mittels XML bzw. einer entsprechenden DTD spezifiziert ist, können weitere Elemente durch entsprechende (XML-Schema) Erweiterungen hinzugefügt werden, innerhalb der Domänen von RSS werden diese als *Module* bezeichnet [40]. Als gemeinsames Ziel dieser Module kann die Verbesserung der Definition der Metadaten des ursprünglichen RSS Formates angesehen werden, da dies einen der wesentlichen Nachteile des Formates darstellt: die einzelnen Elemente (beispielsweise *description*) enthalten keine Angaben über den Typ der enthaltenen Daten. Problematisch bei RSS 2.0 ist zudem die Tatsache, dass es urheberrechtlich geschützt ist und in Folge dessen nur bedingt durch Dritte weiterentwickelt werden kann.

2.1.2 Atom

Atom ist ein zu RSS in Konkurrenz stehendes Format, welches im Jahr 2003 initiiert und 2005 als RFC durch die IETF verabschiedet wurde. Die Entwicklung wurde unter anderem durch die teilweise verwirrende Versionsvielfalt der RSS Formate ausgelöst und versucht die Vorteile dieser zu vereinheitlichen und zu erweitern. Atom ist zudem intensiv durch die Bedürfnisse der seitdem stark verbreiteten Blogs beeinflusst. Formal gesehen handelt es sich bei dem Begriff *Atom* um zwei Standards: das sogenannte *Atom-Syndication-Format (ASF)* sowie *Atom-Publishing-Protocol (APP)*. Ersteres ist das in den Sprachgebrauch übergegangen Synonym für Atom und bezeichnet das zu RSS in Konkurrenz stehende Web-Syndikations-Format bzw. Standard, wohingegen APP eine Schnittstelle definiert, welche dazu dient, sogenannte *Web Ressourcen* zu erstellen bzw. zu ändern. Einer der wesentlichen Vorteile von Atom gegenüber RSS besteht in der Möglichkeit, die Datentypen von Elementen (beispielsweise dem eigentlichen Inhalt) explizit angeben zu können (mittels eines *type*-Attributes). Dies ist bei RSS nicht möglich, in Folge dessen muss der Aggregator selbständig Annahmen treffen, welcher unter Umständen zu fehlerhaften Ausgaben führen können. Im Gegensatz zu RSS sind viele semantisch gleichwertige XML-Elemente obligatorisch, das heißt, es ist nicht möglich Beiträge ohne Inhalt bzw. Titel anzugeben. Des Weiteren unterscheidet Atom explizit zwischen gekürzten *(summary)* und vollständigen *(content)* Inhalten innerhalb eines Beitrages *(entry)*. Dies ist insofern relevant, als dass viele Webseiten nicht den gesamten Inhalt einer Nachricht in den entsprechenden Feed integrieren, mit dem Ziel, dass Nutzer auch die eigentliche Webseite aufrufen. RSS kennt diesbezüglich keine Unterscheidung (es existiert nur das *description*-Element), in Folge dessen kann ein Aggregator ohne zusätzliche Informationen nicht wissen, ob es sich bei einem Beitrag lediglich um einen Auszug handelt. Die Atom Spezifikation befindet sich im Gegensatz zu RSS 2.0 in einem eigenständigen XML-Namensraum [75] und bietet in Folge dessen bessere Möglichkeiten für Erweiterungen. Beispielsweise ist es ist möglich, Elemente und Attribute von externen Namensräumen einzubinden. Der Standard gibt diesbezüglich klare Vorgaben, wie derartige Daten zu verarbeiten sind. Umgedreht ist es somit auch möglich,

Daten von Atom in eigene Namensräume zu integrieren, in Anhang A.1 befindet in einfaches Beispiel einer Atom Datei.

2.2 Activity Streams

Stark durch das Aufkommen von sozialen Netzwerken beeinflusst, zeigte sich, dass die im vorangegangenen Abschnitt beschriebenen Syndikationsformate eine entscheidende Einschränkung haben: der eigentliche Inhalt eines Beitrages ist zwar im Falle von Atom klar typisiert, jedoch sagt keiner der beiden Standards etwas über die Semantik aus. Im klassischen Einsatzgebiet dieser Formate war dies auch nicht vorgesehen, Ziel war die Syndikation, nicht Interpretation, von externen Inhalten. Aktuelle Entwicklungen zeigen jedoch, dass genau dies zunehmend in den Fokus des Nutzerinteresses rückt. Das am weitesten verbreitete [88] soziale Netzwerk *Facebook*[4] führte im Jahr 2006 [27] einen sogenannten *News Feed* ein, dieser zeigt ähnlich einem (Atom bzw. RSS) *Web Feed* die neusten Ereignisse relevanter Kontakte an. Im Gegensatz zu einem klassischen Feed ist die Sichtweise jedoch aktivitätsbezogen, das heißt, es wird nicht wahlfreier Inhalt exportiert - ein Beitrag unterliegt einer eindeutigen Subjekt-Prädikat-Objekt Struktur. In [29] sind mehrere derartige Konstrukte abgebildet, beispielsweise *„Dana Hornbeak* [Subjekt] *removed* [Prädikat] *'piano'* [Objekt] *from her interests“.* Seit der Einführung dieses aktivitätsbezogenen Nachrichtenstroms innerhalb von *Facebook*, welches ein Patent diesbezüglich angemeldet hat [80], wurde dieses Prinzip von zahlreichen weiteren System übernommen.

2.2.1 Historie und Hintergrund

Das Ziel der klassischen Feed-Formate war primär die Integration von fremden Inhalten in externe Webseiten, die sogenannte *Syndikation*. In Folge dessen waren nur sehr wenige dedizierte Felder (repräsentiert durch die entsprechenden XML-Elemente innerhalb der Feed-Formate) notwendig: Titel sowie Inhalt waren für die meisten Anwendungsfälle ausreichend, da dies auch dem klassischen Format einer Nachrichtenschlagzeile entspricht und diese Struktur somit komfortabel abgebildet werden konnte. Dies spiegelt auch die ursprünglichen RSS (aus dem Jahr 1999) Spezifikation (vgl. Abschnitt 2.1) wieder: ein Feed-Eintrag besteht lediglich aus Titel, Link sowie Beschreibung. Mit zunehmender Verbreitung von Web Feeds wurden diese immer öfter direkt durch den Endnutzer mittels Feed-Aggregatoren genutzt, die ursprüngliche Syndikation rückte damit in den Hintergrund. Dabei erwies sich die hohe Flexibilität bzw. geringe Anzahl an Metadaten zunehmend als Problem, da die Aggregatoren zahlreiche Analysen durchführen müssen, um die eigentliche Semantik eines Eintrages erkennen zu können (vgl. beispielsweise [76]). Atom (2005) bot hierfür bereits umfassendere Möglichkeiten, da ein Beitrag neben dem Titel, einem Link und einer Zusammenfassung auch den Autor, eine eindeutige Identifikationsnummer sowie detaillierte Datumsfelder vorsieht. Einem Aggregator standen somit umfassendere Möglichkeiten zur Verfügung, einen Beitrag entsprechenden zu analysieren und zu klassifizieren. Unabhängig hiervon bestand aber weiterhin das Problem, dass der Inhalt eines Eintrages nach wie vor keinerlei Struktur vorsah. Im Laufe der Zeit entstanden auch für RSS verschiedenartige Erweiterungen, sogenannte *Module* (vgl. Abschnitt 2.1.1), welche ebenfalls das Ziel hatten, die Basis-Metadaten zu erweitern. Bekannte Vertreter sind beispielsweise *Dublin Core* oder *Atomic RSS* [70].

4 http://www.facebook.com [02.07.08]

Allen gemein ist die Motivation, die Ausdrucksstärke von RSS zu verbessern bzw. zu formalisieren. Mit dem Aufkommen und der immer stärkeren Verbreitung von sozialen Netzwerken, rückte eine aktivitätsbezogene Sichtweise stärker in den Vordergrund: Nutzer sahen keine abstrakten Statusnachrichten mehr, sondern klar strukturierte Aktivitäten. Facebook kann hierbei als Begründer dieses Prinzips gelten, es führte im Jahr 2006 einen sogenannten *News Feed* ein, welcher die Aktivitäten eines Nutzers einheitlich mittels einer Subjekt-Prädikat-Objekt Struktur (SPO) abbildet (siehe Einleitung in Abschnitt 2.2). In Folge dessen ergibt sich ein Nachrichtenstrom von Aktivitäten, ein sogenannter *Activity Stream*. Viele weitere Dienste und Anwendung des Web 2.0 implementierten dieses Prinzip ebenfalls, jedoch zeigte sich, dass eine Aggregation mittels klassischer Feed-Formate problematisch war. Innerhalb der einzelnen Anwendungen war eine entsprechende aktivitätsbezogene Sicht bzw. Syntax unkritisch, jedoch existierte für den Export nach wie vor nur RSS, respektive Atom, wollte man auf etablierte Standards setzen. In Folge dessen mussten die Aktivitäten innerhalb der Inhaltselemente prinzipbedingt wieder auf einfachen unformatierten Text abgebildet bzw. eigenständige Auszeichnungselemente[5] eingeführt werden, damit externe Anwendungen die SPO-Struktur wieder extrahieren können. Dies soll an nachfolgendem Beispiel verdeutlicht werden: *github*[6] bietet einen zentralen Nachrichtenstrom von aktuellen Aktivitäten, die sogenannte *timeline*[7], an. Intern kann somit die SPO-Struktur abgebildet werden. In Abbildung 1 ist beispielsweise der Eintrag „tassia closed issue 2 on tassia/AppRecommender“ erkennbar. *Github* nutzt intern klar definierte Prädikate, wodurch die Möglichkeit besteht, beispielsweise alle *close*-Aktivitäten des Nutzers *tassia* innerhalb des letzten Monats zu ermitteln.

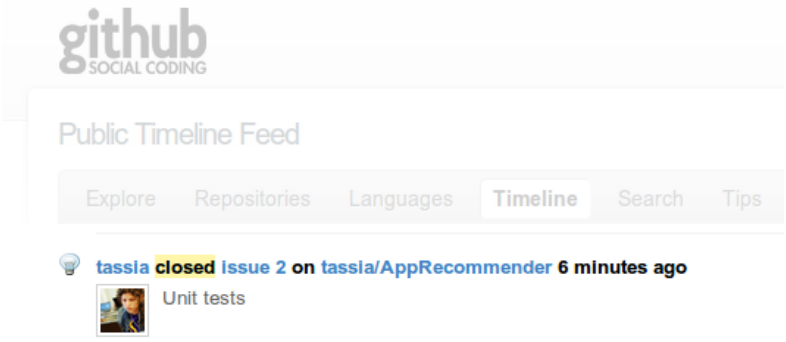


Abbildung 1: Beispiel einer Aktivität innerhalb github

Betrachtet man den entsprechenden Atom Feed, welcher durch *github* bereitgestellt wird, so wird dieser Beitrag wie in Listing 1 zu erkennen ist repräsentiert.

5 beispielsweise mittels HTML-Syntax und CSS-Klassennamen

6 ein webbasierter Hosting-Dienst für Software-Entwicklungsprojekte - http://github.com [11.07.11]

7 https://github.com/timeline [11.07.11]

```
1   <entry>
2       <id>tag:github.com,2008:IssueCommentEvent/1477571321</id>
3       <published>2011-07-15T08:20:38Z</published>
4       <updated>2011-07-15T08:20:38Z</updated>
5       <link type="text/html" rel="alternate" href="https://github.com/tass[...]"/>
6       <title>tassia commented on issue 2 on tassia/AppRecommender</title>
7       ...
8   </entry>
```

Listing 1: Beispiel github Aktivität innerhalb eines Atom Feeds

In Zeile 6 ist deutlich erkennbar, dass die Aktivität nur noch durch einen flachen unstrukturierten String repräsentiert wird. Ein externer Aggregator müsste nun aufwändige[8] Analysen durchführen, um die ursprüngliche SPO-Struktur wieder extrahieren zu können, da diese nun in einer schlecht maschienenlesbaren Form vorliegt. Die Situation betrifft insbesondere innerhalb des Web 2.0 zahlreiche Dienste, welche intern eine aktivitätsbezogene Struktur verwenden, jedoch nach außen für externe Anwendung alle identisch erscheinen.

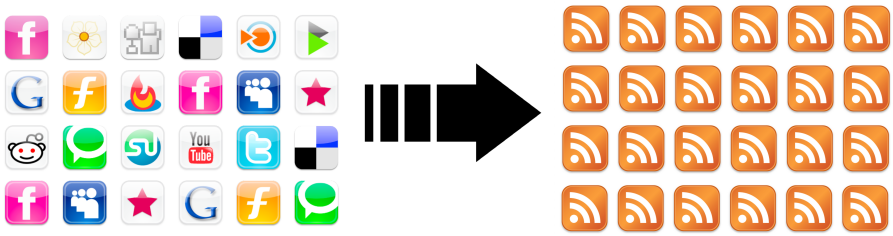


Abbildung 2: Abstrakte Darstellung heterogener Anwendungen und deren Feeds [63]

Abbildung 2 illustriert diese Situation, verschiedenartige Dienste und deren Aktivitätsströme sind nach außen stets identisch. In Folge dessen muss ein Aggregator für jede Datenquelle eigenständige Adapter implementieren, um die einzelnen Aktivitäten aus der flachen Textrepräsentation extrahieren zu können. Im Jahr 2007 entstand aus diesem Grund unter anderem der Echtzeit-Feed-Aggregator *FriendFeed* (vgl. Abschnitt 2.7.3), welcher diese Aufgabe löste, indem für jede Datenquelle (Feed) individuell ein Parser generiert wurde. Zwischenzeitlich wuchs die Anzahl dieser somit auf über 50 Stück an. Dies hat zur Folge, dass ein entsprechender Dienst kontinuierlich die unterstützten Datenquellen und deren Formate überwachen muss - ändert einer der Anbieter das Format oder die Syntax, muss der Aggregator ebenfalls Anpassungen vornehmen. Dies wird mit zunehmender Anzahl an Datenquellen sehr zeitaufwändig und erfordert kontinuierliche Wartungsarbeiten um die Funktionalität gewährleisten zu können.

Fazit Um dieses Problem zu umgehen, wurde im Jahr 2008 damit begonnen, eine einheitliche Repräsentation von Aktivitätsströmen zu entwickeln. Die so entstandenen Spezifikation soll im nachfolgenden Abschnitt näher beschrieben werden.

8 Unter der Annahme, dass die Syntax nicht detailliert bekannt bzw. stark variabel ist.

2.2.2 Spezifikation

Die Activity-Stream-Spezifikation [5] ist offenes Format zur Definition von Aktivitäten, welche im Jahr 2008 initiiert wurde und mittlerweile in der Version 1.0 vorliegt. Bei der Spezifikation handelt es sich um eine Erweiterung des Namensraumes des Atom Formates. Atom erlaubt im Wesentlichen lediglich die Definition eines Titels, Autors sowie des Inhaltes eines Beitrages. Dabei ist es jedoch nicht möglich, innerhalb des Inhaltes weitere Differenzierungen vornehmen zu können - zur Definition von maschienenlesbaren Aktivitäten der Struktur Subjekt-Prädikat-Objekt, fehlen somit entsprechende Metadaten bzw. Felder. Ziel der Spezifikation war somit eine Erweiterung des Atom Formates, um diese Metadaten adäquat innerhalb vorhandener Feeds bzw. Beiträgen abbilden zu können. Diese Metadaten sind umfassend genug, damit externe Anwendungen aus den Information eines Activity Streams beispielsweise vollständige, für den Nutzer lesbare, Sätze konstruieren können. In der einfachsten Form besteht ein Eintrag innerhalb eines Activity Stream aus einem Akteur (dem Subjekt), einem Verb (dem Prädikat) sowie einem Objekt. Hieraus ergibt sich die bereits erwähnte Syntax der Form „Bob löschte ein Bild“. Innerhalb eines Activity Streams besteht eine Aktivität aus verschiedenartigen (logischen) Komponenten, welche in Anhang A.2 innerhalb der Tabelle 4 detailliert charakterisiert werden. Die Activity-Stream-Spezifikation sieht vor, dass jeder Eintrag mindestens die Elemente *Time*, *Actor*, *Verb* und *Object* besitzt, die restlichen Angaben sind optional. In Tabelle 4 wurde bereits angedeutet, dass einige Element nicht atomar sind, sondern sich aus weiteren Unterelementen zusammensetzen, dem sogenannten Objektkonstrukt. Ein Objektkonstrukt ist ein Gegenstand (nicht zwingend real) bzw. eine Sache, welche(r) an einer Aktivität beteiligt ist, hierunter fallen *Actor*, *Object* und *Target*. Diese bestehen selbst wiederum aus logischen Komponente, welchen innerhalb der Tabelle 5 (Anhang A.2) erläutert werden. Die Activity-Stream-Spezifikation bietet für Objektkonstrukte definierte Erweiterungsmöglichkeiten, auf welche aber an dieser Stelle nicht weiter eingegangen werden soll. In den vorangegangenen Beschreibungen der einzelnen Komponente wurde bereits auf einen der Kerngedanken der Activity-Stream-Spezifikation hingewiesen, der Beschränkung des Wertebereiches einzelner Attribute. Dies betrifft primär Verben und Objekte. Die Spezifikation sieht vor, dass die Menge der möglichen **Prädikate**, respektive **Verben**, einer Aktivität begrenzt ist und deren Semantik klar definiert ist. Technisch realisiert wird dies, indem das entsprechende XML-Element mittels einer anzugebenden absoluten IRI eindeutig auf ein Element (dem Prädikat) innerhalb des Activity-Stream-Namensraumes verweist. In der Version 1.0 existieren insgesamt 29 verschiedenartige Prädikate [4]. Jedes Prädikat besitzt eine genau definierte Semantik, welche global eindeutig ist. Dies stellt einen der Kernaspekte der Activity-Stream-Spezifikation dar, Prädikate werden explizit als Entitäten definiert, benutzen also zwei unterschiedliche Anwendungen das gleiche Prädikat, so kann mit Sicherheit davon ausgegangen werden, dass dieses auch global die gleiche Bedeutung hat. Des Weiteren verbessert sich durch diese Struktur die automatisierte Auswertung durch externe Anwendungen erheblich, was im Ergebnis unter anderem dazu führt, dass vergleichsweise einfach Analysen über einer Menge von Activity-Stream-Einträgen konstruiert werden können, beispielsweise die Frage: Wann hat Bob zuletzt ein Foto zu einem Album hinzugefügt? Derartige Abfragen wären in klassischen Feeds nur mit erheblichen Aufwand realisierbar, da letztendlich NLP-Algorithmen (Natural Language Processing) auf den Inhalt eines Beitrages angewendet werden müssten. **Objekte** besitzen analog zu Verben ebenfalls einen klar definierten Wertebereich mit 18 Ausprägungen [4]. In Anhang A.1 ist ein beispielhafter Activity-Stream-Auszug dargestellt, in welchem die

entsprechende SPO-Struktur erkennbar ist: in den Zeilen 8-14 wird das Subjekt bzw. der Autor definiert. Dieses Element existiert bereits innerhalb der Atom-Spezifikation und wird daher weitestgehend übernommen. In Zeile 15 ist das Prädikat der Aktivität definiert, in diesem Fall *post*. In Zeile 16 bis 25 wird anschließend das Objekt der Aktivität deklariert, dieses besitzt eine Vielzahl von Attributen, welche bereits in Tabelle 5 (Anhang A.2) vorgestellt wurden. Mittels des Elementes *object-type* (Zeile 24) wird der bereits erwähnte Verweis auf den global eindeutigen Typ des Objektes definiert. Eine externe Anwendung ist mit diesen Daten sprachneutral in der Lage, eine entsprechende Formulierung abzuleiten, beispielsweise „The person Geraldine posted the photo 'My Cat' to the album 'My Pets'". Die genaue Formulierung obliegt dabei der jeweiligen Endanwendung, die Activity-Stream-Spezifikation definiert bzw. liefert lediglich die hierfür notwendigen (Meta-) Datenbasis. **Erweiterung** des Standards sind bereits von Beginn an berücksichtigt wurden. Die Spezifikation kennt gegenwärtig beispielsweise Elemente um Ortsangaben oder Bewertungen einer Aktivität ausdrücken zu können. Hierfür werden weitere Metadaten bzw. XML-Elemente zu den Entitäten hinzugefügt. Nutzer des Activity-Stream-Formates können je nach Domäne weitere Attribute deklarieren, um so spezifische Anforderungen abdecken zu können.

2.2.3 Vorteile

Nachfolgend sollen noch einmal die Kernelemente der Activity-Stream-Spezifikation und die daraus resultierenden Vorteile für externe Anwendungen, respektive Nutzer, verdeutlicht werden.

- Durch die Verwendung eines klar abgegrenzten Wertebereiches von Prädikaten und Objekten können Aktivitäten problemlos auf **verschiedene Sprachen** abgebildet werden.
- Prädikate und Objekte sind klar definiert und mit einer **eindeutigen Semantik** versehen.
- Aufgrund der SPO-Restriktion ist es für Nutzer leichter, die Informationen zu erfassen, da auch in einer heterogenen Umgebung stets die **gleiche Syntax** verwendet wird.
- Consumer (verarbeitende Anwendungen) von Feeds bzw. Aggregatoren können mittels Activity Streams problemlos Abfragen über gesammelte Einträge generieren, beispielsweise für statistische Auswertungen bzw. generell einer **Weiterverarbeitung**.
- Activity-Stream-Einträge besitzen wesentlich mehr **Metadaten**, was beispielsweise umfassendere Analysen ermöglicht.

2.3 Information Retrieval

Das Gebiet des sogenannten *Information Retrieval* beschäftigt sich mit Fragestellungen, mit welchen Methodiken große Datenbestände (in diesem Kontext als *Korpus* bezeichnet) nach komplexen Mustern, respektive Inhalten, durchsucht werden können (Suchoperationen werden meist als Ab- bzw. Anfragen bezeichnet). In [61] wird dies wie folgt definiert:

> *„Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).“*

Insbesondere mit zunehmender Verbreitung des Internets und den in damit Zusammenhang stehenden Suchmaschinen hat dieses Gebiet stark an Bedeutung gewonnen, da letztere performante Algorithmen benötigen, um die teilweise enorm großen Datenbestände in kurzer Zeit durchsuchen zu können. Aufgrund der Tatsache, dass diese Thematik in zahlreichen Arbeiten ausführlich behandelt wurde, sollen an dieser Stelle nur jene für die vorliegende Arbeit relevanten Bereiche kurz erläutert werden. Der Leser soll so ein grundlegendes Verständnis für die verwendeten Begrifflichkeiten erhalten. Eine sehr ausführliche Beschreibung findet sich beispielsweise in [61], kompaktere Darstellungen hingegen in [73] und [28].

2.3.1 Grundlegende Begriffe

Information Need steht im Kontext des *Information Retrieval* für das Bestreben eines (meist) Individuums, ein (Informations-) Bedürfnis zu befriedigen. Diese (Bedürfnisse) werden in diesem Kontext meist in Form von formalisierten Ausdrücken *(Query)* dargestellt und durch ein entsprechendes System verarbeitet.

Index Um großen Datenmengen effektiv durchsuchen zu können, wird meist ein invertierter Index generiert, welcher die Grundlage für Suchoperationen bildet. Dieser lässt sich weitaus schneller durchsuchen, als beispielsweise sequentiell über jegliche Dokumente zu iterieren.

Vektorraummodell bezeichnet eine sogenanntes *Retrievalmodell*, welches Dokumente und Suchanfragen mittels hochdimensionaler Vektoren abbildet, jedes Wort entspricht hierbei einer Dimension.

Kosinus-Ähnlichkeit bezeichnet eine Berechnungsvorschrift innerhalb des Vektorraummodells, welche die Ähnlichkeit zweier Dokumente (repräsentiert durch Vektoren, in diesem Sinne meist Vektor der Anfrage und Vektor des Vergleichsdokumentes im Korpus) auf Basis einer Winkelberechnung ermittelt.

Termfrequenz ist ein Maß zur Berechnung der Relevanz eines Terms innerhalb eines Dokumentes. Die Termfrequenz tf für einen Term t in einem Dokument d ergibt sich aus der Summe seiner Vorkommen in d $(n_{t,d})$ dividiert durch die Gesamtzahl aller Terme im Dokument (N_d). Grundsätzlich lässt sich somit feststellen, dass mittels dieser Berechnungsvorschrift einzelne Terme umso höher gewichtet werden, je öfter sie in einem Dokument vorkommen. Dies entspricht auch der intuitiven Annahme, dass Terme mit

einer hohen Frequenz ein Dokument stärker charakterisieren als jene mit niedriger.

$$tf_{t,d} = \frac{n_{t,d}}{N_d}$$

Inverse Dokumentfrequenz Die *Inverse Dokumentfrequenz* des Terms t *(idf_t)* berechnet sich aus dem Logarithmus der Menge aller Dokumente *($N=|D|$)* dividiert durch die Anzahl der Dokumente, die Term t beinhalten *(n_t)*. Die IDF ist somit ein Maß für die Relevanz eines Terms im gesamten Korpus und führt dazu, dass Terme mit vergleichsweise häufigem Auftreten (in vielen Dokumenten) als tendenziell irrelevanter klassifiziert werden.

$$idf_t = \log \frac{N}{n_t}$$

Termfrequenz – Inverse Dokumentfrequenz *(tf-idf)* kombiniert die zuvor genannten Berechnungen und misst somit die Relevanz eines Terms in Abhängigkeit des gegenwärtigen Dokumentes sowie des Korpus.

$$tf-idf_{t,d} = tf_{t,d} \cdot idf_t$$

Stoppwörter bezeichnen im Kontext des *Information Retrieval* Wörter eines Textes, welche keine oder eine nur sehr geringe Ausdruckskraft *(Signifikanz)* besitzen. Dies betrifft unter anderem Füllwörter, sprachliche Konjunktionen (*und, oder, um, …*) oder Präpositionen. Innerhalb des Analyseprozesses werden diese Wörter detektiert und aus den Dokumenten bzw. bei der Indizierung entfernt.

Stemming ist ebenfalls ein Vorverarbeitungsschritt und bezeichnet die Transformation einzelner Wörter auf deren Grundform, beispielsweise *fishing* → *fish*. Im Gegensatz zu einem sogenannten *Lemmatizer* wird jedoch nur jedes Wort für sich genommen betrachtet und der Kontext ignoriert. Dies führt in einigen Fällen zu falschen Reduktionen [61], da ein *Stemmer* tendenziell mittels Heuristiken arbeitet und keine Kenntnis über die Bedeutung eines Wortes hat. Des Weiteren entsprechen die Grundformen nicht zwingend korrekten Wörtern. Einer der bekanntesten Stemmer-Algorithmen ist der *Porter-Stemmer-Algorithmus*[9] sowie der *Snowball-Algorithmus*[10].

Lemmatizer sind eng verwandt mit Stemming-Algorithmen, berücksichtigen jedoch stärker die zugrunde liegende Sprache und deren Charakteristika und sind in Folge dessen eher im Gebiet der Linguistik anzusiedeln. Im Gegensatz zum *Stemming* analysieren diese den Kontext eines Wortes (Sätze) und reduzieren einzelne Wörter nur auf korrekte, innerhalb der Sprache existierende, Grundformen. Bekannte Lemmatisierer sind beispielsweise der *Stanford Part-Of-Speech Tagger*[11] (als Bestandteil dessen) sowie *MorphAdorner*[12].

9 http://www.tartarus.org/~martin/PorterStemmer [02.10.11]
10 http://snowball.tartarus.org [02.10.11]
11 http://nlp.stanford.edu [02.10.11]
12 http://morphadorner.northwestern.edu [02.10.11]

2.3.2 Evaluierungskriterien

Im Bereich des Information Retrieval haben sich verschiedene Metriken bzw. Verfahren zur Bewertung von Such- und Ranking-Ergebnissen etabliert. Diese sollen nachfolgend nur kurz erläutert werden, da in einer Vielzahl von Arbeiten diesbezüglich bereits ausführliche Erklärungen existieren [28, 50, 61]. Das grundsätzliche Ziel ist die Bewertung der Qualität eines Systems, insbesondere ist nicht die Datenverarbeitungsleistung Bestandteil derartiger Betrachtungen. Unter dem Begriff *Qualität* werden in diesem Zusammenhang meist folgende Fragestellungen untersucht: „*Wie hoch ist der Anteil der tatsächlich relevanten Informationen am gelieferten Ergebnis?*“ sowie „*Wie hoch ist der Anteil der relevanten Dokumente am gelieferten Ergebnis in Bezug auf die Anzahl der tatsächlich vorhandenen relevanten Dokumente?*“ [50]. Innerhalb des Kontextes der Arbeit entspricht die Relevanz eines Dokumentes einer konkreten Relation, dies bedeutet, an das System wird eine Anfrage gesendet (eine Anfrage entspricht einer Nachricht) und die in Relation stehenden Nachrichten werden als Ergebnis zurück geliefert. Für eine Evaluation kann in Folge dessen untersucht werden, welche Nachrichten korrekt als in Relation stehend erkannt werden konnten bzw. welche nicht, respektive fälschlicherweise, ausgegeben wurden. Diese Ausprägungen werden meist mittels einer sogenannten *Konfusionsmatrix* erfasst, welche nachfolgend dargestellt ist.

	Nachricht A steht in Relation zu Nachricht B	**Nachricht A steht *nicht* in Relation zu Nachricht B**
Relation wurde erkannt	richtig positiv	falsch positiv
Relation wurde nicht erkannt	falsch negativ	richtig negativ

Tabelle 1: Konfusionsmatrix

Genauigkeit (precision) Precision beschreibt die Genauigkeit eines Suchergebnisses, indem analysiert wird, in welchem Maß irrelevante Dokumente vom Ergebnis ausgeschlossen werden.

$$precision = \frac{|\{ relevant\ documents \} \cap \{ retrieved\ documents \}|}{|\{ retrieved\ documents \}|}$$

Trefferquote (recall) ist ein Maß der Vollständigkeit eines Suchergebnisses, indem der Anteil von relevanten Dokumenten (welche bei einer Suche gefunden wurden) von der Gesamtmenge aller relevanten Dokumente ermittelt wird[13].

$$recall = \frac{|\{ relevant\ documents \} \cap \{ retrieved\ documents \}|}{|\{ relevant\ documents \}|}$$

13 Eine isolierte Betrachtung der Trefferquote ist meist nur bedingt aussagekräftig, da ein System theoretisch zu jeder beliebigen Anfrage den gesamten Korpus als Antwortmenge zurück liefern könnte und in Folge dessen stets den Maximalwert erhalten würde.

Trefferquote und Genauigkeit sind innerhalb der meisten Systeme nicht isoliert von einander, sondern beeinflussen sich gegenseitig: prinzipbedingt sinkt die Genauigkeit (irrelevante Ergebnisse nehmen zu) mit steigender Trefferquote. Vice versa sinkt die Trefferquote mit steigender Genauigkeit.

F-Maß (f-measure) entspricht der Kombination von Trefferquote und Genauigkeit, indem das sogenannte *gewichtete harmonische Mittel* eingesetzt wird. Die dargestellte Formel gewichtet Trefferquote und Genauigkeit zu gleichem Maße und wird aus diesem Grund auch als F_1 bezeichnet.

$$f\ measure\ =\ 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Problematisch bei der Berechnung der erwähnten Maße ist jedoch die Tatsache, dass die Menge der relevanten Dokumente (bezogen auf eine Anfrage) meist nicht bekannt ist – da genau diese durch ein entsprechendes System erst untersucht wird. Um dennoch aussagekräftige Werte ermitteln zu können (welche letztendlich eine Notwendigkeit für Vergleiche von verschiedenen Systemen darstellen), werden in verschiedenen Szenarien sogenannte *Gold Standards* [61] verwendet, welche anerkannte und (tendenziell) korrekte Ergebnisse und zugehörige Abfragen enthalten.

Ranking-basierende Metriken Es existieren weitere Metriken wie beispielsweise *Mean Average Precision* und *Mean Reciprocal Rank*, welche jedoch das Ziel haben, Messwerte bezüglich nach Relevanz sortierter Ergebnismengen ermitteln zu können. Innerhalb des zu betrachtenden Kontextes existieren jedoch keine sortierten Mengen (von Relationen). In Folge dessen kommen derartige Metriken in diesem Kontext nicht in Betracht.

2.4 Informationsquellen

Der nachfolgende Abschnitt soll einen Überblick über relevante Werkzeuge bzw. Webanwendungen geben, welche potentiell als Datenquellen von Informationsströmen in Frage kommen. Die nachfolgend vorgestellten Systeme sind prinzipiell domänenunabhängig und als Gattungsbegriff für eventuelle Spezialanwendungen zu verstehen (beispielsweise ist das Prinzip von Mailinglisten an keine spezielle Domäne gebunden).

Issue-Tracking-Systeme Unter Issue-Tracking-Systemen (ITS), auch Fallbearbeitungssysteme genannt, werden Anwendungen zur Aufnahme und Verwaltung von sogenannten *Tickets* verstanden. Innerhalb der Domäne der Softwareentwicklung werden diese (ITS) meist als *Tracker* bezeichnet, wodurch sich sogenannte Bugtracker (Verwaltung von Fehlern), respektive Featuretracker (Funktionalitäten), ableiten. Unabhängig der Domäne arbeiten diese Systeme prinzipiell auf die gleiche Art und Weise: ein Nutzer erstellt ein Ticket, welches meist einen Titel sowie eine Beschreibung besitzt. Innerhalb des Systems wird diesem automatisiert ein systemweit eindeutiger Identifikator gegeben (eine Ticketnummer bzw. ID). Unter der Annahme, dass das Issue-Tracking-System einen Feed bereit stellt, beinhaltet dieser somit Informationen über neue und geänderte Tickets innerhalb des Systems. Die Ticketidentifikatoren (IDs) stellen ein wichtiges

Merkmal innerhalb einer Systemumgebung dar (vgl. Abschnitt 2.5.3.1), da davon ausgegangen werden kann, dass deren Zeichenketten in verschiedenen anderen Tools ebenfalls auftreten, beispielsweise wenn Diskussionen innerhalb eines Forums sich mit diesen beschäftigen. Eine umfassende Liste verbreiteter Issue-Tracking-Systeme ist unter [16] zu finden. Neben der Verwaltung der eigentlichen Reports bieten die meisten Systeme die Möglichkeit, Kommentare zu den jeweiligen Meldungen abgeben zu können.

Wikis[14] sind hypertextbasierende Programme[15], welche mittels eines vereinfachten CMS lesenden und schreibenden Zugriff auf Inhalte ermöglichen, meist direkt innerhalb des Browsers. Eine der bekanntesten Anwendungen stellt die Online-Enzyklopädie Wikipedia dar. Wikis erlauben einen kollaborativen Erstellungs- und Bearbeitungsprozess von Inhalten bzw. Datenbanken und können in diesem Kontext zahlreiche Informationsströme bereitstellen, beispielsweise über neue oder geänderte Artikel. Da derartige Systeme oft im Kontext des Dokumentationsprozesses eingesetzt werden, können relevante Beziehungen beispielsweise aus der Analyse der Änderungen eines ITS und deren Auswirkungen auf das Dokumentationswiki detektiert werden.

Mailinglisten sind eine der ältesten Kommunikationsformen innerhalb des Internets und stellen gewissermaßen den Vorläufer heutiger Foren dar. Es existieren zahlreiche Anwendungen zur Erstellung und Verwaltung derartiger Listen, beispielsweise *GNU Mailman*[16] oder *Google Groups*[17]. Die von den Programmen bereitgestellten Feeds können den gesamten Text oder nur Auszüge der Inhalte transportieren. Charakteristisch für Mailinglisten sind detektierbare Konversationsstrukturen wie sie in Abschnitt 2.5.2.2 geschildert werden.

Artefakt-Repositorien Insbesondere in der Domäne der Softwareentwicklung werden Repositorien intensiv eingesetzt, um beispielsweise den Quellcode eines Projektes zu verwalten. Derartige Werkzeuge werden unter dem Begriff *Versionsverwaltungssystem* zusammengefasst, bekannte Vertreter sind beispielsweise *SVN*[18] oder *Git*[19]. Die Art bzw. der Typ der zu verwaltenden Artefakte ist jedoch beliebig, eine Beschränkung auf eine konkrete Domäne existiert demnach nicht. Derartige Anwendungen stellen meist Feeds bereit, welche den Nutzer über neue oder modifizierte Artefakte informieren, sogenannte *Check-ins* oder *Commits*. Eine entsprechende Nachricht könnte beispielsweise lauten: „Revision 3384: doc fix“. Anhand sogenannter *Revisionsnummern* werden auch hier Änderungen an einzelnen Artefakten systemweit eindeutig identifiziert und stellen einen potentiellen Indikator für Relationen dar. Insbesondere innerhalb der Domäne der Softwareentwicklung stehen Repositorien eng mit den bereits erwähnten Issue-Tracking-Systemen in Zusammenhang: ein Fehler wird durch ein Ticket gemeldet bzw. erfasst und anschließend durch geänderte Artefakte in einem Check-in referenziert - derartige Ereignisketten werden in [49] im Detail beschrieben.

Foren (auch *Internet Forum*, *Message Board* oder *Webboard* genannt) sind ein weit verbreitetes Kommunikationsmittel für den asynchronen Informationsaustausch. Hierbei

14 hawaiisch für *schnell*
15 http://www.wikimatrix.org/ [26.10.11]
16 http://www.gnu.org/software/mailman [26.10.11]
17 http://groups.google.com [26.10.11]
18 http://subversion.tigris.org [26.10.11]
19 http://git-scm.com [26.10.11]

können Diskussionen durchgeführt werden, deren Inhalte an zentraler Stelle (dem Forum bzw. der entsprechende Webseite) allen Nutzern zur Verfügung gestellt werden. In Folge dessen sind im Gegensatz zu (beispielsweise) Mailinglisten keine externen Programme notwendig. Grundsätzlich ist ein Forum in Themen *(Threads)* und den darin enthaltenen Kommentaren *(Comments)* strukturiert und kann entsprechende Informationsströme zur Verfügung stellen. Im Bereich der Webforen-Software existieren sehr viele Anwendungen für die Realisierung entsprechender Systeme, beispielsweise das weit verbreitete *phpBB*[20] sowie *VBulletin*[21].

Blogs und Microblogs sind eine, als mittlerweile etablierte anzusehende, Form der Kommunikation innerhalb der Internets, bei welcher einer oder mehrere Autoren ein (meist) öffentliches Journal in Form einer Webseite führen. Meistens besitzen externe Nutzer die Möglichkeit, Kommentare zu den einzelnen Artikeln abgeben zu können. Diese Systeme können in den verschiedenen Domänen relevante Informationen liefern, beispielsweise innerhalb der Domäne der Software-Entwicklungsprojekte, bei welchen einzelne Entwickler individuelle Blogs betreiben um ggf. über zukünftige Entwicklungen zu berichten, welche noch nicht in den offiziellen Kanälen vertreten sind. Die Bereitstellung entsprechender Feeds ist im Kontext von Blogs fast als obligatorisch anzusehen. Die Nachrichtenstruktur entspricht im Wesentlichen der eines Forums, wobei jedoch die Kommentare in Abhängigkeit des Kontextes weniger gewichtet werden bzw. eine geringere Relevanz besitzen.

Neben den aufgezählten Typen existieren verschiedenartige weitere Formen, beispielsweise *Instant Messaging.* Eine wesentlich ausführlichere Beschreibung der einzelnen Tools und deren Zusammenwirken im Rahmen von FLOSS-Projekten wird in *Producing Open Source Software* [31] Kapitel *Technical Infrastructure* vorgestellt.

2.5 Relationen

Das grundsätzliche Ziel des zu konzipierenden Systems ist die Detektion von Relationen zwischen verschiedenartigen Nachrichten einer oder mehrerer Informationsquellen respektive Strömen. Der nachfolgende Abschnitt soll einen Überblick sowie eine Kategorisierung möglicher Beziehungen geben, welche durch geeignete Algorithmen erkannt werden können. Innerhalb der Domäne der Software-Entwicklungsprojekte wird in [19] ein ERD vorgestellt, welches bereits grundlegende Abhängigkeiten auf konzeptioneller Ebene illustriert (vgl. Abschnitt 2.7.4 Abbildung 5).

2.5.1 Modell

Prinzipiell kann eine Nachricht A zu beliebig vielen Nachrichten B in Beziehung stehen, da Feeds, bzw. die zugrunde liegende Systemsicht, stets einen zeitlichen Bezug haben, kann von gerichteten Relationen ausgegangen werden (eine Nachricht mit Zeitstempel n kann auf andere Nachrichten mit $m<n$ verweisen, jedoch nicht umgekehrt, da dies bedeuten würde, dass ältere Nachrichten nachträglich in Relation zu Beiträgen gesetzt werden, welche zum Zeitpunkt m gar nicht existent waren). Aus mathematischer Sicht kann diese Struktur mittels einem gerichteten azyklischen Graphen (DAG) abgebildet werden, bei welchem eine Kante *(A,B)* für die Relation *Nachricht A zeigt auf B* respek-

20 http://phpbb.com/ [26.10.11]

21 http://www.vbulletin.com/ [26.10.11]

tive *B ist der Elter von A*, steht.

$$G=(V,E)\ mit\ E:V\times V$$

Im Kontext des World Wide Web wird dies oft als sogenannter *Web Graph* [51] bezeichnet, im Rahmen des zu konzipierenden Systems ist die Bedeutung der Kanten jedoch wie bereits geschildert erweitert zu betrachten[22]. Die Knoten der Menge V entsprechen den einzelnen Nachrichten der Feeds, die gerichteten Kanten repräsentieren Beziehung zueinander. Aufgrund oben genannter Einschränkungen kann bzw. muss ausgeschlossen sein, dass sich in G Zyklen ergeben, da dies in einigen Anwendungsfällen zu fehlerhaften Zuständen führen würde[23]. Des Weiteren muss davon ausgegangen werden, dass Relationen nicht zwingend auf einen binären Zustandsraum (existent und nicht existent) abgebildet werden. Vielmehr können diese mit Wahrscheinlichkeiten ausgedrückt werden, dies betrifft beispielsweise Nachrichten, welche aufgrund von ähnlichen Inhalten in Beziehung zueinander stehen. Dieses Ähnlichkeitsmaß sollte innerhalb der Relationen erkennbar sein, wodurch sich innerhalb des Graphen gewichtete Kanten ergeben:

$$G=(V,E,f)\ mit\ \ f:E\rightarrow\mathbb{R}$$

Je größer ein Element aus f ist, desto wahrscheinlicher ist die zugrunde liegende Relation der Nachrichten. Damit eine bessere Vergleichbarkeit gewährleistet ist, sollten diese Werte normalisiert werden. Grundsätzlich können zwei Klassen von Beziehungen unterschieden werden: domänenabhängige und domänenunabhängige. Diese sollen in den folgenden Abschnitten detailliert erläutert werden.

2.5.2 Domänenunabhängige Relationen

Nachfolgend werden Nachrichtenrelationen beschrieben, welche unabhängig des konkreten Einsatzgebietes bzw. Kontextes als global gültig betrachtet werden können.

2.5.2.1 Web Graph

Das gesamte Word Wide Web stellt im Grunde genommen selbst einen Graphen dar, in welchem Hyperlinks als fundamentaler Bestandteil Kanten repräsentieren bzw. generieren [51]. Diese Tatsache spiegelt sich auch innerhalb der Nachrichten der Feeds wieder: enthält *Nachricht A* einen Hyperlink auf *Nachricht B*, so ist dies ein absoluter Indikator für eine Relation bzw. repräsentiert diese bereits. Dieser Umstand spiegelt sich auch in sogenannten *linktopologischen Ranking-Verfahren* [56] wieder, welche die Verlinkung einzelner Dokumente untereinander (bzw. den Grad dieser) als Indiz für deren Relevanz messen. Die Erkennung dieser an sich simplen Struktur wird jedoch aufgrund der Heterogenität von URLs erschwert, bereits einfache Unterschiede führen dazu, dass simple String-Vergleiche fehl schlagen. Um dieses Problem zu lösen, existieren Ansätze in Richtung einer sogenannten *URL-Normalisierung*, welche das gemeinsame Ziel haben, eine URL auf eine Normalform bzw. kanonische Form abzubilden, um so feststellen zu können, ob zwei syntaktisch unterschiedliche URLs semantisch äquivalent sind. Innerhalb

22 Eine Kante repräsentiert nicht zwingend einen Hyperlink sondern kann aufgrund verschiedenartiger Indikatoren existieren.

23 Beispielsweise kann ein Kommentar A zu einem Thread B nur durch die Relation $A \rightarrow B$ bzw. *(A,B)* abgebildet werden, jedoch niemals $B \rightarrow A$, da ein Thread eine notwendige Bedingung für einen untergeordneten Kommentar darstellt.

des *RFC 3986* [71] wird ebenfalls auf diese Problematik eingegangen und eine Rangliste von Vergleichsmethodiken präsentiert. Semantisch äquivalent bedeutet in diesem Kontext, dass die resultierende Ressource identisch ist, respektive zwei URLs das gleiche Ziel haben. Die Komplexität derartiger Normalisierer ist hierbei fließend, beginnend bei der simplen Transformation in Kleinbuchstaben oder Parametersortierung, bis hin zur Auswertung der referenzierten IP Adresse bzw. Ressource, existiert ein breites Spektrum an Möglichkeiten. Diese können grundsätzlich in zwei Kategorien eingeteilt werden: jene welche die Semantik einer URL erhalten und jene, welche diese ändern. Letztere setzen auf De-facto-Standards bzw. auf Heuristiken, um den Normalisierungsprozess zu verbessern, mit dem Risiko, dass unter Umständen zwei semantisch unterschiedliche URLs als äquivalent gekennzeichnet werden (falsch-positiv-Ergebnisse). In ihrer Arbeit „Do not crawl in the DUST“ [9] präsentieren die Autoren weiterführende Ansätze zur Detektion semantisch äquivalenter URLs.

2.5.2.2 Thread Detection

Innerhalb von Mailinglisten bzw. Foren ergeben sich durch die Frage-Antwort-Schemata entsprechende Konversationsstrukturen, welche in den Nachrichten detektiert werden können. In [57] werden diese Strukturen im Kontext von E-Mail-Anwendungen charakterisiert und im Detail erläutert. Nachricht *C* könnte beispielsweise eine Antwort auf Nachricht *B* darstellen, welche wiederum ein Kommentar zu Nachricht *A* ist, wodurch sich die Relationen *(C,B)* und *(B,A)* ergeben. Um derartige Verweise abbilden zu können, existiert innerhalb des E-Mail-Headers das sogenannte *In-Reply-To*-Feld, welches auf eine eindeutige *Message-ID* verweist, dies ist innerhalb des RFC 5322 [74] *Internet Message Format* definiert. Diese Informationen stehen aber innerhalb der Feeds, welche die Nachrichten (in diesem Fall Inhalte der E-Mails) transportieren, nicht mehr zur Verfügung. Des Weiteren existieren beispielsweise für Foren derartige Standards nicht. Es existieren jedoch bereits verschiedenartige Ansätze, welche gemein hin unter dem Begriff *Thread Detection* klassifiziert werden können, die darauf abzielen, Konversationen anhand unterschiedlicher Merkmale extrahieren zu können. *Erera* und *Carmel* analysieren in ihrer Arbeit „Conversation detection in email systems“ [24] entsprechende Methodiken und State-of-the-Art-Verfahren. Des Weiteren beschreibt der RFC 5322 eine Syntax für Textnachrichten innerhalb der Domäne der E-Mail-Kommunikation. Jedoch werden in diesem keine präzisen Definitionen geliefert, sondern vielmehr wie Konversationen innerhalb von Nachrichtenströmen anhand von gegebenen Attributen (der Nachrichten) detektiert werden können. In [57] wurde untersucht, welche Merkmale innerhalb von E-Mail-Nachrichten nutzbar sind, um Threads detektieren zu können. Die Analysen zeigten, dass die Suche nach zitierten Bereichen im Korpus der Originaltexte die besten Ergebnisse ergab.

Zitate Wie innerhalb des RFC 2822 beschrieben wird, fügen die meisten E-Mail-Clients die ursprüngliche Nachricht an eine Antwort, in entsprechende Bereiche bzw. mittels definierter Zeichen, an. Diese zitierten Abschnitte *(Quoted Text)* können extrahiert und gegen anderen Nachrichten *(Unquoted Text)* verglichen werden [57]. Problematisch hierbei ist jedoch unter anderem, dass Antworten nicht zwingend die Elternachricht als Zitat enthalten müssen. Des Weiteren stellen sehr kurze Zitate insofern ein Problem dar, als dass bei diesen die Wahrscheinlichkeit steigt, dass mehrere Nachrichten diese Zeichenkette enthalten und somit nur anhand dieses Merkmals nicht eindeutig entschieden werden kann, welche dieser der Elter ist. Zur Suche, respektive Vergleich, von *Quoted-*

und *Unquoted*-Zeichenketten kann beispielsweise das in Abschnitt 2.3 erwähnte Vektorraummodell [61] eingesetzt werden.

Titel/Betreff Obwohl es keinem Standard entspricht bzw. keine Voraussetzung darstellt, hat sich insbesondere durch den Einsatz entsprechender E-Mail-Clients die Verwendung von Abkürzungen im Titel einer Nachricht etabliert. Der Titel einer Antwort wird hierbei mit dem Präfix *„RE:“ (response)* versehen, wodurch sich bei mehreren Antworten Kaskaden dieser Abkürzung ergeben. Unter [58] werden weitere Ausprägungen dieser Präfixe und deren Sprachvarianten vorstellt. In [24] wird in diesem Kontext der Begriff des sogenannten *core* [*subject*] verwendet: Nachrichten bilden hierbei einen sogenannten *Subject Thread*, wenn diese einen identischen Titel haben, nachdem die geschilderten Abkürzungen entfernt wurden. Eine Alternative besteht darin, nicht nach exakten Übereinstimmungen des Titels bzw. der Zeichenketten zu suchen, sondern auch nach ähnlichen Formulierungen. In [24] wird dies mittels des sogenannten *Dice-Koeffizienten* realisiert:

$$subj(e_i, e_j) = \frac{2|S_i \cap S_j|}{|S_i| + |S_j|}$$

Die Ähnlichkeit der Titel (*subj*) der Nachrichten e_i und e_j berechnet sich hierbei aus deren jeweiligen Wortmengen S_i und S_j.

Autoren In [32] wird als weiteres Attribut die Liste der Empfänger einer Nachricht ausgewertet und als ein wesentliches Merkmal einer Konversation angesehen, unter der Annahme, dass eine entsprechende Gruppe von Personen identifizierbar ist. Schreibt Nutzer *A* eine Nachricht an Nutzer *B, C* und *D,* dann kann dieses Muster als ein weiteres Merkmal einer Relation untersucht werden. Innerhalb von Feeds sowie insbesondere bei der Verwendung von Mailinglisten existieren jedoch prinzipiell keine Empfänger (dies ist stets die Adresse der Mailingliste).

URL Viele Webanwendungen generieren Themen- und Kommentar-URLs in einer relativ eindeutigen bzw. analysierbaren Weise, beispielsweise der Form: *http://forum.com/thread/6dff81/show_ docId=7dce.* Entsprechende Kenntnisse vorausgesetzt, ist es somit möglich, Kommentare und Themen korrekt zu strukturieren. Problematisch ist jedoch, dass hierfür keinerlei Standards existieren und eine Identifizierbarkeit nicht generell möglich ist bzw. stark domänen- bzw. systemabhängig ist.

Grundsätzlich kann innerhalb der Thread-Detection-Analysen zwischen rein strukturbasierenden und tendenziell stärker auf Semantik-beruhenden Verfahren unterschieden werden. Beispielsweise existieren Ansätze, welche thematisch zusammengehörige Nachrichten zu Gruppen zusammenfassen können *(clustering)* [79], indem stochastische Wort- und Satzanalysen angewendet werden. Im Gegensatz zu rein strukturbasierenden Gruppierungen (wie beispielsweise mittels den erwähnten Analysen von zitierten Bereichen) ergeben sich auf diese Weise Nachrichtengruppen, welche erst durch die Analysen entstehen und vorher als solche nicht existent waren.

2.5.2.3 Textähnlichkeit

Mittels der in Abschnitt 2.3 geschilderten Methodiken kann für jede Nachricht *A* eine zu ihr maximal ähnliche Nachricht *B* ermittelt werden. Hierbei wird der gesamte Text bzw.

Inhalt von *A* als Anfrage *(Query*) gegen den Korpus definiert und ein Ähnlichkeitsmaß *(Score)* ermittelt, dieses Vorgehen wird beispielsweise in [19] und [24] angewendet. Abgesehen vom Spezialfall, dass ein derartiger Vergleich kein Ergebnis zurück liefert, ergibt sich für eine Relation *(A,B)* somit eine nach Relevanz sortierbare Menge *M* von Dokumenten, auch *top-k*-Liste genannt. Der Wert des *Scores* entspricht somit dem Wert der gewichteten Kante zwischen *A* und *B* bzw. fließt in diese ein. Um derartige Vergleiche zu verbessern, ist der Einsatz von Stoppwort-Algorithmen obligatorisch, da ansonsten Begriffe mit hoher Relevanz in der Menge irrelevanter Zeichenketten untergehen und in Folge dessen, an sich wenig aussagekräftige Texte zu hoch gewichtet würden. Eine weitere Erhöhung der Genauigkeit kann durch die in Abschnitt 2.3.1 beschriebenen *Stemming*- und *Lemmatizer*-Algorithmen erreicht werden. Mit zunehmender Intensität derartiger Prozesse kann eine Nachricht durch eine wesentlich kompaktere Anzahl von extrahierten Schlüsselwörtern charakterisiert werden.

Part-of-speech-Tagging-Mechanismen können die Menge der relevanten Wörter weiter eingrenzen, indem die zuvor ermittelten Wörter (oder auch Wortstämme) basierend auf deren Wortart weiter gefiltert werden. POS-Tagger annotieren hierbei jedes Wort eines Satzes mit dessen Wortart, der Satz „This is a sample sentence" wird demnach beispielsweise durch „This/DT is/VBZ a/DT sample/NN sentence/NN" dargestellt. Für die Abbildung der grammatikalischen Konstrukte wird (bezogen auf die Englische Sprache) meist das sogenannte *Penn Treebank Tag Set* [84] verwendet. Grundsätzlich können zwei verschiedene Typen von POS-Taggern unterschieden werden: *überwachte* und *nicht überwachte* Systeme. Erstere arbeiten mit einem Korpus von bereits korrekt annotierten Wörter bzw. Sätzen, wohingegen letztere lediglich mit den vorhandenen Daten und stochastischen Ansätzen, Gruppen von Wörtern versuchen zu bilden, wobei die genaue Bedeutung einer Gruppe dem System jedoch nicht bekannt ist. Überwachte POS-Tagger arbeiten prinzipiell ähnlich, wobei jedoch durch den zur Verfügung stehenden Datensatz die Wahrscheinlichkeiten von spezifischen Abfolgen der Wortarten bekannt sind und in Folge dessen, bei der Analyse gegen diese verglichen und entschieden werden kann. Die genaue technische Realisierung kann hierbei mit verschiedenen theoretischen Ansätzen erreicht werden, beispielsweise mittels *Hidden-Markov-Modellen* oder Clustering-Algorithmen. Eine Liste aktueller POS-Tagger und deren Algorithmen findet sich unter [68]. Beim Vergleich der Textähnlichkeit kann somit das POS-Tagging zum Einsatz kommen, um beispielsweise Verben zu entfernen und so das Rauschen weiter minimieren zu können.

Keyword Extraction bezeichnet Verfahren, welche die relevanten Wörter eines gegebenen Textes zu extrahieren versuchen. *Relevant* bedeutet in diesem Zusammenhang, dass diese das Dokument möglichst gut charakterisieren bzw. zusammenfassen. Unter [78] werden drei grundlegende *Keyword-Extraction*-Typen vorgestellt, welche oft auch gleichzeitig eingesetzt werden:

- Einfache statistische Ansätze, welche keine Trainingsphase benötigen und sich auf nicht-linguistische-Merkmale eines Textes stützen, beispielsweise die Termfrequenz (vgl. Abschnitt 2.3.1) oder die Position eines Schlüsselwortes. Für viele Anwendungen sind derartige Ansätze bereits ausreichend.
- Linguistische Ansätze, welche eng im Zusammenhang mit den bereits erwähnten *POS-Taggern* stehen, versuchen neben den statistischen Merkmalen auch den

grammatikalischen Typ eines Wortes zu berücksichtigen. Beispielsweise sind Substantive bzw. Nomen meist relevanter als Verben.

- *Machine Learning* Ansätze, welche in die Kategorie der überwachten Algorithmen fallen. Diese versuchen mittels getaggten Datensätzen Schlüsselwörter zu extrahieren, welche unter Umständen auch abstrahiert werden können. Beispielsweise indem einem Dokument mit den Wörtern *table* und *desk* nicht nur diese beiden Begriffe als Schlüsselwörter zugeordnet werden, sondern auch *furniture.* Rein statistische Ansätze können dies nicht leisten.

Fazit Um die Textähnlichkeit einzelner Nachrichten vergleichen zu können, existieren sehr viele verschiedenartige Ansätze. Die hier vorgestellten Methoden fallen jedoch tendenziell eher in den Bereich der Vorverarbeitung, da sie nicht den eigentlichen Vergleich zum Ziel haben. Dieser kann beispielsweise mit den in Abschnitt 2.3 beschriebenen Methoden realisiert werden.

2.5.2.4 Sozialer Graph und Aliaserkennung

Abgesehen von automatisiert generierten Systemnachrichten kann davon ausgegangen werden, dass jede Nachricht einem (teilweise auch mehreren) Autor[en] zugeordnet werden kann. Nachrichten stehen demnach in einer weiteren Ebene in Relation zueinander, sofern sie von der selben Person verfasst wurden [32]. Problematisch ist hierbei jedoch, dass die Identitäten oft nicht systemweit eindeutig sind, in dem Sinne, dass eine Person oft mehrere verschiedene Adressen bzw. Aliase verwendet, welche aber semantisch einer Identität entsprechen. Es existieren verschiedenartige Methoden, um dennoch syntaktisch unterschiedliche Aliase auf einheitliche Identitäten (Personen) abbilden zu können. Dies wird unter dem Begriff **Alias Detection** [46] zusammengefasst und mittels unterschiedlicher Ansätze versucht zu realisieren. Über Motivationen seitens der Nutzer, sich mehrere Identitäten zu generieren sowie seitens der (beispielsweise) Administratoren, diese dennoch zu erkennen, soll an dieser Stelle nicht weiter diskutiert werden. Die *Stilometrie* ist eine Teilgebiet der Linguistik, welches mit Methodiken aus dem Bereich der Statistik Untersuchungen zum Sprachstil durchführt. Große Bedeutung haben derartige Verfahren im Bereich der forensischen Linguistik, um beispielsweise anonyme Schriftstücke einem potentiellen Autor zuordnen zu können. Im Kontext der Arbeit können entsprechende Mechanismen zum Einsatz kommen, um aus einer Menge M von Nachrichten eine Teilmenge N zu extrahieren, welche trotz unterschiedlicher Autorenbezeichner (wahrscheinlich) einen identischen Urheber besitzen. Ziel der Analyse eines Textes ist die Ermittlung einer sogenannten *Writer Invariante*, also Mengen von Merkmalen, welche unabhängig des Textes und dessen Domäne invariant, bezogen auf den jeweiligen Autor, sind. Welche Eigenschaften bzw. Merkmale dies im einzelnen sind, ist jedoch nicht klar definiert bzw. Forschungsgegenstand. In Frage kommen hierbei unter anderem die mittlere Satzlänge, Wortlängen, Frequenzen von spezifischen und universellen Substantiven und vieles mehr. Bezogen auf Inhalte im Bereich des World Wide Webs ergeben sich zudem noch zahlreiche weitere Merkmale: Punktationen, Quellcodeformatierungen, Zeitstempel, HTML-Tags und vieles mehr. Voraussetzung für derartige Analysen ist jedoch eine ausreichend Große Datenmenge (Summe der Wörter aller Nachrichten pro potentiellem Autor), da mit abnehmenden Umfang eines Textes die Wahrscheinlichkeit sinkt, aussagekräftige Merkmale *(Writer Invariante)* extrahieren zu können. Frühere Arbeiten gehen von einem Schwellenwert von 6.500 Wörtern aus, neuere Ansätze [66]

von circa der Hälfte (3.000 Wörter).

In vielen Informationsquellen kann davon ausgegangen werden, dass einzelne Nutzer (Autoren) miteinander interagieren und in Folge dessen, auch auf dieser Ebene Hinweise für mögliche Relationen generieren können. Aus dem Bereich der sozialen Netzwerke sind beispielsweise zahlreiche Analysen bekannt [90], welche auch in der Domäne des zu konzipierenden Systems von Relevanz sind. Durch Art und Intensität der Interaktionen der einzelnen Nutzer untereinander, kann ein sogenanntes *Soziales Netzwerk* (**Social Graph)** konstruiert werden, welches für sich genommen jedoch keine eigenständigen Relationen (bezogen auf die eigentlichen Nachrichtenentitäten) generieren kann. Ist beispielsweise bekannt, dass Nutzer *A* sehr oft mit Nutzer *B* kollaborativ arbeitet, wird dies durch eine entsprechende gewichtet Kante im Graphen abgebildet. Des Weiteren repräsentiert der Knotengrad *(social degree)* auch die Signifikanz eines Nutzers innerhalb eines Systems, da davon ausgegangen werden kann, dass ein sehr aktiver Nutzer zwangsläufig intensiv interagiert, respektive Beziehungen hält. Ein hoher Knotengrad kann demnach genutzt werden, um eine existierende Relation höher zu gewichten, unter der Annahme, dass beispielsweise Administratoren sehr aktiv in verschiedenen Bereichen sind. Diese besitzen in Folge dessen eine sogenannte *Intermediationszentralität* (*betweenness centrality,* dies bedeutet, dass ein Nutzer häufig als Verbindungsglied zwischen Akteuren fungiert, beispielsweise als Moderator).

2.5.2.5 Zeitliche Korrelationen

Ausnahmslos alle Nachrichten eines Korpus können mittels einer zeitlichen Ordnungsrelation sortiert bzw. analysiert werden. Die temporale Dimension stellt somit einen wesentlichen Faktor für zahlreiche Relationen dar, beispielsweise wird in [24] nach Konversationen[24] in E-Mail-Nachrichten gesucht und hierbei ein maximaler Zeitabstand analysiert: behandeln zwei unterschiedliche Nachrichten ähnliche Themen, bilden sie dennoch keine Konversationen, sofern sie zeitlich einen zu großen Abstand zueinander haben. Dieser Schwellenwert kann nicht generell festgelegt werden und ist demnach stark von der jeweiligen Domäne abhängig. Um einen vergleichbaren und normalisierten Wert zu erhalten, wird in [24] folgende Berechnungsvorschrift eingesetzt:

$$date(e_i, e_j) = 1 - min(1, \frac{|d_i - d_j|}{mdf})$$

Das Maß der temporalen Ähnlichkeit zweier Nachrichten d_i und d_j berechnet sich demnach aus der Differenz derer Zeitstempel, dividiert durch einen festzulegenden Schwellenwert *mdf* (*max date difference*). Mittels der Minimalfunktion ergibt sich in Folge dessen ein normalisierter Vergleichswert. Die Verwendung einer derartigen Berechnungsvorschrift hat des Weiteren den Vorteil, dass sie neutral bezüglich der eingesetzten Granularität (Sekunde, Stunden etc.) ist. In [19] wird der inverse Fall beschrieben, bei welchem thematisch und strukturell unterschiedliche Nachrichten dennoch auf einer zeitlichen Ebene in Relation zueinander stehen, sofern sie innerhalb eines definierten Zeitfensters auftraten (also einen maximalen Zeitabstand nicht überschreiten) und ggf. weitere Bedienungen erfüllen (beispielsweise indem sie durch identische Autoren verfasst wurden).

24 Unter Konversationen wird hierbei eine Menge an Nachrichten verstanden, die trotz unterschiedlicher Titel und/oder Autoren die gleiche Thematik behandeln.

2.5.3 Domänenabhängige Relationen

Nachfolgend werden Nachrichtenrelationen beschrieben, welche abhängig von konkreten Einsatzgebieten bzw. Kontexten sind. In Folge dessen sind diese ohne zusätzliche Informationen nicht beliebig auf jede Domäne übertragbar, sondern müssen je nach Einsatzgebiet entsprechend parametrisiert bzw. konfiguriert werden.

2.5.3.1 Named Entitiy Recognition

In Abhängigkeit der Domäne können Muster und Bezeichner existieren, welche auf Beziehungen hinweisen können. Innerhalb von Software-Entwicklungsprojekten werden beispielsweise oft sogenannte Bug-IDs mittels „Bug #123456“ oder ähnlichen Formulierungen hinterlegt. Treten diese in mehreren Nachrichten auf, so stehen diese bezogen auf diesen Identifikator in Relation zueinander. Problematisch ist jedoch eine vollständige Automatisierung der Erkennung, da beispielsweise nicht davon ausgegangen werden kann, dass lediglich numerische Werte auftreten. Exemplarisch könnte ein Verweis lauten „siehe Aktenzeichen 12 C 580/06“, derartige Strukturen werden oftmals mittels regulären Ausdrücken beschrieben. Ein anderer Ansatz besteht darin, Identifikatoren aus Texten mittels einer Analyse der Aussprechbarkeit der einzelnen Zeichenketten zu extrahieren. Dies ist jedoch insofern schwierig, da verschiedene Sprachen hierfür sehr unterschiedliche Maße besitzen. Ein naiver Ansatz könnte beispielsweise der Zeichenkette *wawrzyniec* einen sehr geringen Wert zuweisen und somit als Identifikator klassifizieren - jedoch stellt die genannte Zeichenkette einen polnischen Vornamen dar. Des Weiteren sind bei Textauszügen, welche keine natürliche Sprache darstellen (beispielsweise Codebereiche), sehr viele Zeichenketten enthalten welche weder Wörter noch Identifikatoren darstellen. Die Analyse und Erkennung derartiger Muster und Bezeichner fällt in das Fachgebiet des sogenannte *Named Entitiy Recognition*, wobei der Begriff *Entität* hierbei weiter gefasst zu verstehen ist, beispielsweise sollen auch Personennamen erkannt werden. [53] beschreibt dies wie folgt:

> *„Named entity recognition (NER) is a subtask of information extraction that seeks to locate and classify atomic elements in text into predefined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc. “*

Ein natürlichsprachlicher Text wie beispielsweise „Jim bought 300 shares of Acme Corp. In 2006.“ kann mittels eines NER-Systems wie folgt annotiert werden:

```
1  <ENAMEX TYPE="PERSON">Jim</ENAMEX> bought <NUMEX TYPE="QUANTITY">300</NUMEX>
2  shares of <ENAMEX TYPE="ORGANIZATION">Acme Corp.</ENAMEX> in
3  <TIME TYPE="DATE">2006</TIMEX>
```

Listing 2: Beispiel einer NER-Annotierung

Die auf diese Weise extrahieren Entitäten können anschließend genutzt werden, um Beziehungen (bei gleichen oder ähnlichen Vorkommen) zwischen Nachrichten herstellen zu können. Es existieren einige frei verfügbare Bibliotheken, welche NER Funktionalitäten bereitstellen, beispielsweise *Apache OpenNLP*[25] sowie *Stanford NER*[26].

Reguläre Ausdrücke stellen eine weitere Möglichkeit dar, spezifische Muster innerhalb von Nachrichtentexten extrahieren zu können. Im Gegensatz zu den geschilderten NER-Prozessen ist bei diesen eine detaillierte Konfiguration bzw. Eingaben in ein System notwendig. Der Einsatz von Regulären Ausdrücken (oft auch *RegEx* genannt) erlaubt jedoch eine wesentlich spezifischere und genauere Definitionsbasis, welche mittels NER nicht zu erreichen ist. Die im Abschnitt 2.4 vorgestellten Issue-Tracking-Systeme setzen beispielsweise komplexe Identifikationsnummern bzw. Muster ein, welche exemplarisch mittels des Ausdrucks „(bug|patch|item) ?(report){0,1} ?#?[0-9]{3,}“ detektiert werden können.

2.5.3.2 Interaktions- und Ereignismuster

Unabhängig der in den vorangegangen Abschnitten vorgestellten Beziehungsmerkmale zwischen Nachrichten, existiert in jeder Domäne eine mehr oder weniger klar definierte Menge an verbalen Beschreibungen, welche spezifische Charakteristika eines konkreten Kontextes definieren. In der Domäne der Software-Entwicklungsprojekte könnte ein entsprechendes Muster[27] beispielsweise lauten:

> *„Wenn ein Nutzer eine Datei in das Versionsverwaltungssystem einfügt und innerhalb einer definierten Zeitspanne einen Fehler im Issue-Tracking-Systems als abgeschlossen markiert, so stehen beide Ergebnisse mit hoher Wahrscheinlichkeit in Beziehungen zueinander“.*

Exemplarisch soll ein weiteres (triviales) Szenario erläutert werden: das Ereignis A „es regnet“ gefolgt von dem Ereignis B „es ist nass“, steht für ein ungelerntes System ohne zusätzliches Vorwissen in keiner Beziehung zueinander, da es keine Information über die Semantik besitzt. Damit ein System in der Lage ist, zu erkennen, dass A stets B impliziert, respektive A und B in Beziehung zueinander stehen, existieren konzeptionell zwei Möglichkeiten:

- Mittels Methodiken des sogenannten *Machine Learning* lernt das System mit der Zeit selbstständig, dass A und B in Relation zueinander stehen. Hierfür können händische Annotationen sowie statistische Analysen eingesetzt werden.
- Dem System wird dieses Wissen explizit, mittels einer maschinenlesbaren Darstellung, zur Verfügung gestellt.

Die erst genannte Variante ist mit zunehmender Komplexität, respektive Umfang, der Nachrichten (welche Ereignisse darstellen) immer aufwändiger bzw. müssen entsprechende Algorithmen Unterschiede und Gemeinsamkeiten selbstständig extrahieren können. Die Wahrscheinlichkeit von falsch-positiv-Ergebnissen (vgl. Abschnitt 2.3.2) ist hierbei ggf. sehr hoch. Die zweite Möglichkeit erfordert im Vergleich dazu jedoch die

25 http://incubator.apache.org/opennlp/index.html [13.09.11]

26 http://nlp.stanford.edu/ner/index.shtml [13.09.11]

27 Hierbei ist nicht der Begriff *Ereignismuster* aus dem Bereich der Wirtschaftssoziologie gemeint.

explizite Formulierung von Ereignismustern, mit zunehmender Komplexität dieser sind somit auch entsprechende fachliche Kenntnisse erforderlich. Innerhalb des Abschnitts 2.6 *Event Processing* wird detaillierter auf diesen Umstand eingegangen. Betrachtet man komplexere Ereignismuster[28] konkreter Domänen, so wird deutlich, dass eine automatisierte Erkennung dieser äußert schwierig ist. Grundsätzlich besitzen die vorgestellten Varianten auch eine andere Motivation: in vielen Domänen ist das hohe Aufkommen von Nachrichten bzw. Events ausschlaggebend dafür, dass eine händische Filterung (und somit Annotierung) gerade nicht erfolgen kann und soll, vielmehr sollen definierte Ereignismuster korrekt und performant detektiert werden können. Sind Ereignismuster jedoch gar nicht bekannt und erst durch Nutzer händisch zu analysieren, versagen derartige Systeme und die erst genannte Variante ist einzusetzen. Eine weitere Möglichkeit existiert durch den Einsatz von Ontologien, welche detaillierte Entitäten und deren Relationen bzw. mögliche Interaktionen beschreiben können. Im Gegensatz zur konkreten Formulierung von Ereignisketten, sind diese jedoch statisch, das heißt, eine zeitliche Dimension ist nicht vorhanden. Es existieren jedoch Ansätze, welche die zeitliche Dimension in Ontologien integriert [7]. Grundsätzlich ist die Formulierung einer umfangreichen Ontologie einer Domäne insbesondere in Hinblick auf zeitliche Aspekte entsprechender Ereignisketten als sehr komplex einzustufen.

2.6 Event Processing

Event Processing ist ein Teilgebiet der Informatik, welches sich mit Detektion, Analyse, Gruppierung sowie Verarbeitung von sogenannten *Events* (im Kontext der Arbeit entspricht dies einzelnen Nachrichten) innerhalb von komplexen Systemlandschaften beschäftigt. Eine konkrete Anwendung finden derartiger Prinzipien beispielsweise in sogenannten *Event Driven Architectures* [14], welche insbesondere im Kontext von Unternehmensanwendungen von hoher Relevanz sind. Große Bedeutung haben derartige Systeme weiterhin innerhalb der Netzwerkanalyse und Diagnose [41], um beispielsweise mittels einer sogenannten *Root Cause Analyses* die Ursache aufgetretener Fehler innerhalb von komplexen Komponenten detektieren zu können. Ein wesentliches Merkmal entsprechender Systeme besteht darin, dass die Verarbeitung in Echtzeit erfolgt, das heißt, es werden (beispielsweise) im Gegensatz zu Information-Retrieval-Systemen nicht erst Daten gesammelt, um diese „rückwirkend“ innerhalb von Datenbanken durchsuchen zu können – stattdessen werden Analysen, Verarbeitungen und Entscheidungen kontinuierlich, bereits während des Auftretens der Events, getroffen (Echtzeitverarbeitung).

28 Unter [18] werden komplexere Szenarien und mögliche Ereignismuster vorgestellt.

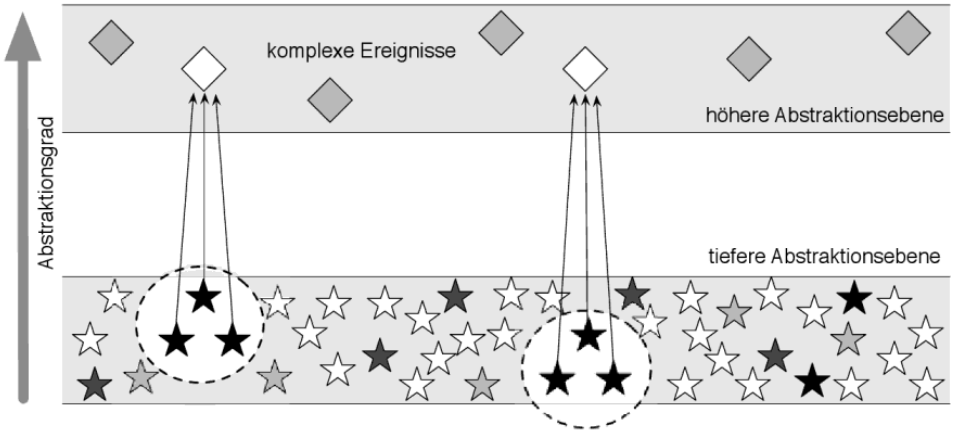


Abbildung 3: Mustererkennung und komplexe Ereignisse [14]

Die grundlegende Zielsetzung von *Event-Processing-Systemen* besteht darin, Muster innerhalb der atomaren Ereignissen, respektive deren Strömen, zu detektieren, um auf diese Weise vorher nicht existentes implizites Wissen bzw. Daten aus diesen generieren zu können (sogenannte *komplexe Ereignisse*), welche (normalerweise) aufgrund der Masse der Daten nicht händisch erkannt worden wären, siehe Abbildung 3. In diesem Sinne erfolgt des Weiteren eine Abstraktion der Nachrichtenelemente (Events), das heißt, es erfolgt eine Aggregation auf bzw. zu einer höheren Ebene, beispielsweise indem menschenlesbare Inhalte generiert werden. Dies unterscheidet die hier beschriebenen Systeme von einfachen ereignisgesteuerten Anwendungen, beispielsweise die asynchrone Verarbeitung von Nutzereingaben innerhalb von Benutzeroberflächen: bei diesen wird lediglich auf einzelne Events mit entsprechenden Handlungen reagiert, es erfolgt in diesem Kontext jedoch keine Analyse bzw. Mustererkennung.

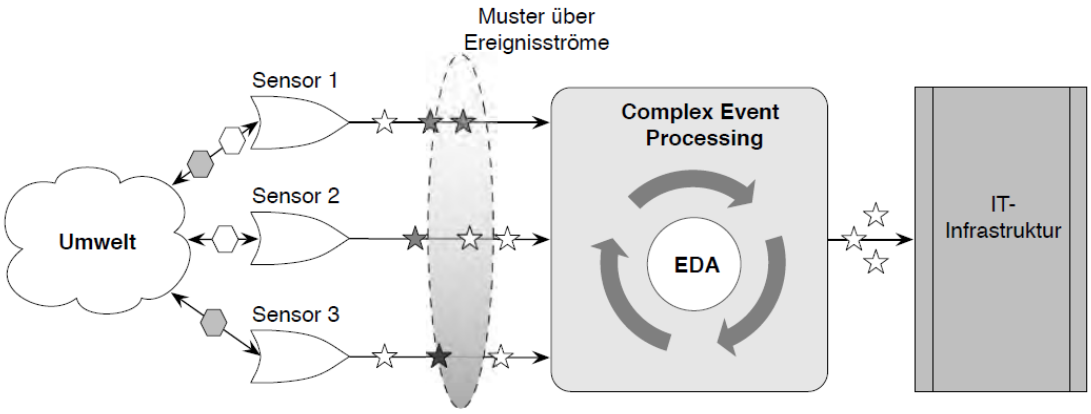


Abbildung 4: Abstrakte Darstellung eines Event-Processing-Systems [14]

In Abbildung 4 sind die beschriebenen Sachverhalte nochmals im Kontext einer gesamten Architektur grafisch dargestellt. Der Begriff *Event Processing* ist jedoch eine tendenziell abstraktere Bezeichnung und steht als Sammelbegriff für eine Reihe speziellerer Ausprägungen, welche nachfolgend charakterisiert werden sollen.

2.6.1 Events

Innerhalb von Event-Processing-Systemen steht der Begriff *Event* einerseits für nahezu alles „was passiert, oder als passiert angesehen wird“ sowie innerhalb der verarbeitenden Systeme als „ein Objekt, das ein Ereignis repräsentiert, kodiert oder speichert; für gewöhnlich zum Zwecke der maschinenbasierten Verarbeitung“ [59]. Innerhalb des zu konzipierenden Systems steht der Begriff *Event* nachfolgend somit stets für eine konkrete Nachricht eines Feeds bzw. Nachrichtenstromes. Hierbei kann es erforderlich bzw. gewünscht sein, nicht nur die real aufgetretenen Ereignisse zu emittieren, sondern dass auch sogenannte *virtuelle Events* generiert werden. In [59] wird dies als ein "Ereignis, das nicht tatsächlich in der physikalischen Welt auftritt, sondern erscheint, um ein Ereignis in der wirklichen Welt anzudeuten. Ein Ereignis, das man sich vorstellt, modelliert oder simuliert." charakterisiert. Derartige Ereignisse werden anschließend transparent (im Sinne eines realen Events) von den Anwendungen weiterverarbeitet.

2.6.2 Arten

Der Begriff Event Processing wird meist als Oberbegriff für konkretere Ausprägungen verstanden, diese sollen nachfolgend erläutert werden.

2.6.2.1 Simple Event Processing

Wie der Name bereits andeutet, handelt es sich bei derartigen Systemen um einfache ereignisgesteuerte Anwendungen, bei welchen ein Event direkt konkrete Prozesse initiiert, ohne dass vorher Analysen wie beispielsweise Mustererkennungen durchgeführt werden. In diesem Sinne repräsentiert ein Ereignis gewissermaßen eine relevante Zustandsänderung der zugrunde liegenden Nachrichtenquelle. Ein einfaches Beispiel stellen sogenannte *Notificator Applets*[29] dar, welche beim Eintreffen von E-Mails eine audiovisuelle Information innerhalb der Benutzeroberfläche generieren. Es erfolgt hierbei keine Analyse ob oder welche Nachrichten aufgetreten sind. Auch beispielsweise der *Java Message Service*[30] stellt eine SEP-Architektur dar. Grundsätzlich können SEP-Systeme demnach zustandslos arbeiten, da keine Speicherung von Events erforderlich ist.

2.6.2.2 Stream Event Processing

Stream-Event-Processing-Systeme (*Verarbeitung von Ereignisströmen,* auch *Event-Stream-Processing - ESP genannt*) beschreiben im Gegensatz zu SEP-Systemen eine kontinuierliche (daher *Stream*) Bearbeitung der eingehenden Nachrichten. Im Vergleich zu den bereits erwähnten SEP-Systemen ist bei der Verarbeitung der Nachrichten nicht jede einzelne Nachricht von Relevanz, sondern erst der Strom als Ganzes. Konkret bedeutet dies, das beispielsweise innerhalb von Temperaturmessungen im Kontext von Produktionsanlagen (oder auch der CPU eines Rechnersystems) nicht der atomare Wert (Temperatur zum Zeitpunkt t) entscheidend ist, sondern eine Reaktion erst erfolgt, wenn mehrere Werte einen gewissen Schwellenwert über einen definierten Zeitraum überschritten haben. Ein weiteres Beispiel stellen Börsenkurse dar, bei welchen nicht die Veränderung einzelner Werte von Interesse ist, sondern die Detektion von Trends, also die längerfristige Gruppierung von Daten. Ein weiterer Unterschied im Vergleich zu SEP-Systemen besteht somit darin, dass ESP-Systeme zustandsbehaftet sind und demnach mehrere

29 Beispielsweise https://launchpad.net/indicator-applet [20.11.11]

30 http://www.jcp.org/en/jsr/detail?id=914 [20.11.11]

atomare Events bzw. deren Werte gespeichert werden.

2.6.2.3 Complex Event Processing

Complex Event Processing (CEP) charakterisiert Systeme mit Eigenschaften, wie sie bereits innerhalb des übergeordneten Abschnitts erläutert wurden: Ziel ist die Detektion von komplexen Mustern innerhalb einzelner (oder auch zwischen verschiedenen) Ereignisströme. Dies markiert auch den wesentlichen Unterschied zu ESP-Systemen, bei welchen keine übergreifenden Muster erkannt werden (sollen), sondern lediglich eine einfache Analyse (oft Summierung oder Mittlung) erfolgt. Innerhalb von CEP-Systemen werden dagegen multiple Ereignisse, welche in einem ursächlichen, räumlichen oder temporalen Zusammenhang stehen, zu neuen Daten oder Prozessen aggregiert. Eine notwendige Voraussetzung ist in Folge dessen, dass das System Kenntnis über entsprechende Strukturen bzw. Muster hat, welche zumindest partiell manuell integriert werden müssen, um eine Identifikation vornehmen zu können. Die Definition derartiger Regelwerke (Liste von deklarativen Beschreibungen von Mustern mittels einer geeigneten Beschreibungssprache) für komplexe ereignisgesteuerte Systeme, erfordert meist fundierte Kenntnisse über die jeweiligen Domänen und stellt auch aus technischer Sicht (Fähigkeiten des Domänenexperten) eine anspruchsvolle Aufgabe dar. Des Weiteren sind meist auch die technischen Anforderungen an die zugrunde liegenden Anwendungen hoch, insbesondere im Hinblick auf Domänen, welche Ereignisströme mit einem sehr hohem Aufkommen von Nachrichten vorweisen. Ein weiteres Abgrenzungsmerkmal im Vergleich zu ESP-Systemen besteht in der Tatsache, dass einzelne atomare Ereignisse eventuelle gar keine oder nur eine sehr niedrige Relevanz im Einzelnen aufweisen, jedoch in spezifischen Kontexten als Indiz für signifikante Ereignisse gelten können. Dieser Umstand existiert in ESP-Systemen in dieser Form nicht. Logischerweise arbeiten CEP-Systeme in Folge dessen zwingend zustandsbehaftet, da einzelne Events durchaus über sehr lange Zeiträume in Beziehung zu aktuellen Ereignissen stehen können.

2.6.3 Systeme

In der betrieblichen Praxis stellt beispielsweise eine Monitoring-Funktionalität von Supply-Chain-Event-Management-Systemen ein potenzielles Einsatzfeld für CEP dar. Ein entsprechendes System kann hierbei kritische Abweichungen innerhalb der Lieferketten detektieren, welche aufgrund des hohen Datenvolumens andernfalls möglicherweise unerkannt bzw. erst rückwirkend erkannt worden wären. Generell existieren im Bereich des (Complex) Event Processing zahlreiche Systeme, oft im dargestellten Kontext von Unternehmensanwendungen, bei welchen eine ereignisgesteuerte Komponente eine serviceorientierte Architektur dahingehend ergänzen kann, dass Dienste durch Ereignisse ausgelöst werden können. Neben diesen Szenarien ist das Gebiet des Event Processing nach wie vor intensiver Forschungsgegenstand bzw. in zahlreichen Projekten vertreten [17]. Es existieren verschiedene frei verfügbare Stand-Alone-Systeme, beispielsweise *SASE*[31] *(Stream-based And Shared Event processing)* oder *Cayuga*[32]. Diese laufen in Folge dessen als eigenständige Anwendungen und können nur über die angebotenen Schnittstellen in ein externes System integriert werden. Eine direktere Integration auf Implementierungsebene ermöglichen hingegen Bibliotheken, respektive Frameworks, wie

31 http://sase.cs.umass.edu/ [20.11.11]

32 http://www.cs.cornell.edu/database/cayuga/ [20.11.11]

beispielsweise *Esper*. Esper[33] ist ein Event-Processing-System für CEP und ESP-Anwendungen, welches sowohl für Java (als *Esper*) sowie für .NET (als *Nesper*) verfügbar ist. Esper liefert eine API mit entsprechenden Methoden, welche Ereignisse in verschiedenen technischen Formaten verarbeiten können, beispielsweise als Schlüssel-Werte-Paare, XML oder direkt mittels sogenannter POJOs (Plain Old Java Object) [14]. Insbesondere letzteres ermöglicht eine effiziente Implementierung, da praktisch jedes Java Objekt (bzw. Klasse) direkt, ohne vorherige Transformationsprozesse, analysiert werden kann. Das heißt, Esper kann mittels *Reflection* direkt auf Objektattribute und deren Werte zugreifen. Des Weiteren liefert Esper eine komfortable Beschreibungssprache (die sogenannte EPL, *Event Processing Language*) sowie eine ausführliche Dokumentation [25]. Esper bietet zwei wesentliche Hauptkomponenten: einen Query- sowie einen Listener/Subscriber-Mechanismus. Ersteres dient unter anderem der Definition und dem Filtern von Ereignissen. Der zweiten Komponente werden die erkannten Events aus den Abfragen *(Query)* übermittelt, innerhalb dieser können dann weitere Aktionen durchgeführt werden, beispielsweise beliebiger Java-Code. Neben den genannten Vertreten existieren weitere kommerzielle Systeme wie beispielsweise *BEA WebLogic*[34] oder *StreamBase*[35]. Eine sehr umfassende Übersicht über zahlreiche Systeme liefert beispielsweise [48]. Um zu entscheiden, welches System für einen konkreten Anwendungsfall geeignet ist, sind entsprechende Benchmarks notwendig, die Erstellung dieser ist aber im Bereich der Event-Processing-Systeme nicht trivial aufgrund der komplexen Anforderungen. Das Projekt *BiCEP*[36] beschäftigt sich mit derartigen Fragestellungen und hat unter anderem die Entwicklung eines universellen Frameworks *(FINCoS)* explizit für die Evaluierung von Event-Processing-Systemen zum Ziel. Zum gegenwärtigen Zeitpunkt existieren keine einheitlichen Benchmarks, welche eine Vergleichbarkeit gewährleisten würden. Der einzige Anbieter eines derartigen Benchmarks ist *STAC*[37] bzw. deren sogenanntes *A1 Benchmark*, welches jedoch nur von einem einzigen Anbieter evaluiert wurde [11]. Innerhalb von [11] finden sich des Weiteren auch mögliche Ursachen für diesen Umstand.

2.6.4 Beschreibungssprache

Allen System gemein ist die notwendige Eigenschaft, Ereignismuster mittels einer geeigneten Beschreibungssprache, einer sogenannten *Event Processing Language,* deklarativ definieren zu können. Hierbei existiert jedoch kein offener bzw. definierter Standard, jedes System verwendet eine eigene Notation bzw. Syntax, was sich auch in der Bezeichnung der jeweiligen Sprachen wieder findet: *Cayuga Event Language* (CEL), *Event Query Language* (EQL), *Event Processing Language* (EPL), *SASE+* sind nur einige exemplarische Vertreter. Als Gemeinsamkeit lässt sich jedoch festhalten, dass die meisten Sprachen an die verbreitete Datenbankabfragesprache SQL angelehnt sind, bzw. Untermengen sogar identisch zu dieser sind. Von einigen Autoren wird dieser Ansatz jedoch in Frage gestellt [67], da eine deklarative Beschreibung in einigen Szenarien, im Vergleich zu einem imperativen Vorgehen, wesentlich aufwändiger sein kann. Nachfolgend wird jedoch nur die deklarative Sprache von *Esper* näher erläutert, da die meisten genannten Systeme ebenfalls mit SQL-artigen Sprachen arbeiten [18]. Die folgenden Strukturen beziehen sich auf eine stark vereinfachte Finanzanwendung, bei welcher

33 http://esper.codehaus.org/ [20.11.11]
34 http://ww.bea.com/ [20.11.11]
35 http://www.streambase.com/ [20.11.11]
36 http://bicep.dei.uc.pt/ [20.11.11]
37 http://www.stacresearch.com/ [20.11.11]

neben anderen fiktiven Ereignissen, exemplarisch das Event einer Abbuchungen betrachtet wird. Ein simples Beispiel einer Abfrage über diesem Ereignisstrom könnte wie folgt lauten:

```
1    select * from Abbuchung
```

Listing 3: EPL-Beispiel 1

Diese Anweisung filtert aus allen aufgetretenen Ereignissen jene vom Typ „Abbuchung“ heraus, deutlich wird hier, dass eine derartige Abfrage identisch zu SQL ist. EPL unterstützt auch einen *Window*-Mechanismus, mit welchem Gruppen von Ereignissen betrachtet werden können:

```
1    select * from Abbuchung.win:length(5)
```

Listing 4: EPL-Beispiel 2

Diese Abfrage ermöglicht übergeordneten Anwendungen Berechnungen über die letzten fünf Abbuchungen anzustellen, das System informiert diese dabei über jegliche Änderung. Dies kann beispielsweise für folgende Berechnung genutzt werden:

```
1    select person, avg(betrag)
2    from Abbuchung.win:time(4 sec)
3    group by person
4    having betrag > 1000
```

Listing 5: EPL-Beispiel 3

Das dritte Beispiel beschreibt ein Szenario, bei welchem alle Personen ausgegeben werden, welche innerhalb der letzten vier Sekunden einen durchschnittlichen Abbuchungsbetrag größer 1000 vorzuweisen hatten. Werden diese Ereignisdefinitionen im System registriert, können mittels boolescher Operatoren korrelierende Events analysiert werden:

```
1    select a,b from pattern [ every a=EPL2 ->b= EPL3 ]
```

Listing 6: EPL-Beispiel 4

Beispiel vier emittiert ein Ereignis, wenn das definierte Event aus Beispiel 2 (EPL2) und anschließend Beispiel 3 (EPL3) eintritt. Diese Auflistung stellt nur einen sehr groben Auszug möglicher (Esper EPL) Konstrukte dar, weitere komplexere Beispiele finden sich in [25]. Eine umfassende Liste weiterer Event-Processing-Sprachen findet sich unter [3].

Bezogen auf das zu konzipierende System lässt sich somit festhalten, dass mittels einer geeigneten EPL die innerhalb von Abschnitt 2.5.3.2 beschriebenen Relationstypen *(Interaktions- und Ereignismuster)* formuliert werden könnten. Insbesondere erscheint ein deklarativer Ansatz, bezogen auf die Adaptierbarkeit des Systems durch einen Domänenexperten, zielführend.

2.7 State of the Art

Der nachfolgende Abschnitt soll bereits vorhandene Systeme und Ansätze vorstellen, welche im weitestgehenden Sinne den definierten Anforderungen entsprechen bzw. ähnliche Zielstellungen besitzen. Hierbei soll mittels exemplarisch ausgewählten Anwendungen, ein möglichst breites Spektrum abgedeckt und charakterisiert werden.

2.7.1 Personalisierung und Ranking

Wie in Abschnitt 2.1 bereits diskutiert wurde, haben sich Feeds mittlerweile als De-facto-Standard für die Bereitstellung von Nachrichtenströmen etabliert, praktisch jede Webseite, respektive Anwendung, bietet einen oder mehrere, teilweise dynamische[38] Feeds an. In Folge dessen erhöht sich auch die Anzahl der von den Nutzern abonnierten Quellen enorm, teilweise weit bis über 100 Abonnements [10, 44]. Lösungsansätze für das Problem des damit einhergehenden *Information Overload* (vgl. Abschnitt 1.1) werden in Folge dessen zunehmend wichtiger: ein Nutzer hat gar nicht die Absicht, jegliche Nachrichten zu lesen – der Aggregator filtert und bewertet einzelne Beiträge, und präsentiert nur jene mit einer entsprechenden Relevanz. Der grundsätzliche Ansatz besteht demnach darin, dem Aggregator ein möglichst umfassendes Modell des Nutzers zur Verfügung zu stellen: welche Interessen hat dieser? Gibt es Schlüsselwörter, welche besonders relevant sind? Welche Beiträge werden intensiv gelesen, welche praktisch nur überflogen und vieles mehr. Eingehende Nachrichten werden in Folge dessen gegen das Nutzermodell geprüft und nur bei ausreichender Relevanz weitergeleitet. Nachfolgend sollen einiger Vertreter dieses Typs kurz charakterisiert werden: **PostRank**[39] versucht die Relevanz einer Nachricht zu ermitteln, indem beispielsweise verweisende Links (ähnlich dem *PageRank* Algorithmus [12] von *Google*) oder auch die Anzahl der Kommentare zu einem (Blog) Beitrag analysiert werden. Neben diesen Faktoren werden zahlreiche weitere Analysen durchgeführt, welche unter dem Begriff *Engagement* zusammengefasst werden und in der Summe ein Signifikanzmaß ergeben. Der eigentliche (Text) Inhalt wird jedoch nicht betrachtet. **NectaRSS** [76] analysiert das Nutzerverhalten, indem es untersucht, welche Nachrichten tatsächlich gelesen werden und was jene Nachrichten charakterisiert. Zum Einsatz kommt das Information-Retrieval-Vektorraummodell, um Nutzer bzw. deren Interessen abzubilden und mittels diesem Vergleiche gegen neu eintreffende Nachrichten anstellen zu können. Prinzipbedingt ist dieses System auf eine Trainingsphase angewiesen. **PersoNews** [8] arbeitet ähnlich NectaRSS, es werden jedoch zusätzlich auch explizite Bewertungen, ähnlich der Funktionsweise eines Spam Filters, mit berücksichtigt. PersoNews verwaltet des Weiteren auch sogenannte *Themenbereiche*, welche Ontologien von Begriffen darstellen. Mittels dieser kann das System in Relation stehende Nachrichten erkennen, auch wenn diese inhaltlich nur eine geringe Ähnlichkeit haben. **FeedWinnower** [43] versucht das Prinzip der Facettenklassifikation [47] auf Feeds anzuwenden, indem diese nach Thema, Personen, Quelle und Zeit strukturiert werden. Der Nutzer hat so die Möglichkeit, einzelne Dimensionen separat zu betrachten und Interessengebiete sukzessive zu filtern. In der Arbeit **Synthesizing Correlated RSS**

38 Dynamisch bedeutet in diesem Kontext, dass die Feed URL verschiedene Parameter bereitstellt, welche die Generierung des eigentlichen XML-Dokumentes (des Feeds) steuern, siehe beispielsweise http://feeds.gametrailers.com/rssgenform.php bzw. http://feeds.gametrailers.com/rssgenerate.php?s1=&favplats[wii]=wii&vidformat[flv]=on&embed=on&quality[sd]=on&agegate[no]=on&orderby=allpopular&limit=5

39 http://www.postrank.com [11.07.11]

News Articles Based on a Fuzzy Equivalence Relation [69] werden Ansätze präsentiert, welche unter anderem mittels Clustering-Methoden und einem Fuzzy-Set-Information-Retrieval-Modell Mengen von relevanten und nicht redundanten Nachrichten versuchen zu filtern. Neben den genannten Systemen existieren noch zahlreiche weitere Ansätze. Grundsätzlich hat jedoch jede dieser Anwendungen das Ziel, die Ausgangsmenge m von Feeds dahingehend zu analysieren, dass aus dieser eine Menge n generiert werden kann (im Sinne einer Sicht), für welche $(m \supset n) \wedge (|m| \gg |n|)$ gilt, indem unter anderem redundante sowie uninteressante Einträge entfernt werden. Hierbei wird jedoch prinzipiell von unstrukturierten Informationen ausgegangen und in Folge dessen keine Relationsdetektion unterstützt. Des Weiteren werden keine Aktivitäten untersucht, respektive auch kein Activity Stream der aggregierten Nachrichten generiert.

2.7.2 RSS Feed Complex Event Detection

In [34] wird ein sogenanntes *Complex Event Detection* (CED) System beschrieben, welches es ermöglicht, mittels einer deklarativen SQL-artigen Sprache *(RSSQL)* Abfragen über RSS Feeds stellen zu können. Dabei definiert im Kontext des Systems, nachfolgend als *RSS CED* bezeichnet, jede Abfrage ein zu überwachendes Ereignis. Einzelne Nachrichten von überwachten Feeds werden hierbei als primitive Events behandelt, diese wiederum können in sogenannten komplexen Events verwendet werden, um so umfassende Ereignisse definieren zu können. Analog können komplexe Events auch selbst wieder in weiteren komplexen Events, ähnlich einem Kompositum [35], eingesetzt werden. Innerhalb der Abfragen (welche in diesem Sinne Filtern entsprechen) unterstützt RSS CED weiterhin noch verschiedenartige Operatoren wie beispielsweise logisches *AND* bzw. *OR* sowie Sequenzen, welche einzelne Events in abhängige Relationen setzen können. Jede durch den Nutzer definierte Abfrage wird im System registriert und fortan kontinuierlich gegen die zu überwachenden Feeds geprüft. Einer der wesentlichen Unterschiede zu vergleichbaren Systemen ist die Tatsache, dass die hier beschriebene Anwendung zustandsbehaftet arbeitet, dies bedeutet, dass detektierte Events zwischengespeichert werden um so komplexere Ereignisse erkennen zu können. RSSQL unterstützt jedoch nur eine geringe Teilmenge der SQL-Syntax bzw. deren Operatoren, trotzdem sind mit dieser komplexere Analysen bzw. Events definierbar, als beispielsweise mittels einfacher Wortvergleiche. Der nachfolgende Auszug beschreibt ein einfaches Event in RSSQL.

```
1   SELECT title, description, link, pubDate
2   FROM all_news, all_blogs
3   MATCH_RECOGNIZE(
4   PATTERN and(A, B; 5)
5   DEFINE A AS keyword_filter({*}, {''Baseball'', ''Scandal''}) AND feed_name = ''all_news''
6          B AS keyword_filter({title}, {A.link}) AND feed_name = ''all_blogs''
7   );
```

Listing 7: Beispiel einer RSSQL-Abfrage

Diese Abfrage beschreibt ein Event, welches emittiert wird, wenn ein Kommentar zu einem Blog-Eintrag abgegeben wird, welcher ein bestimmtes Thema beschreibt.

Schlussfolgerung Der von RSS CED gewählte Ansatz, Ereignisse innerhalb von Feeds mittels einer kompakten SQL-artigen Sprache definieren zu können, erscheint zunächst

viel versprechend. Für einfache Zielstellungen ist dieser Ansatz durchaus geeignet, jedoch bezogen auf die definierten Anforderungen (vgl. Abschnitt 1.5) unzureichend. Folgendes Beispiel soll dies verdeutlichen: Emittiert ein Issue-Tracking-System einen Bug Report, so sollte unter anderem untersucht werden, ob bereits Foren- oder Blog-Einträge existieren, welche semantisch in Relation zu diesem stehen. Die Detektion des Bugs ist hierbei trivial, die Analyse einer Ähnlichkeit mittels RSSQL jedoch kaum möglich. Andere Abfragen wären jedoch denkbar, beispielsweise eine Korrelation zwischen einem Check-In im Versionsverwaltungssystem und einem als gelöst markierten Bug, indem der zeitliche Abstand und der Autor untersucht werden, vgl. Abschnitt 2.7.4. Generell ist jedoch die Ausdrucksstärke von RSSQL für tiefer gehende Analysen unzureichend, [34] beschreibt ebenfalls ein Szenario, mit der Erkenntnis:

> *„By trying to run some of these queries* [zuvor beschriebenes komplexeres Szenario] *we realized that our language made it very difficult to do so.“*

Des Weiteren bietet RSS CED keine aktivitätsbezogene Sicht. Aufgrund des begrenzten Sprachumfanges könnte das System somit nur eine geringe Teilmenge der Anforderungen abdecken. Prinzipiell erscheint jedoch der Ansatz, Muster von Ereignissen mittels einer deklarativen Sprache über Web-Feed-Formaten zu definieren, dennoch viel versprechend. Betrachtet man jedoch vergleichbare universellere Ansätze (vgl. Abschnitt 2.6.3), so bietet RSS CED bezogen auf das zu konzipierende System keine nennenswerten Vorteile.

2.7.3 FriendFeed

FriendFeed[40] ist ein sogenannter *Lifestreaming-Dienst* welcher Feeds von Blogs, Twitter, SocialBookmarking Diensten und vielen Weiteren aggregieren kann. Entstanden ist das Projekt aufgrund der Tatsache, dass innerhalb des Web 2.0 immer mehr Dienste zur Verfügung stehen, welche die Aktivitäten einzelner Nutzer externen Anwendungen mittels entsprechender Feeds zur Verfügung stellen. In Folge dessen sind die einzelnen Informationen immer stärker über verschiedenartige Systeme verteilt, man spricht in diesem Zusammenhang ebenfalls von sogenanntem *Information Scattering* [39], vgl. Abschnitt 1.1. Neben FriendFeed existieren noch zahlreiche ähnliche Anwendungen [2], welche aber an dieser Stelle nicht im Detail diskutiert werden sollen, da der grundsätzliche Ansatz meist identisch ist. Prinzipiell handelt es sich bei FriendFeed primär um einen einfachen Feed-Aggregator, welcher jedoch versucht, die große Menge an heterogenen Diensten und deren Feeds dahingehend zusammen zu führen, dass diese dem Endnutzer mittels einer einheitlichen Syntax präsentiert werden – einem Activity Stream. FriendFeed ist somit in der Lage, aus unstrukturierten Quellen entsprechende Aktivitäten ableiten zu können. Laut [63] besteht das Kernproblem dieses Konzeptes in der mangelnden Skalierbarkeit: für jeden Dienst, welcher unterstützt werden soll und einen individuellen Feed mit individueller Syntax anbietet, muss ein entsprechender Adapter generiert werden. Ändert eine Quelle ihre Syntax, so muss auch der Adapter entsprechend angepasst werden, in Abschnitt 2.2.1 wurde diese Problematik bereits ausführlicher diskutiert. Welchen Aufwand dies konkret nach sich zieht und mittels welcher Algorithmen FriendFeed die Ausgangsdaten auswertet ist nicht bekannt, bzw. wurde nicht veröffentlicht.

Schlussfolgerung Bezogen auf das zu konzipierende System zeigt FriendFeed zumindest die grundlegende Realisierbarkeit einer der wesentlichen Anforderungen (vgl.

40 http://friendfeed.com [11.07.11]

Abschnitt 1.5 unter Systemsicht): die Transformation von heterogenen Quellen mit unterschiedlicher Syntax, in einen aktivitätsbezogenen Nachrichtenstrom. FriendFeed unterstützt jedoch keine Erkennung und Abbildung von Relationen zwischen Nachrichten. In Folge dessen werden diese, analog zahlreicher ähnlicher Angebote, als flacher Strom chronologisch sortiert dargestellt.

2.7.4 Hipikat

Hipikat[41] [19] basiert auf folgendem Grundgedanken: in einem Software-Entwicklungsprojekt ist es für Neueinsteiger (welche aktiv an dem Projekt mitwirken wollen) teilweise sehr schwierig, einen effizienten Einstieg zu finden. Insbesondere innerhalb der Anfangsphase müssen hierbei große Datenmengen bzw. Informationen (Modelle, Klassen, Konventionen, Architektur und vieles mehr) erfasst, analysiert und verarbeitet werden. Um beispielsweise Informationen über den Entwicklungsprozess oder die Systemarchitektur erhalten zu können, müssen verschiedenartige Quellen bzw. Repositorien analysiert werden: die Versionsverwaltung, Issue-Tracking-Systeme, Newsgroups und Dokumentationen. Das Ziel von Hipikat ist die Generierung eines sogenannten *Implicit Group Memory* (IGM), indem die genannten Datenquellen auf existierende Relationen untersucht werden, um diese dem Nutzer präsentieren zu können. Hipikat arbeitet aufgabenorientiert, das heißt, der Nutzer soll passend zu seinem gegenwärtigen Ziel adäquate Informationen, in Form von relevanten und in Relation stehenden Artefakten, erhalten können. Der Prototyp untersucht hierbei folgende Quellen: das Versionsverwaltungssystem *CVS*[42], das Issue-Tracking-System *Bugzilla*[43], diverse Kommunikationskanäle (Newsgroups sowie Mailinglisten) sowie Online-Dokumentationen. Hipikat besteht konzeptionell aus drei Komponenten: *Identification*, *Selection* sowie *Update*. Jegliche Artefakte müssen zunächst auf potentielle Relationen untersucht werden, um als Ergebnis das IGM erstellen zu können (Identification). Der Nutzer kann anschließend explizit oder implizit Anfragen (Queries) an das System stellen, welches als Antwort relevante Artefakte identifiziert und zurück liefert (Selection). Da kontinuierlich neue Daten generiert werden, müssen die Quellen entsprechend überwacht und das IGM stetig aktualisiert werden (Update). Um die bereits erwähnten Relationen zwischen Artefakten abbilden zu können, wurde ein Schema entwickelt (Abbildung 5), welches speziell auf die Domäne der Softwareentwicklung ausgelegt ist. In [19] werden jedoch weder Details präsentiert, wie die Schemata intern abgebildet werden, noch wie diese auf unterschiedliche Domänen angepasst werden könnten bzw. welchen Aufwand dies hätte.
Ohne dass dies in der Arbeit explizit erwähnt wurde, kann bei Hipikat eine aktivitätsbezogene Sicht abgeleitet werden: jede Entität steht mit *Person* in Verbindung, in Folge dessen kann ausgehend von dieser, eine entsprechende Aktivität bestimmt werden, da die hierfür notwendigen Prädikate ebenfalls bereits berücksichtigt wurden (beispielsweise *Person posts Message*). Das Datenmodell erlaubt explizit auch ein Traversieren der Entitäten. Einer der Kernaspekte des Systems stellt die Funktionsweise der Identifikationskomponente dar, deren Aufgabe es ist, Relationen zu detektieren. Hipikat versucht dies unter anderem mit folgenden Ansätzen:

41 Hipikat bedeutet *Augen weit geöffnet* in der West Afrikanischen Sprache *Wolof*
42 http://www.cvshome.org [11.07.11]
43 http://www.bugzilla.org [11.07.11]

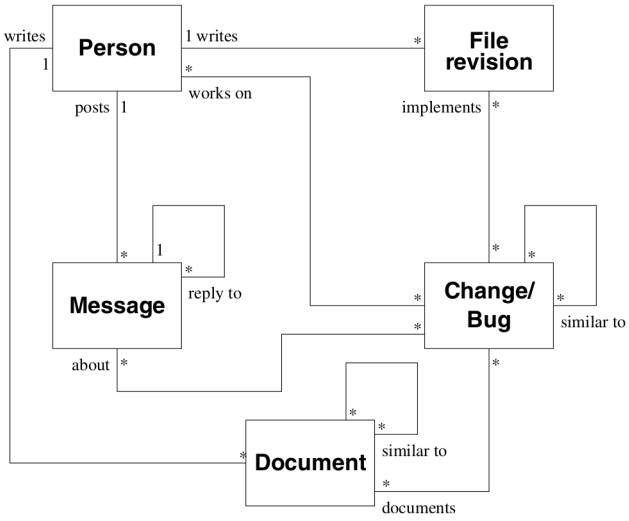


Abbildung 5: Datenmodell (ERM) Hipikat [19]

- Innerhalb von Logeinträgen des Versionsverwaltungssystems wird nach definierten **Syntaxmustern** gesucht, beispielsweise in Form von Identifikationsnummern, um Korrelationen zwischen diesem und dem Issue-Tracking-System erkennen zu können. Beispielsweise "Fixes bug id" oder "id: .. ".
- Es werden **temporale Muster** gesucht, welche mit weiteren Metadaten kombiniert werden. Dies soll an einem Beispiel verdeutlicht werden: Entwickler A behebt einen Bug, er checkt diesen in die Versionsverwaltung ein (ohne einen entsprechenden Log-Eintrag) und setzt anschließend den Bug im Issue-Tracking-System auf gelöst. Das System erkennt hierbei, dass ein und derselbe Autor in einem kurzen zeitlichen Abstand zwei Ereignisse generiert hat, in Folge dessen ist es wahrscheinlich, dass die eingecheckte Datei in Relationen zu dem Bug steht.
- Jegliche Nachrichten werden indiziert und auf Ähnlichkeiten mit bereits vorhandenen Elementen überprüft. Hierfür wird das **Vektorraummodell** aus dem Bereich des *Information Retrieval* [61] angewandt. Dieser Mechanismus wird des Weiteren auch für die eigentlichen Anfragen an das System genutzt.
- Innerhalb von Newsgroups und Mailinglisten wird nach **Referenz-Headern** in den Metadaten gesucht.

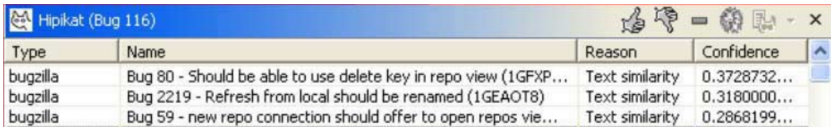


Abbildung 6: Screenshot Hipikat Vorschläge [19]

Hipikat nutzt keine Feeds für den Datenimport, vielmehr existiert für jede Quelle eine eigenständige Komponente: ein Crawler für Webseiten, native Befehle zur Abfrage der Versionsverwaltung und NNTP[44], um Nachrichten erfassen zu können. Hierbei werden

44 NNTP (Network News Transfer Protocol), ein Übertragungsprotokoll für Nachrichten in Newsgroups

stets zahlreiche Metadaten ermittelt, beispielsweise der Zeitpunkt eines Updates. Die Vorschläge der relevanten Artefakte präsentiert Hipikat angereichert mit zwei weiteren Attributen (vgl. Abbildung 6): einer Begründung *(Reason)* sowie einem sogenannten *Confidence*-Wert, welcher einen Indikator für die Korrektheit der erkannten Relationen darstellt.

Schlussfolgerung Die von Hipikat präsentierten Ansätze lassen sich teilweise auch für die innerhalb des zu konzipierenden Systems geschilderten Anforderungen (vgl. Abschnitt 1.5) übertragen, insbesondere der Einsatz von Mechanismen aus dem Bereich des Information Retrieval sowie die Erkennung von temporalen Mustern, erscheinen viel versprechend. Hipikat analysiert und unterstützt jedoch keine reduzierbaren Hierarchien von Nachrichten, es wird stets je eine Menge von eins-zu-eins-Relation emittiert. Das System muss sich aufgrund dieser Funktionsweise auch nicht auf eine konkrete Relation festlegen, im Gegensatz zu einer Hierarchie, bei welcher nicht bei jedem Knoten Entitätsmengen angeboten werden können. Ferner bietet Hipikat zwar theoretisch eine Grundlage für die Modellierung einer aktivitätsbezogenen Systemsicht (siehe Abbildung 5), setzt diese jedoch selbst nicht ein. In Folge dessen wird auch kein Activity Stream modelliert bzw. emittiert, das System bietet generell keinen Nachrichtenstrom in Form eines Feeds[45] an. Aufgrund der zugrunde liegenden Motivation ist das System stark auf eine Domäne (Software-Entwicklungsprojekte) fixiert, was sich auch in dem bereits diskutierten Datenschema wieder spiegelt. Bezogen auf diesen Aspekt stellt Hipikat somit einen möglichen Ansatz dar, eine domänenunabhängigkeit ist jedoch nicht gegeben. Neben Hipikat existieren noch weitere Systeme mit ähnlicher Zielsetzung bzw. Architektur, beispielsweise *softChange* [36], bzw. generell Anwendungen welche in den Bereich der Visualisierung von Softwareentwicklungslinien fallen. Diese sollen aber aufgrund der grundsätzlich vergleichbaren Ausrichtung an dieser Stelle nicht näher diskutiert werden.

2.7.5 Social Stream Event Detection

In [93] und [77] werden sogenannte „Social Stream Event Detection“ Systeme beschrieben, welche innerhalb der Nachrichtenströme von sozialen Medien (Blogs, Usenet, Foren etc.) *Events* detektieren sollen. Der Grundgedanke basiert auf der Annahme, dass diese *Social Streams* als eine Art Sensoren der „realen“ Welt betrachtet werden können, unter *Events* werden in diesem Kontext somit real stattgefundene Ereignisse verstanden. Die Zielstellung ist hierbei, aus den Texten Ereignisse verschiedener Granularität (größere Zusammenhänge werden als *Episode* bzw. *Sage* bezeichnet) zu extrahieren, um anschließend bei Bedarf einzelne Dokumente den detektieren Ereignissen zuordnen zu können. Um dieses Ziel zu erreichen, werden aus den emittierten Nachrichten Schlüsselwörter *(Keywords)* mit verschieden Verfahren extrahiert, welche die Nachricht in Folge dessen charakterisieren respektive repräsentieren. Diese werden anschließend innerhalb eines sogenannten *KeyGraph* modelliert: Knoten entsprechen hierbei einzelnen Schlüsselwörtern, treten zwei Wörter in einem Zusammenhang *(Kontext)* auf, so wird dies mit einer Kante zwischen diesen abgebildet. Stärker gewichtete Kanten deuten in Folge dessen auf eine Korrelation hin. Auf diese Weise entstehen Gruppen *(cluster)* von Schlüsselwörtern, welche unter Berücksichtigung des zeitlichen Verlaufes Rückschlüsse auf übergeordnete *Events* geben. Wirken diese Ansätze auf den ersten Blick viel versprechend, zeigt sich genauer Betrachtung ein grundsätzliches Problem: das Ziel dieser Systeme ist die Gruppierung von Nachrichten zu übergeordneten Ereignissen bzw. die Extraktion dieser,

45 Hipikat wurde 2003 vorgestellt, zu dieser Zeit war die Verbreitung von Web-Feeds wesentlich geringer.

weniger die Generierung konkreter Relationen auf Nachrichtenebene. Jedoch ist es nicht das Ziel des zu konzipierenden Systems, größere Gruppen von Ereignissen zu generieren, sondern konkrete Relationen innerhalb konkreter Nachrichten zu detektieren. Dennoch kann eine entsprechendes Clustering, respektive Modellierung, eines *KeyGraph* unter Umständen hierfür hilfreich sein.

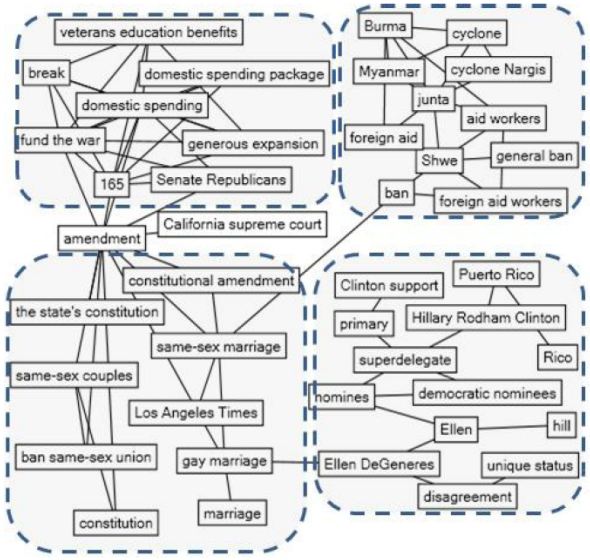


Abbildung 7: KeyGraph von Schlüsselwörtern [77]

2.7.6 Schlussfolgerung

In den voran gegangen Abschnitten wurden exemplarisch verschiedenartige Systeme, respektive Ansätze diskutiert, welche (Teil-) Bereiche der definierten Anforderungen und Ziele abdecken bzw. diesen ähnlich sind. Da innerhalb des zu konzipierenden Systems davon ausgegangen werden kann, dass redundante und unerwünschte Daten einen zu vernachlässigenden Faktor darstellen, sind die in Abschnitt 2.7.1 vorgestellten Anwendungen und Prinzipien eher sekundär. Lediglich einige Prinzipien des Rankings könnten genutzt werden, beispielsweise für rein textuelle Analysen. Anhand von FriendFeed wurde gezeigt, dass es grundsätzlich möglich ist, zur Generierung eines Activity Streams Aktivitäten aus Nachrichtenströmen extrahieren zu können. Es wurden hierbei jedoch keine Angaben über die Funktionsweise bzw. Algorithmen veröffentlicht, des Weiteren untersucht das System keinerlei Relationen – jede Nachricht wird isoliert betrachtet. Das zuletzt vorgestellte System Hipikat kommt den definierten Anforderungen am nächsten, jedoch ist es sehr starr auf eine Domäne und deren Bedingungen ausgerichtet. Es werden zwar gewisse Relationen erkannt, jedoch keine Hierarchien aus diesen generiert – der Nutzer erhält zu einer (auch impliziten) Eingabe stets eine nach Relevanz geordnete Menge von Entitäten. Das im Abschnitt 2.7.2 vorgestellte System zu Detektion von Ereignissen bietet interessante Ansätze für die Konzeption/Implementierung des System, erfüllt jedoch für sich genommen keine der genannten Anforderungen, da lediglich Mechanismen zur Realisierung dieser bereitgestellt werden könnten. Abschließend bleibt festzuhalten, dass eine theoretische Vereinigung der vorstellten Ansätze einen Großteil der definierten Anforderungen und Ziele abdecken würde.

3 Konzeption

In diesem Abschnitt soll die Konzeption des zu realisierenden Systems bzw. dessen Architektur beschrieben werden. Dabei werden die jeweils getroffenen Entwurfsentscheidungen erläutert und mögliche Alternativen zu diesen im Rahmen der Arbeit diskutiert. Des Weiteren soll das das zu konzipierende System nachfolgend mittels des Akronyms RDAS bezeichnet werden, dies steht für *Relation Detection* [and] *Activity Stream* [Transformation].

3.1 Systemarchitektur

Nach nachfolgende Abschnitt soll die logische Struktur sowie die Funktionsweise von RDAS auf einer abstrahierten Ebene erläutern, um dem Leser so ein besseres Verständnis der einzelnen Komponenten und deren Informationsflüsse geben zu können. Eine detaillierte Beschreibung der einzelnen Elemente erfolgt dann in den nachgelagerten Abschnitten, in welchen die konkreten Entwurfsentscheidungen im Speziellen diskutiert werden. Grundsätzliche besteht RDAS aus einer Backend- sowie einer Frontend-Komponente, letztere besteht im Wesentlichen aus der zu implementierenden prototypischen Benutzeroberfläche in Form einer Webanwendung. Das Backend ist konzeptionell für den Import der Daten, die Relationsdetektion sowie die Transformation in einen Activity Stream zuständig[46]. Die Backendprozesse laufen dabei kontinuierlich, das heißt, es erfolgt permanent eine Überprüfung auf neue Daten (neue Nachrichten wurden durch die Quellen emittiert) sowie deren Verarbeitung. Das Frontend, respektive dessen Prozesse, werden nur im Falle eines konkreten Aufrufs initiiert – da diese Komponente im Wesentlichen eine Visualisierung der zuvor generierten Daten darstellt und tendenziell als prototypisch zu betrachten ist, sollen die entsprechenden Funktionalitäten nur auszugsweise erläutert werden. Generell ist die in Abbildung 8 dargestellte Architektur komprimiert worden, um so die wesentlichen Bestandteile hervorheben zu können. Grundsätzlich entspricht die vertikale Anordnung dem logischen Ablauf der Verarbeitung des Systems (von oben nach unten), aus diesem Grund wurde teilweise auf die Angabe von Pfeilen zur Repräsentation des Datenflusses verzichtet. Elemente innerhalb von gepunkteten Bereichen sind von einander unabhängig und können potentiell parallel bzw. in beliebiger Reihenfolge verarbeitet werden.

46 Obligatorische Bereiche wie beispielsweise die Persistierung wurden an dieser Stelle nicht aufgezählt.

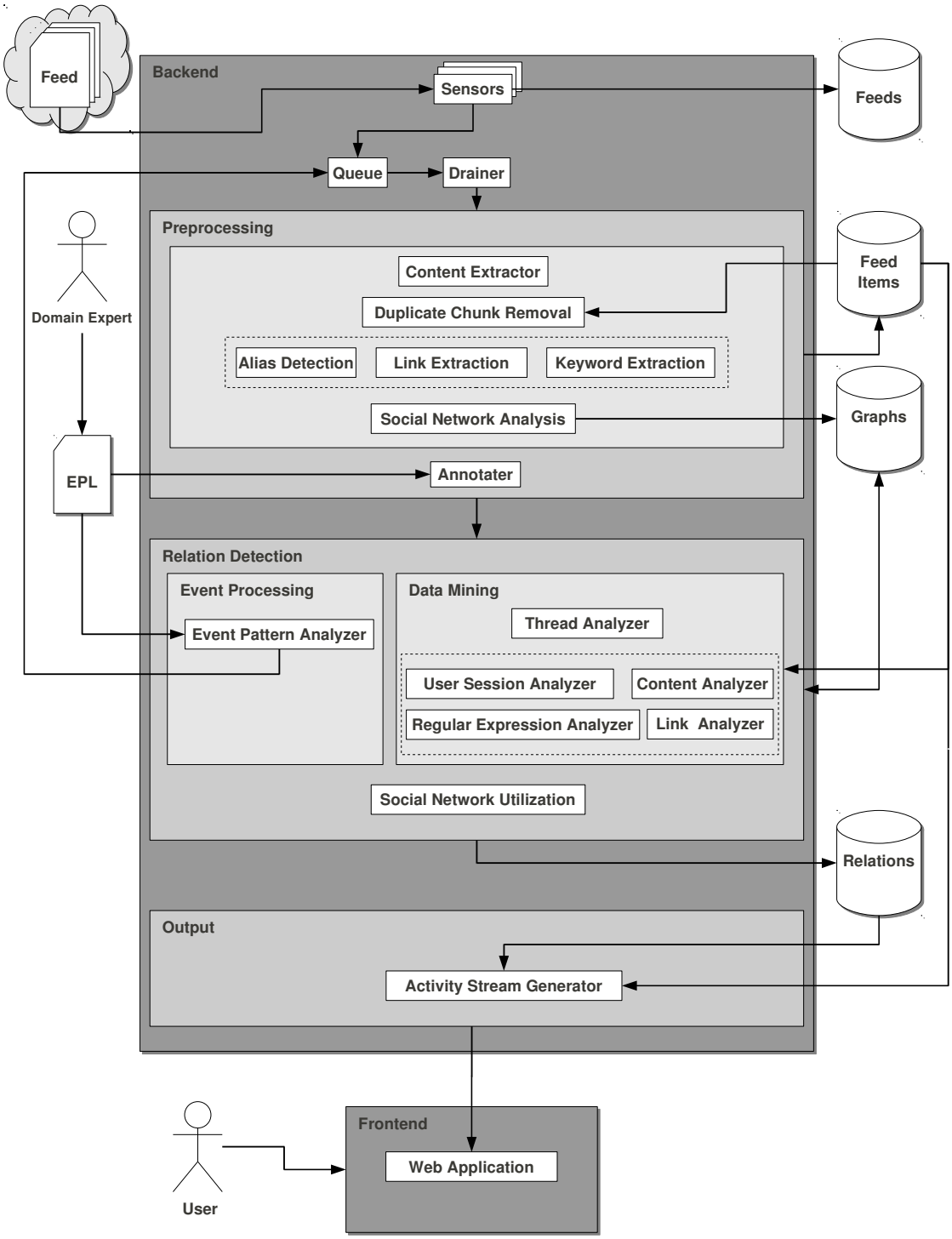


Abbildung 8: Architektur von RDAS

3.1.1 Import

Wie der Name bereits impliziert, besteht die Hauptaufgabe der Import-Komponente im Einlesen der Daten bzw. Abfrage der Informationskanäle. Die empfangenen Nachrichten werden hierbei mittels einer zentralen Warteschlange für die Weiterverarbeitung zur Verfügung gestellt.

- **Feed** repräsentiert jeweils einen konkreten Informationsstrom (vgl. Abschnitt 2.1) und entspricht demnach einer Ressourcen innerhalb des Inter- respektive Intranets.
- **Sensors** sind asynchron arbeitende eigenständige Prozesse, welche jeweils einen konkreten Feed überwachen (mittels Polling, vgl. Abschnitt 2.1) und die emittierten Nachrichten an das System weiterleiten.
- **Queue** repräsentiert die zentrale Nachrichtenwarteschlange, in welche einzelne *Sensors* die empfangenen bzw. herunter geladenen Nachrichten speichern, diese arbeitet nach dem FIFO-Prinzip und fungiert als flüchtiger Zwischenspeicher.
- **Drainer** bezeichnet die zentrale Polling-Komponente des Systems, welche periodisch die *Queue* auf neue Nachrichten überprüft und im positiven Fall die nachgelagerten Verarbeitungsschritte initiiert, gleichzeitig leert sie auf diese Weise die *Queue*.

3.1.2 Preprocessing

Die Vorverarbeitungsschritte werden konzeptionell innerhalb der Komponente *Preprocessing* zusammengefasst. Das Primärziel stellt die Aufbereitung der eingehenden Nachrichten für die nachfolgende Relationsdetektion dar, dies beinhaltet unter anderem die Extraktion der Nutzdaten mittels einer Redundanzreduktion. Des Weiteren erfolgt am Ende dieser Phase die Persistierung der Nachrichten, da diese anschließend nicht weiter modifiziert werden.

- **Content Extractor** bezeichnet den ersten Verarbeitungsschritt der Vorverarbeitung (Preprocessing), welcher die Nachrichten auf deren Nettoinhalt reduzieren soll, indem beispielsweise Signaturen, zitierte Bereiche und redundante Inhalte entfernt werden.
- **Duplicate Chunk Removal** zielt darauf ab, zitierte Bereiche einer Nachricht[47] zu erkennen und zu entfernen, hierfür greift diese auf die bereits gespeicherte Daten bzw. Nachrichten zu, um erkennen zu können, ob ein Textabschnitt bereits existiert ist.
- **Alias Detection** realisiert die in Abschnitt 2.5.2.4 beschriebene Erkennung von einheitlichen Identitäten unterschiedlicher Autorenbezeichner, dies ist für die nachgelagerte *Social Network Analysis* von Relevanz.
- **Link Extraction** bezeichnet eine Komponente, welche die in Abschnitt 2.5.2.1 geschilderte Erkennung und Extraktion von Hyperlinks sowie eine URL-Normalisierung übernimmt.
- **Social Network Analysis** nutzt die zuvor mittels *Alias Detection* extrahierten Identitäten, um einen Graphen sozialer Beziehungen konstruieren zu können (vgl. Abschnitt 2.5.2.4), dieser wird in der Relationsdetektion innerhalb der Komponente *Social Network Detection* ausgewertet.

47 Konkret betrifft dies meist Nachrichten aus Mailinglisten und Foren.

- **Annotator** ist eine Komponente, welche die Nachrichten mit zusätzlichen Informationen anreichert, insbesondere bezogen auf die innerhalb des Activity Streams relevante SPO-Struktur (vgl. Abschnitt 3.6). Die hierfür notwendigen Daten werden mittels einer Konfigurationsdatei unter Verwendung einer EPL (vgl. Abschnitt 2.6.4) definiert.

3.1.3 Relationsdetektion

Die Erkennung von Relationen unterteilt sich in zwei Hauptbereiche: einer rückwärts gerichteten *Data-Mining-Komponente,* welche mittels Methoden des Information Retrieval eine eingehende Nachricht mit bereits bestehenden Daten vergleicht und analysiert sowie eine vorwärts gerichtete *Event-Processing-Komponente*, welche insbesondere für die Erkennung von Mustern zustandsbasiert Analysen durchführt[48].

- **Event Pattern Analyzer** ist für die eben beschriebene Erkennung von Ereignismustern zuständig, welche mittels EPL in einer Konfigurationsdatei dem System zur Verfügung gestellt werden. Hierbei können als Ergebnis auch neue, sogenannte *virtuelle,* Events bzw. Nachrichten generiert werden, welche in die *Queue* eingefügt werden.
- **Thread Analyzer** bezeichnet eine Komponente, welche „thread-artige" Strukturen erkennt und insbesondere zusammengehörige Nachrichten gruppiert.
- **Content Analyzer** ist eine Komponente, welche mittels Methoden des Information Retrieval textuell bzw. inhaltlich ähnliche Nachrichten in Beziehung setzt.
- **User Session Analyzer** untersucht temporal zusammengehörige Aktionen eines Nutzers, um auf diese Weise auf einer weiteren Ebene Beziehungen detektieren zu können.
- **Link Analyzer** wertet die zuvor innerhalb der Vorverarbeitung extrahieren Hyperlinks aus und generiert entsprechende Relationen auf Basis verschiedenartiger Link-Merkmale.
- **Regular Expression Analyzer** bezeichnet eine Komponente, welche einen generelleren Ansatz, in Bezug auf spezielle Entitäten und Syntaxmuster, als der bereits erwähnte *Event Pattern Analyzer* realisiert.
- **Social Network Utilization** wertet das generierte soziale Netzwerk respektive den Graphen aus, um detektiere Relationen ggf. höher zu gewichten, sofern deren Akteure intensive Beziehungen innerhalb des Graphen besitzen.

3.1.4 Output

- **Activity Stream Generator** bezeichnet die eigentliche Generierung der aggregierten Nachrichten bzw. deren Abbildung auf das Activity-Stream-Format. Diese Komponenten wird durch das Frontend jeweils im Bedarfsfall, also durch einen konkreten Aufruf, initiiert.

48 Hierfür wird im Gegensatz zur *Data-Mining-Komponente* nicht auf persistierte Daten zurück gegriffen.

- **Frontend** bezeichnet die im Rahmen des zu konzipierenden Systems realisierte prototypische Benutzeroberfläche in Form einer Webanwendung.

3.2 Import

Im Gegensatz zu den, beispielsweise in Abschnitt 2.7.1, erwähnten Feed-Aggregatoren, muss das zu konzipierende System eine vergleichsweise kleine Menge an Feeds überwachen[49]. Des Weiteren kann davon ausgegangen werden, dass eine Feed-Suchfunktionalität, wie sie beispielsweise in [50] beschrieben wird, ebenfalls nicht erforderlich ist, da die entsprechenden Feeds der einzelnen Tools als bekannt vorausgesetzt werden können.

3.2.1 Inhaltsextraktion

Für eine Analyse und Detektion von Relationen ist ein Zugriff auf den Inhalt einer Nachricht zwingend erforderlich, jedoch kann dies nicht generell als gegeben vorausgesetzt werden. Insbesondere innerhalb von Webangeboten, welche durch das Schalten von Werbung finanziert werden, transportieren deren Feeds nicht den gesamten Inhalt einer Nachricht, stattdessen werden lediglich Auszüge übermittelt, um so den Leser zum Öffnen der eigentlichen Webseite zu animieren[50]. Um dennoch an den gesamten Inhalt einer Nachricht zu gelangen, können sogenannte *Wrapper* bzw. *Scraper*[51] eingesetzt werden, welche teilweise nur anhand der URL mit verschiedenartigen Methoden den eigentlichen Inhalt extrahieren können. Derartige Anwendungen betreffen nicht ausschließlich RSS bzw. Atom Feeds und können generell unter dem Begriff *Screen-* oder *Web Scraping* zusammengefasst werden. Grundsätzlicher Ansatz ist hierbei, die Erkennung und Eliminierung von, in diesem Zusammenhang als Rauschen bezeichneten, unerwünschten Informationen [92]. Dies kann beispielsweise mittels einer sogenannten *Template Detection* realisiert werden, bei welcher der überwiegend statische Teil einer Webseite (Navigationsbereiche, Header, Footer etc.) durch mehrfaches Aufrufen unterschiedlicher Beiträge identifiziert und entfernt wird, um somit das Delta bzw. den reinen Inhalt zu erhalten. Naivere Ansätze bestehen beispielsweise darin, die Position des gekürzten Textes eines Feed-Eintrages auf der entsprechenden Webseite zu suchen und dessen Kontext (umschließende bzw. nachfolgende Textelemente) an bzw. einzufügen. Entsprechende Kenntnisse der Struktur der Webseite (HTML-Baum) vorausgesetzt, kann dies mittels *XPath*[52] realisiert werden. Problematisch bei derartigen Ansätzen ist aber die relativ geringe Robustheit gegenüber Modifikationen der Quelldokumente: selbst kleine Änderungen der XML-Struktur führen zum Fehlerfall. Diese Problematik spielt jedoch im Kontext von FLOSS-Projekten bzw. bezogen auf Feeds entsprechender Tools (beispielsweise eines Issue-Tracking-Systems) nur eine untergeordnete Rolle. Da aufgrund der geringen Datenmengen eine Reduzierung der Inhalte irrelevant ist sowie die Generierung von Seitenbesuchen ebenfalls keine Motivation darstellt, kann davon ausgegangen werden, dass das zu konzipierende System generell Zugriff auf Feeds mit vollständigen Inhalten hat. Ungeachtet dessen soll dieser Umstand innerhalb des zu konzipierenden System durch die optionale Einbindung entsprechender Adapterkomponenten berück-

49 Eine generelle Aussage lässt sich jedoch nicht abschließend treffen, da dies stark von der jeweiligen Domänen abhängig ist.

50 Die Anzahl der Besucher eines Webangebotes ist ein wesentliches Maß für dessen Verbreitung bzw. Relevanz und spiegelt sich dementsprechend in werbefinanzierten Einnahmen wieder.

51 http://fulltextrssfeed.com [11.07.11]

52 Eine Abfragesprache zum Adressieren beliebiger Elementen innerhalb von XML-Dokumenten.

sichtigt werden. Diese würden jedoch eigenständig und insbesondere für die weiterverarbeitenden Prozesse transparent arbeiten.

3.2.2 Archivdaten

Betrachtet man die von den Werkzeugen bereitgestellten Feeds, so lässt sich feststellen, dass diese meist nur zeitnahe bzw. aktuelle Nachrichten beinhalten, da dies letztendlich das Prinzip von Web Feeds darstellt. Für das zu konzipierende System stellt sich somit die Frage, wie Relationen zu Archivdaten bzw. Nachrichten detektiert werden können, wenn diese innerhalb der zu observierenden Feeds nicht mehr existieren. Grundsätzlich existieren für dieses Problem verschiedenartige mögliche Lösungen.

Einige Webanwendung bieten einen direkten **API-Zugriff** um beispielsweise innerhalb der Feed URLs mittels entsprechender Parameter[53] (REST) die Anzahl der zu emittierenden Nachrichten zu erhöhen. Sofern keine obere Schranke existiert, könnte man auf diese Weise ältere Nachrichten auslesen. Diese Möglichkeit ist aber nur bei einer geringen Zahl an Webangeboten vorhanden und kann demnach nicht als gegeben vorausgesetzt werden. Existiert die erst genannte Möglichkeit nicht, so müssten mittels sogenannter **Crawler** im Zusammenspiel mit den bereits erwähnten *Screen Scrapern* die Nachrichten der Webseiten extrahiert werden. Aufgrund der sehr heterogenen Systemlandschaft kann dies jedoch kaum vollständig automatisiert werden und bedarf somit entsprechender Kenntnisse auf diesem Gebiet. Es existieren bereits zahlreiche frei verfügbare Lösungen, beispielsweise *Nutch*[54], bzw. stellt das kommandozeilenbasierende *Wget*[55] ebenfalls bereits einen einfachen Crawler dar. Des Weiteren bieten einige Tools die Möglichkeit eines **Datenexports**, um Archivdaten in einem maschinenlesbaren Format (meistens CSV oder XML) exportieren zu können. Ein bekannter Vertreter, welcher diese Funktionalität bereitstellt, ist beispielsweise *Bugzilla*[56]. Je nach Systemumgebung ist des Weiteren auch ein **direkter Datenbankzugriff** auf die Persistenzschicht, respektive die Datenbank, der Anwendungen denkbar. Wird beispielsweise das bereits erwähnte Bugzilla eingesetzt, so kann auf dessen MySQL-Datenbank zugegriffen werden um die darin enthaltenen Tabellen auslesen zu können. Diese Möglichkeit bieten sehr viele Open-Source-Anwendungen, insbesondere wenn diese einer Client-Server-Architektur entsprechen.

Da das zu konzipierende System eine weitestgehende Domänenunabhängigkeit gewährleisten soll, ist es nicht möglich, für jede Datenquelle einen entsprechenden Crawler zu generieren. Aus diesem Grund soll lediglich eine definierte Schnittstellen für den Datenimport bereitgestellt werden, über welche das System initial mit Nachrichten gefüllt werden kann. Auf welche Art und Weise diese Daten generiert wurden, ist demnach nicht Bestandteil des Systems.

3.3 Datensatz

Um die in Abschnitt 2.5 vorgestellten Relationen in ihren konkreten (möglichen) Ausprägungen analysieren zu können, kam innerhalb der Konzeption bzw. späteren Imple-

53 http://groups.google.com/group/mootools-users/feed/atom_v1_0_topics.xml?**num=100** [09.11.11]
54 http://nutch.apache.org [09.11.11]
55 http://www.gnu.org/software/wget [09.11.11]
56 http://www.bugzilla.org [09.11.11]

mentierung, ein umfassender Datensatz zum Einsatz. Dieser enthält Daten des FLOSS-Projektes *phpMyAdmin*[57] aus den letzten 10 Jahren. Hierbei sind unter anderem jegliche Mailinglisten, ITS, Repositorien-Logs, Entwickler-Blogs sowie Forenbeiträge enthalten. Insgesamt umfasst der Datensatz somit rund 250.000[58] Nachrichten, respektive rund 300 MByte an Textdaten. Große Teile dieses Datensatzes wurden durch die Arbeit innerhalb von [49] zur Verfügung gestellt. Innerhalb der Analyse und Konzeption wurden des Weiteren stichprobenartig Daten aus ähnlichen Quellen hinsichtlich deren Struktur verglichen, beispielsweise [45]. Auf diese Weise soll sichergestellt sein, dass konzeptionelle Entscheidungen nicht zu stark auf konkreten Ausprägungen des vorliegenden Datensatzes basieren. In Abbildung 31 und 32 in Anhang A.3 finden sich Informationen bezüglich der Zusammensetzung dieser Daten.

3.4 Vorverarbeitung

Die nachfolgenden Abschnitte beschreiben die einzelnen konzeptionellen Vorverarbeitungsschritte innerhalb von RDAS. Wie in Abbildung 8 dargestellt ist, werden diese vor der eigentlichen Relationsdetektion abgearbeitet, in dieser Phase werden jedoch noch keine Beziehungen analysiert.

3.4.1 Content Extraction

In einem ersten Vorverarbeitungsschritt soll das zu konzipierende System die eintreffenden Nachrichten auf deren tatsächliche Nutzdaten reduzieren, um die Menge der zu verarbeitenden Daten sowie Redundanzen zu minimieren bzw. eliminieren. **Signaturen** stellen insbesondere innerhalb von Mailinglisten und Foren eine oft genutzte Möglichkeit der Individualisierung dar. Diese enthalten jedoch bezogen auf eine konkrete Nachricht keine relevanten Inhalte (da stets identisch). Oft werden diese mittels spezifischer Trennzeichen (beispielsweise „-- “) eingefügt, was eine Eliminierung relativ einfach ermöglicht. Das zu konzipierende System sollte somit eine konfigurierbare Menge an entsprechenden Trennzeichen auswerten, um mittels dieser Signaturen erkennen und entfernen zu können. In Abhängigkeit der konkreten Domäne sind eine Vielzahl weiterer Möglichkeiten denkbar, beispielsweise fixe Kopfzeilen. Diese könnten mittels regulärer Ausdrücke ebenfalls eliminiert werden. Eine generelle Aussage über deren Struktur ist hierbei jedoch schwieriger, des Weiteren sind treten diese innerhalb des Datensatzes nur in zu vernachlässigenden Größen auf. Grundsätzlich bleibt festzuhalten, dass das zu konzipierende System bezogen auf das Datenformat der Web Feeds im Wesentlichen bereits extrahierte Nutzdaten zur Verfügung hat. Davon ausgenommen sind zitierte Bereiche, welche im folgenden Abschnitt diskutiert werden sollen.

3.4.2 Duplicates Removal

Insbesondere innerhalb von Mailinglisten, aber auch in Foren, stellen zitierten Bereiche in Nachrichten einen teilweise nicht unerheblichen Anteil dar. Bei sehr langen Konversationen können diese durchaus den größten Teil der Nachricht darstellen. Dies ist insofern problematisch, als dass diese Texte zum einen redundant sind und zum anderen zu fehlerhaften Relationen führen können. Beispielsweise könnte Nachricht A Hyperlink *H1* enthalten. Stellt nun Nachricht B eine Antwort auf A dar und enthält A als zitierten

57 http://www.phpmyadmin.net/ [09.11.11]

58 Exakt 248.160 Nachrichten

Bereich (Anhang), so enthält auch Nachricht *B* Hyperlink *H1*, obwohl deren Autor diesen nie generierte. Eine auf *H1* beruhende Relation könnte demnach als fehlerhaft eingestuft werden. Eine Analyse des Datensatzes sowie [57] und [79] zeigen, dass im Wesentlichen drei Arten von Zitierungen zu berücksichtigen sind.

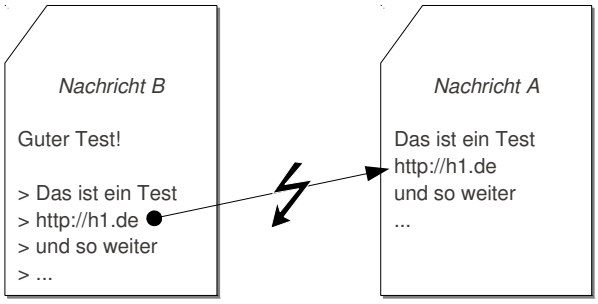


Abbildung 9: Redundante Bereiche und Relationen

Trennzeichen (bzw. Zeichenketten) separieren einzelne Nachrichten innerhalb einer Gesamtnachricht. Dies ist beispielsweise innerhalb von Mailinglisten möglich, bei welchen jede neue Nachricht an oberster Stelle angefügt wird, analog einem Stack bzw. Stapelspeicher. Sind diese Trennzeichen bekannt, kann die Nachricht entsprechend zerlegt werden. **Zeilenpräfixe** sind insbesondere innerhalb von E-Mail-Nachrichten weit verbreitet, die Zeilen einer zitierten Mail werden hierbei (oft) mit einer schließenden Tag-Klammer („>“) beginnend angehängt. Mit jeder weiteren Nachricht erweitert sich demnach die Anzahl der Tag-Klammern, dies stellt somit einen Indikator für die Hierarchie dar. Des Weiteren kommt diese Syntax auch bei händischen Zitaten vor, das heißt, ein Autor zitiert nur einzelne Zeilen einer anderen Nachricht innerhalb des gegenwärtigen Fließtextes. **N-Gramme** stellen eine generelle Möglichkeit dar, einen Text in einzelne Fragmente zu zerlegen, indem mittels eines *Schiebefensterverfahrens* Zeichen- oder Wortgruppen spezifischer Länge n extrahiert werden. Im Gegensatz zu den erstgenannten Möglichkeiten werden hierbei jedoch keinerlei Erkenntnisse bezüglich der Struktur einer Nachricht ausgewertet, das heißt, Zitierungen und original Text sind hier völlig gleichwertig zu betrachten.

RDAS sollte somit konfigurierbare Zeichenketten als Begrenzer verarbeiten und die geschilderte Tag-Klammer-Hierarchie extrahieren können. Im Ergebnis kann zu jeder Nachricht eine Menge an sogenannten *chunks* assoziiert werden – *chunk* steht in diesem Kontext für eine (meist) echte Teilmenge an Wörtern bzw. Sätzen einer Nachricht, welche als zitierter Bereich erkannt wurden. Ein naiver Ansatz könnte nun über diese Menge iterieren und jegliche Vorkommen innerhalb der Originalnachricht entfernen, diese Vorgehensweise ignoriert jedoch die Fragestellung, inwiefern die extrahierten *chunks* auch tatsächlich bereits durch eine vorherige Nachricht emittiert wurden. Beispielsweise muss ausgeschlossen sein, dass eine rein zufällige Verwendung der Trennzeichen oder Tag-Klammer Teile einer Nachricht entfernt, welcher keine Zitate darstellen. Um dies gewährleisten zu können, muss das zu konzipierende System demnach ermitteln, ob ein *chunk* bereits im Korpus vorhanden ist – in diesem Sinne also eine Suchoperation durchführen. Eine ausreichende Länge (der *chunks*) vorausgesetzt, kann im positiven Fall davon ausgegangen werden, dass es sich tatsächlich um einen zitierten Bereich handelt. Aus diesem Grund soll die dargestellte dritte Variante der *N-Gramm Fragmentierung* nicht berücksichtigt werden, da die Anzahl der Vergleiche hierbei um ein Vielfaches höher wäre.

Die innerhalb dieses Prozesses extrahierten Abschnitte werden am Ende aus den Nutzdaten entfernt, sollen jedoch weiterhin mit der Nachricht assoziiert bleiben, da diese Informationen innerhalb der anschließenden Relationsdetektion ausgewertet werden kann, siehe Abschnitt 3.5.3.

3.4.3 Alias Detection

In Abschnitt 2.5.2.4 wurde bereits auf den Umstand eingegangen, dass aufgrund der Heterogenität der Systeme nicht gewährleistet ist, dass eine Person, respektive Identität, immer unter dem gleichen Alias aktiv ist. In [46, 66] werden Lösungen vorgestellt, welche auf Analysen der durch die Autoren generierten Texte basieren. Hierbei existieren Grenzwerte[59], unter welchen eine aussagekräftige Analyse nicht möglich ist. In [66] wird hierfür eine Menge von 3.000 Wörtern genannt, andere (ältere) Ansätze benötigen teilweise wesentlich mehr. Betrachtet man die Beiträge der Autoren innerhalb des phpMyAdmin-Datensatzes, so lässt sich feststellen, dass nur in einigen Ausnahmefällen diese Größe erreicht wird, Abbildung 10 visualisiert diesen Sachverhalt.

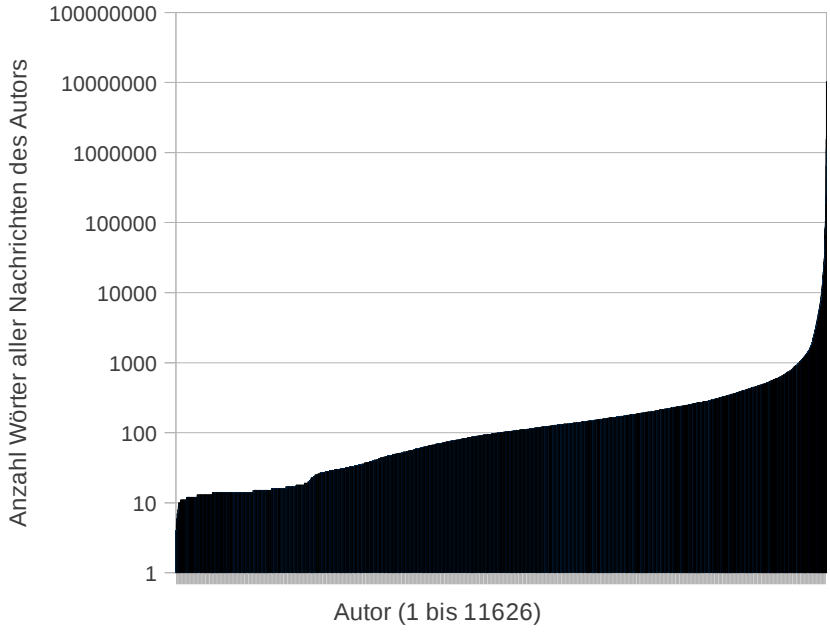


Abbildung 10: Kumulierte Wortmenge der Autoren im phpMyAdmin-Datensatz

Deutlich wird, dass der überwiegende Anteil (>90%) der Autoren nicht genügend Wörter für eine derartige Analyse generiert hat[60], es müssen demnach andere Ansätze gefunden werden. Einen Hinweis liefert hierbei eine Betrachtung der Datensätze der Hauptentwickler[61] bzw. deren Ausprägungen in den einzelnen Systemen (vgl. Abschnitt 2.4), das nachfolgende Listing stellt einen Auszug dieser dar.

59 Bezogen auf die kumuliert Menge der Wörter welcher ein Autor generiert hat

60 Was zusätzlich dadurch verstärkt wird, dass Datenmenge über 10 Jahre umfasst.

61 http://www.phpmyadmin.net/home_page/team.php [29.10.11]

```
 1   <lem9@us...>
 2   delisle <delislma@co...>
 3   Delisle Marc <delislma@Co...>
 4   enfant <lem9@us...>
 5   lem9 <lem9@us...>
 6   lem9@users.sf.net
 7   lem9@users.sourceforge.net
 8   Marc Delisle
 9   Marc Delisle <DelislMa@Co...>
10   Marc Delisle <lem9@us...>
11   Marc Delisle <Marc.Delisle@ce...>
12   Marc Delisle <marc@in...>
```

Listing 8: Verschiedenartige Aliasse des Entwicklers 'Marc Delisle'

Unter der Annahme, dass diese intuitiv[62] zusammengehörigen Bezeichner ein und den selben Entwickler darstellen, ergeben sich somit mindestens 12 verschiedenartige Adressen für „Marc Delisle“. Zerlegt man die einzelnen Adressen in die enthaltenen Wörter bzw. Zeichenketten, so fallen bei genauerer Betrachtung zwei Dinge auf: zum einen dass das Delta oft aufgrund unterschiedlicher Domänenteile entsteht (heterogene Systeme) und zum anderen, dass die Bezeichner in einer transitiven Relation zueinander stehen und die gesamte Menge *(Transitive Hülle)* den Autor repräsentieren könnte. Zeile 10 stellt beispielsweise zu Zeile 1, 4 und 5 eine Relation über das Wort „lem9“ her. Ein entsprechender Algorithmus könnte demnach mittels einer derartigen Analyse folgende Gesamtmenge generieren: *{„delislma@co...“, „lem9@us...“, „marc.delisle@ce...“, „marc@in...“, „delisle“, „delislma“, „enfant“, „lem9“, „marc“, „marc.delisle“}*. In diesem Sinne kann innerhalb der Gesamtmenge aller Autoren ein sogenanntes *Clustering* [61] erfolgen, welches ähnliche Bezeichner gruppiert. *Flache Clustering-Verfahren* (beispielsweise der sogenannten *K-Means-Algorithmus*) erfordern hierfür jedoch meist eine initiale Definition der Anzahl der zu generierenden Cluster, diese ist jedoch nicht bekannt – allein schon aufgrund der Tatsache, dass stetig neue Autoren hinzu kommen können. *Hierarchische Clustering-Verfahren* besitzen diese Einschränkung nicht und wären demnach vorzuziehen. Im Rahmen der Konzeption erfolgte eine Evaluation mittels der *Carrot*[63] *Clustering Engine*, innerhalb dieser wurde der durch das Tool angebotene *Lingo, STC (Suffix Tree Clustering)* sowie *Bisecting K-means* Algorithmus untersucht. Dabei stellte sich heraus, dass diese am Beispiel des Autors „Marc Delisle“ die Menge der Bezeichner ungefähr auf die Hälfte reduzieren konnten, respektive fünf Cluster bildeten. Neben dieser Tatsache ist des Weiteren bei einem hohen Nachrichtenaufkommen problematisch, dass die eingesetzten Algorithmen bzw. deren Berechnung *(Komplexität)* sehr zeitaufwändig sind.

Kritisch bei derartigen Ansätzen ist des Weiteren, dass eine vorherige Zerlegung der Autoren in einzelne Wörter und entsprechende nachgelagerte Vergleiche von der Annahme ausgehen, dass gleiche Bezeichner auch semantisch identisch sind, respektive den gleichen Autor abbilden. Dies Annahme stellt jedoch tendenziell eine Heuristik dar und kann in Abhängigkeit der konkreten Domäne problematisch sein. Grundsätzlich stel-

62 Inwiefern beispielsweise andere Nutzer diese Adressen bewusst ähnlich gewählt haben, ist nicht bekannt bzw. kann als unwahrscheinlich angekommen werden.

63 http://carrot2.org/ [29.10.11] Anm. : die exakte Bezeichnung lautet *Carrot²*

len jegliche Ansätze einen *Trade-off* zwischen einer umfassenderen Abbildung der Autorenbezeichner auf mögliche Identitäten und einer fälschlichen Gruppierung unterschiedlicher Autoren dar.

Um die Komplexitätsprobleme der Clustering-Verfahren zu umgehen, soll in RDAS ein vereinfachter Algorithmus basierend auf Information-Retrieval-Ansätzen zum Einsatz kommen: Jeder Autorenbezeichner wird dabei wie beschrieben in seine Bestandteile (Wörter) zerlegt und stellt initial eine Identität dar. Dieser Wert wird indiziert und in einem entsprechenden Suchindex abgelegt. Beim Auftreten eines neuen Autors wird dieser gegen den gesamten Korpus verglichen und der höchst gewichtete Treffer (vgl. Abschnitt 2.3.1) untersucht: ist dieser dem gesuchten Autorenbezeichner sehr ähnlich (Schwellenwert) so wird dieser um die noch nicht existenten Terme erweitert, ansonsten wird ein neuer Autor generiert.

3.4.4 Link Extraction

In Abschnitt 2.5.2.1 wurde erläutert, welche Hyperlink-Merkmale für Relationen relevant sein können, des Weiteren wurde auf den Umstand der sogenannten URL-Normalisierung eingegangen. Die Extraktion der entsprechenden Entitäten soll mittels regulären Ausdrücken erfolgen (vgl. Abschnitt 2.5.3.2), es existieren hierbei zahlreiche Varianten, welche sich in ihrer Liberalität bezüglich „exotischen“ Protokollen etc. unterscheiden. Eine mögliche Variante findet sich unter [21], mittels diesem konnten in einer Vorevaluation 36.835 Links innerhalb des Datensatzes detektiert werden. Eine stichprobenartige Analyse zeigte des Weiteren, dass der überwiegende Teil der auf diese Weise extrahierten URLs dem „Standardformat“[64] entsprechen, aus diesem Grund soll an dieser Stelle keine umfassende Analyse verschiedenartiger regulärer Ausdrücke und deren Extraktionsmächtigkeit erfolgen.

3.4.5 Keyword Extraction

Um Relationen basierend auf dem Inhalt einer Nachricht untersuchen zu können (vgl. Abschnitt 2.5.2.3), soll das System in der Lage sein, relevante Schlüsselwörter extrahieren zu können. Eine erste Vorevaluation mittels *Lucene* (vgl. Abschnitt 4.2.1) und *Luke*[65] zeigte erwartungsgemäß, dass eine Komprimierung der Texte notwendig ist, da ansonsten die Suchergebnisse zu ungenau sind. Wird beispielsweise kein *Stemming* bzw. *Lemmatisierung* (vgl. Abschnitt 2.3.1) angewendet, so findet eine Suche nach „COMMIT“ bereits keine Ergebnisse welche „commited“ beinhalten, obwohl diese semantisch tendenziell gleichwertig sind. Innerhalb des zu konzipierendes Systems dienen die Schlüsselwörter jedoch nur der Detektion von Beziehungen, das heißt, eine möglichst weitreichende Generierung von Schlüsselwörtern, wie sie beispielsweise in [78] diskutiert wird, ist nicht erforderlich. Aus diesen Gründen soll innerhalb von RDAS ein *POS-Tagging* eingesetzt werden, um mittels diesem jegliche Substantive erkennen und extrahieren zu können (eine Konzentration auf Substantive wird in vielen *Keyword Extraction* Systemen angewandt [37] und erscheint intuitiv nachvollziehbar). In einem weiteren Schritt soll mittels eines geeigneten Lemmatizers eine Reduzierung dieser Substantive auf deren Grundform erfolgen, um die Vergleichbarkeit der Schlüsselwörter weiter zu verbessern. Mit dem bereits erwähnten *Stanford POS-Tagger* sowie dem darin enthaltenen *Lemmatizer* wur-

64 Im Sinne von *http* als Netzwerkprotokoll, Beschränkung auf *ASCII* Zeichen, keine *Unicode* Domains etc.
65 http://code.google.com/p/luke/ [09.11.11] ein rudimentäres GUI für Lucene

den innerhalb einer Vorevaluation gute Ergebnisse erzielt, die Menge der Wörter innerhalb einer Nachricht konnte im Mittel um über 60% reduziert werden. Die Höchstwerte betragen hierbei über 90%, die Minimalwerte teilweise auch <10%, das heißt, die Schlüsselwörter entsprechenden hierbei fast exakt der Originalnachricht.

3.4.6 Social Network Analysis

Die innerhalb der Aliaserkennung extrahierten Identitäten sollen in einem weiteren Schritt genutzt werden, um einen Graphen sozialer Beziehungen generieren zu können. Dieser kann genutzt werden, um Relationen stärker zu gewichten. Der Begriff soziale Beziehung ist im Kontext der Arbeit jedoch weiter zu fassen, als die beispielsweise innerhalb von *Social Networks* bekannten *Freundschaftsbeziehungen.* Daher soll eine entsprechende Analyse globale bzw. generellere Ansätze ermöglichen. Die Fragestellung, welche Interaktionen zwischen einzelnen Autoren hierbei als Merkmal einer *sozialen Beziehung* gelten, ist jedoch nicht generell definierbar, vielmehr kann dies von der spezifischen Domäne abhängig sein. Das System soll daher entsprechend konzipiert werden, so dass ein Domänenexperte geeignete Konfigurationsparameter setzen kann. Einige dieser Faktoren können dennoch bereits vordefiniert werden, diskutieren zwei oder mehr Autoren beispielsweise innerhalb eines Themas (*Threads*), so kann davon ausgegangen werden, dass diese im weitesten Sinne (auf dieser Ebene) eine soziale Beziehung generieren. Je öfter eine derartige Kommunikation stattfindet, desto stärker wird die Ausprägung im sozialen Graphen.

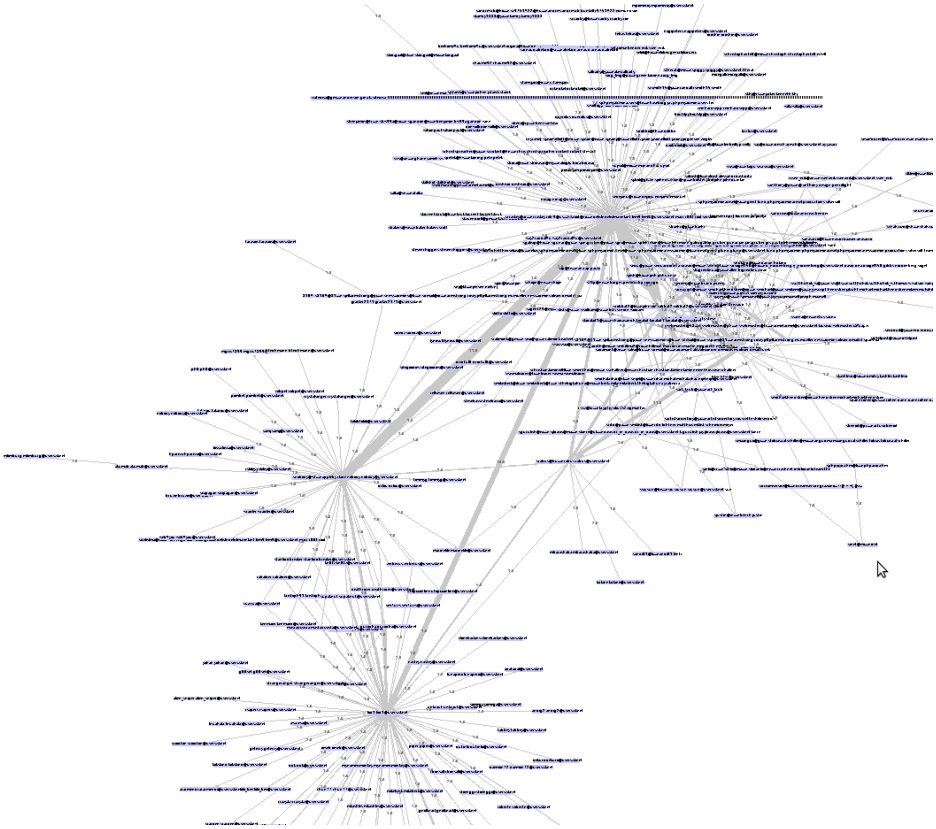


Abbildung 11: Soziale Beziehungen innerhalb des phpMyAdmin-Datensatzes

Eine Vorevaluation (Abbildung 11) zeigte hierbei, dass sich auf diese Weise die Hauptentwickler bzw. sehr aktive Autoren extrahieren lassen. Aufgrund des beschriebenen

Algorithmus der Aliaserkennung, muss das System in der Lage sein, den generierten Graphen aktualisieren zu können, beispielsweise müssen Knoten entfernt und zusammengelegt werden *(Fusion)*. Im Gegensatz zum Relationsgraphen können innerhalb des sozialen Graphen Zyklen auftreten, des Weiteren ist dieser nicht gerichtet. Ferner kann der Graph die Ausgangsbasis für zahlreiche weitere Funktionalitäten darstellen, beispielsweise könnten Nachrichten von sehr aktiven Autoren automatisiert höher gewichtet oder annotiert werden[66].

3.4.7 Annotation

Die Notwendigkeit einer Annotierung ergibt sich aus den geschilderten Anforderungen in Abschnitts 3.6.2 und soll daher an dieser Stelle nicht näher erläutert werden. Der Domänenexperte soll innerhalb einer Konfigurationsdatei Muster definieren können, welche eintreffende Nachrichten um Informationen erweitern oder ggf. auch vorhandene Datenfelder überschreiben. Dieses sogenannte *content enrichment* [14] kann somit auch für die anschließende Relationsdetektion relevant bzw. nützlich sein.

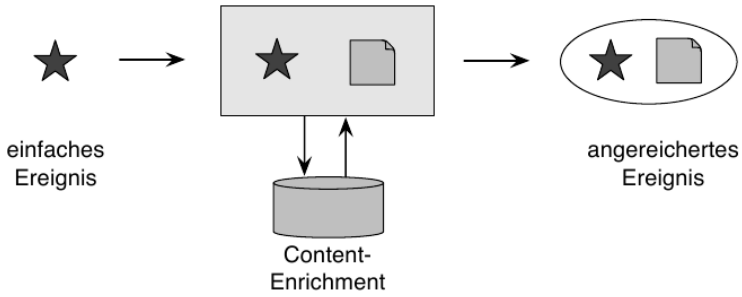


Abbildung 12: Content Enrichment [14]

3.5 Relation Detection

Nachdem die Vorverarbeitung abgeschlossen ist, soll das System aufbauend auf den extrahierten Nutzdaten Relationen detektieren. Wie bereits geschildert (vgl. Abschnitt 3.1.3), unterteilt sich das System in zwei Bereiche: einer *Event-Processing-Komponente* sowie eine *Data-Mining-Komponente*.

3.5.1 Confidence

Die Systemarchitektur von RDAS sieht vor, dass die Relationsdetektion ein mehrstufiger Prozess ist, bei welchem verschiedenartige Faktoren *(Analyzer)* im Falle einer erkannten Relation jeweils einen Wahrscheinlichkeitswert dieser definieren *(confidence)* [19]. Die einzelnen Werte sollen hierbei mittels einer Konfigurationsdatei parametrisierbar sein, der Domänenexperte soll auf diese Weise des Weiteren einen Schwellenwert festlegen können. Dieser kann anschließend genutzt werden, um nur Relationen mit einer ausreichend hohen Signifikanz auszugeben, das heißt, Relationen mit einem niedrigen confidence-Wert werden gefiltert.

66 Da innerhalb des zu konzipierenden Systems der Fokus jedoch auf der Detektion von Relationen liegt, soll dies nur rudimentär realisiert, bzw. entsprechende Mechanismen bereitgestellt werden.

3.5.2 Event Pattern Analyzer

Der Event Pattern Analyzer stellt eine der Hauptkomponenten des Systems dar, welche durch die von einen Domänenexperten zu spezifizierenden Konfigurationsdateien parametrisiert werden können soll. Diese beinhalten Beschreibungen von zu überwachenden Ereignismustern innerhalb der eingehenden Informationsströme. Innerhalb einer Vorevaluation wurden einige potentielle Ereignismuster definiert und deren spezifische Merkmale untersucht, grundsätzlich besteht bezüglich der Frage, in welcher Form bzw. Format die Ereignismuster definiert werden sollen, ein Zielkonflikt zwischen Ausdrucksstärke und Verständlichkeit. Einerseits sollen die Ereignismuster möglichst einfach und intuitiv definierbar sein, anderseits muss sichergestellt sein, dass dennoch alle relevanten Muster auch innerhalb einer entsprechenden Sprache realisierbar sind. Ein naiver Ansatz welcher nur einfache Vergleiche der entsprechenden Attribute[67] unterstützte, zeigte hierbei schnell, dass dies bei weitem nicht ausreichend ist – die Möglichkeit beispielsweise logischen Ausdrücke formulieren zu können muss ebenfalls unterstützt werden.

3.5.2.1 Virtuelle Events

Eine Analyse potentieller Ereignismuster zeigte, dass das in Abschnitt 2.6.1 beschriebene Prinzip der sogenannten *virtuellen Events* auch innerhalb von RDAS eine wichtige Rolle spielt. Nachfolgend sollen einige der ermittelten Konstellationen beschrieben werden:

- Es ist möglich, dass zwei oder mehr Nachrichten bezogen auf ein Ereignismuster in Relation zueinander stehen, jedoch die Relation für den Endnutzer nicht verständlich ist, beispielsweise wenn diese sehr komplex ist oder die Nachrichten schwer verständlich sind bzw. abstrakte technische Begrifflichkeiten verwenden. In diesem Fall kann es nützlich sein, wenn ein zusätzliches Event generiert wird, dessen Text beliebig formuliert werden kann.
- Dies kann im Spezialfall auch dann sinnvoll sein, wenn eine Änderung der Originalnachricht nicht erwünscht ist *(Annotierung)*, aber dennoch Umformulierungen stattfinden sollen. In diesem Fall kann ein *virtuelles Event* emittiert werden, welches jedoch exakt ein *reales Event* repräsentiert, jedoch keine Relation.

Dies soll anhand des folgenden Beispiels fiktiver Sensorenmeldungen illustriert werden.

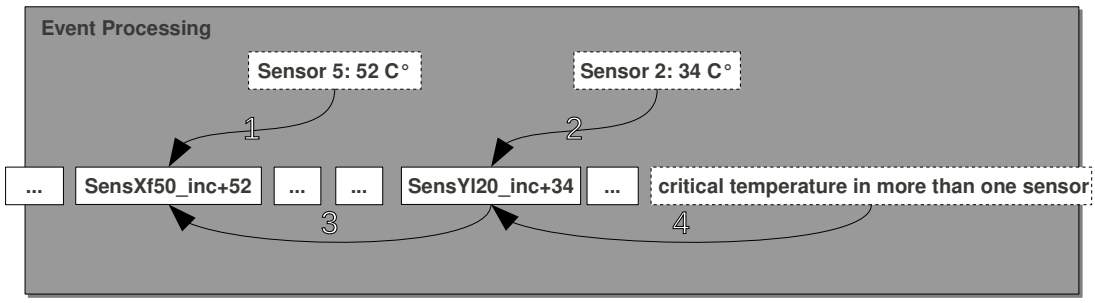


Abbildung 13: Abstraktes Beispiel virtuelle Events und Relationen

67 Beispielsweise Autor, Datum oder Titel einer Nachricht.

Die einzelnen Events laufen durch die Event-Processing-Komponente von rechts nach links, entsprechend deren Auftrittszeitpunkt. Die systeminternen Sensorenmeldungen sind hierbei mit durchgängigen Linien dargestellt, virtuelle Events mit unterbrochenen Linien. Das System soll einerseits ermöglichen, dass abstrakte bzw. schwer verständliche Events durch zusätzliche virtuelle Events repräsentiert werden können, indem die virtuellen Events mit den realen Events in Relation gesetzt werden, siehe 1 und 2. Erkennt das System nun eine kritische Temperaturveränderung, so können die ersten beiden Sensorenmeldungen in Relation gesetzt werden, zusätzlich ist aber wünschenswert, dass ein virtuelles, besser verständliches Event emittiert wird, welches automatisiert mit den verursachenden Events in Relation gesetzt wird, siehe 3 und 4.

3.5.2.2 Erweiterung der EPL

Trotz dass die Esper EPL sehr mächtig ist, reicht diese dennoch für sich genommen für die Anforderungen nicht vollständig aus. Der Domänenexperte kann mit dieser sehr detailliert Ereignismuster definieren, jedoch stellt sich die Frage, wie und welche Nachrichten anschließend in Relation zu setzen sind. Vereinfacht ausgedrückt, entspricht ein EPL Statement einer bedingten Anweisung *(wenn → dann)*. Ein Ereignismuster entspricht hierbei der Bedingung, offen bleibt, wie die Handlungsanweisung definiert werden kann, die dem System mitteilt, welche Relationen zu generieren sind. Hierbei erschienen zunächst drei Konzepte sinnvoll:

- **Automatisiert** Wird ein Ereignismuster aktiv, so extrahiert das System selbstständig alle Nachrichten, welche innerhalb des entsprechenden EPL-Ausdrucks Bestandteil dessen waren und setzt diese in Relation zueinander.
- **Semi-automatisch** Der Domänenexperte muss innerhalb des EPL-Ausdrucks explizit angeben, welche Nachrichten des Ausdrucks in Beziehung zueinander gesetzt werden sollen. Diese Rückgabewerte (Nachrichten) werden anschließend vom System miteinander in Beziehung gesetzt.
- **Manuell** Die EPL stellt Mechanismen bereit, welche es ermöglichen, dass aktiv in den Datenstrom mittels deklarativen Ausdrücken[68] eingegriffen werden kann. Der Domänenexperte kann somit selbstständig Relation generieren.

Diese Varianten unterscheiden sich hinsichtlich Komplexität bzw. Anforderungen an den Domänenexperten und deren Ausdrucksstärke, was durch die gewählten Bezeichner *(semi/automatisiert, manuell)* intuitiv verdeutlicht werden soll. Variante 1 erschien zunächst ausreichend, weist jedoch bei komplexen Ausdrücken Unzulänglichkeiten auf: stets jede Nachricht eines Ausdrucks in Relation zu setzen ist zu stark vereinfacht, beispielsweise wenn ein Ereignismuster für das Verständnis (bzw. dem Nutzer) weniger relevante Nachrichtenereignisse einschließt. Variante 3 leidet nicht unter dieser Problematik und wäre am flexibelsten, der Domänenexperte muss hierfür aber sehr umfassende Kenntnisse des Systems haben, in Folge dessen steigt auch die Fehleranfälligkeit. Aus diesem Grund soll die in Variante 2 skizziere Zwischenlösung realisiert werden, bei welcher der Domänenexperte selbstständig entscheidet, welche Nachrichten in Relation gesetzt werden.

68 Die Esper EPL kennt hierfür unter anderem UPDATE und INSERT Ausdrücke, ähnlich SQL.

Eine weitere Anforderung entsteht aus dem Umstand, dass es nicht ausreichend ist, nur die Verknüpfung der Nachrichten (welche Nachrichten eines Ereignismusters in Relation zu setzen sind) zu definieren – es muss ferner auch ein Bezeichner sowie Confidence-Wert für die Relation definierbar sein. Dies ist wichtig, da der Nutzer eine Information erhalten muss, warum eine Relation gesetzt wurde bzw. wie diese bezeichnet wird. Eine erste intuitive Variante bestand darin, diese Felder statisch zu definieren, das heißt, der Domänenexperte gibt explizit einen fixen Bezeichner vor. Komplexere Beispiele zeigten aber, dass es wünschenswert erscheint, *dynamische Bezeichner* für Relationen definieren zu können. Dynamisch bedeutet in diesem Zusammenhang, dass Daten aus den Nachrichten des aktivierten Ereignismusters in die generierte Relation einfließen können. Das Beispiel in Listing 9 soll dies verdeutlichen:

```
1   WHEN person1.istEinkaufen = true AND person2.istEinkaufen = true
2   AND
3   person1.ort = person2.ort
4   WITHIN 2 hours
5   ACTION create relation person1 TO person2 relation.title = 'Kauften am
6   '+person1.istEinkaufen.datum+' gleichzeitig in '+person1.istEinkaufen.ort+' ein'
```

Listing 9: Abstraktes EPL-Beispiel mit dynamischen Relationsbezeichner

Eine statische Relation könnte im Vergleich dazu nur „Kauften am gleichen Ort ein“ als Information beinhalten, der dynamische Ansatz dagegen integriert auch die konkrete Ausprägung. Bezüglich des Confidence-Wertes erscheint es gegenwärtig fraglich, inwiefern ein Domänenexperte ein (möglicherweise) sehr detailliertes Ereignismuster definieren sollte, welches anschließend jedoch von ihm mit einem niedrigen Confidence-Wert annotiert wird. Dennoch sollte diese Option bei Bedarf zur Verfügung stehen, auch im Hinblick auf Erweiterungsmöglichkeiten.

3.5.3 Thread Analyzer

Die in Abschnitt 2.5.2.2 vorgestellten Thread-Detection-Mechanismen sollen ebenfalls innerhalb des Systems berücksichtigt werden, dies betrifft im Wesentlichen Konversationsstrukturen aus Mails sowie Foren. Eine Analyse des phpMyAdmin-Datensatzes sowie Vergleiche mit beispielsweise Mailarchiven aus *Alioth*[69] [45] zeigte, dass eine Auswertung der E-Mail-Header-Informationen nicht als gegeben vorausgesetzt werden kann, in Folge dessen stehen diese Daten nicht zur Verfügung bzw. sollen in dieser Komponente nicht berücksichtigt werden, da E-Mail-Nachrichten gewissermaßen eine konkrete Ausprägung der abstrakten Prinzipien darstellen.

```
1   https://www.phpbb.de/community/viewtopic.php?f=87&t=218901#p1249757
2   https://www.phpbb.de/community/viewtopic.php?f=87&t=218901#p1249760
```

Listing 10: Delta-Analyse in URL

Der Thread Analyzer soll sich daher auf folgende Merkmale konzentrieren: Zitate, Titel[präfixe] sowie die (zumindest innerhalb von Sourceforge-Projekten relevante) Möglichkeit von zusammengefügten Nachrichten einer Mailinglisten mittels Trennzeichen. Erste Konzepte sahen auch eine Analyse der URL vor, da diese ein eindeutiges Merkmal

69 http://alioth.debian.org/ [17.11.11]

darstellt und Strukturinformationen *(Thread)* enthalten können. Ein naiver Ansatz bestand darin, mittels des *Levenshtein-Distanz-Algorithmus* ähnliche URLs zu detektieren, um so eine mögliche Elternachricht erkennen zu können, wenn deren URL einen entsprechend kleinen Distanzwert besitzt. Dies stellte sich jedoch als zu fehleranfällig heraus, da auch eine minimale Distanz, insbesondere innerhalb komplexer Identifikatoren, eine völlig andere Ressource und damit Elternachricht adressieren kann. Eine weiteres Konzept sah vor, dass der Domänenexperte mittels regulärer Ausdrücke die Struktur der URLs im Detail definiert, so dass das System anschließend entsprechende Relationen setzen kann. Dies ist prinzipiell möglich und zielführend, erschien jedoch in Anbetracht der notwendigen Detailkenntnisse als zu speziell, da es fraglich ist, ob der Domänenexperte die URL-Struktur jeglicher Quellen händisch analysieren kann. In einem weiteren Konzept wurde untersucht, inwiefern sich durch Bildung des Deltas zwei URLs eines Threads vergleichen lassen, dargestellt in Listing 10. Im Ergebnis kann der variable Anteil entfernt und der extrahierte Basisanteil fortan für Vergleiche genutzt werden, so dass weitere Nachrichten des gleichen Threads detektiert werden. Dieser Ansatz scheiterte jedoch daran, dass auch hier die Fehleranfälligkeit hoch ist, beispielsweise wenn die URLs jede Nachricht mit einem einzigen Identifikator kodieren, so dass der Basisanteil sehr gering wird und anschließend jede Nachricht positiv in Relation gesetzt werden würde. Aus diesen Gründen soll sich der Thread Analyzer auf die extrahierten *chunks,* respektive die zitierten Bereiche, konzentrieren. Hierbei soll bei jeder eintreffenden Nachricht über alle extrahierten Bereiche (Abschnitt 2.5.2.2) iteriert und mittels Vergleichsoperationen ermittelt werden, inwiefern der entsprechende Text bereits im Korpus existiert ist. Im positiven Fall soll RDAS eine Relation zwischen der Treffernachricht und aktueller Nachricht generieren. Bei längeren Konversationen kann dieser Ansatz logischerweise mehrere Treffer zurück liefern, es ist jedoch ausreichend, eine Relation stets nur zum neusten Treffer (Nachricht) zu setzen, da dieser Mechanismus entsprechend bei älteren Nachricht bereits aktiv war und Relationen generierte *(Transitivität).* Innerhalb einer Vorevaluation wurde untersucht, inwiefern ein *fuzzy*-Vergleich zielführend ist, das heißt, es werden nicht die exakten Zeichenketten vergleichen bzw. gesucht sondern nur die entsprechenden Wortmengen (*bag of words model* [61]). Die Ergebnisse zeigten jedoch, dass die Anzahl von falsch-positiv-Resultaten zu hoch ist. Aus diesem Grund wurde untersucht, inwiefern exakte Vergleiche bessere Resultate liefern können, problematisch erwies sich hierbei jedoch, dass einzelne Nachrichten verschiedenartige Zeichenkodierungen hatten bzw. nicht ASCII Zeichen in einigen Fällen falsch abgebildet wurden [70]. In Folge dessen ergaben sich zahlreiche falsch-negativ-Ergebnisse, da ein exakter Vergleich die Originale nicht mehr erkannte. Innerhalb von RDAS soll daher ein Mechanismus realisiert werden, welcher die Nachrichtentext für Vergleichsoperationen auf eine einheitliche Zeichenbasis abbildet bzw. umwandelt.

Innerhalb des Abschnitts 2.5.2.2 wurde des Weiteren auf die Möglichkeit eingegangen, die Thread-Strukturen anhand der Titelinformationen einer Nachricht extrahieren zu können. Eine Analyse der Datensätze zeigte, dass die angesprochene Auswertung der Präfixe hierfür zielführend erscheint, da entsprechende Bezeichner konsistent verwendet werden. Auch hier wurde die Möglichkeit eines unscharfen Vergleichs mittels des dargestellten *Dice-Koeffizienten* untersucht, es zeigt sich aber, dass dies zu zahlreichen falsch-positiv-Ergebnissen führt. Dies wurde unter anderem dadurch verursacht, dass einige Anwendungen große Bereiche der Titelzeichenketten statisch generieren und diese in

70 Beispielsweise wenn der zitierte Bereich den Autor „Michal Čihař“ enthält, dies aber in der Nachricht unter „Michal iha“ oder „Michal ihaY“ falsch codiert wurde.

Folge dessen sehr häufig auftreten. Des Weiteren erschien ein derartiger Vergleich tendenziell redundant mit der geplanten Funktionalität des *Content Analyzer* (vgl. Abschnitt 3.5.4).

3.5.4 Content Analyzer

Ziel dieser Komponente soll die Erkennung von Relationen basierend auf dem Inhalt der Nachrichten sein. Hierbei soll auf die innerhalb der Vorerarbeitung extrahierten Schlüsselwörter zurück gegriffen werden und basierend auf deren Vorkommen eine Analyse mittels Methodiken des *Information Retrievals* stattfinden, beispielsweise durch Berechnung der *Kosinus-Ähnlichkeit* [57]. Im Ergebnis soll RDAS zwei Nachrichten in Relation setzen, wenn diese eine hohe Schnittmenge von Schlüsselwörtern besitzen. Dabei gilt es jedoch zu untersuchen, welcher Schwellenwert für die Ähnlichkeitsberechnung an dieser Stelle zielführend ist, das heißt, wann ein Nutzer zwei Nachrichten aufgrund dieser Analyse auch als ähnlich akzeptiert. Als weiteres Problem zeigte sich durch eine Vorevaluation, dass innerhalb des phpMyAdmin-Datensatzes zahlreiche Nachrichten automatisiert generiert werden, beispielsweise Reports von Trackern, und in Folge dessen zu großen Teilen stets identisch sind[71]. Dies hat zur Folge, dass entsprechende Nachrichten praktisch immer ein sehr hohes Ähnlichkeitsmaß zu vorangegangen Report-Nachrichten generieren. Für den Endnutzer sind derartige Relationen aber nur begrenzt nützlich, aus diesem Grund soll das System diese erkennen und ggf. zusätzlich abwerten, beispielsweise durch eine höhere Gewichtung der entsprechenden *Inversen Dokumentfrequenz.*

3.5.5 User Session Analyzer

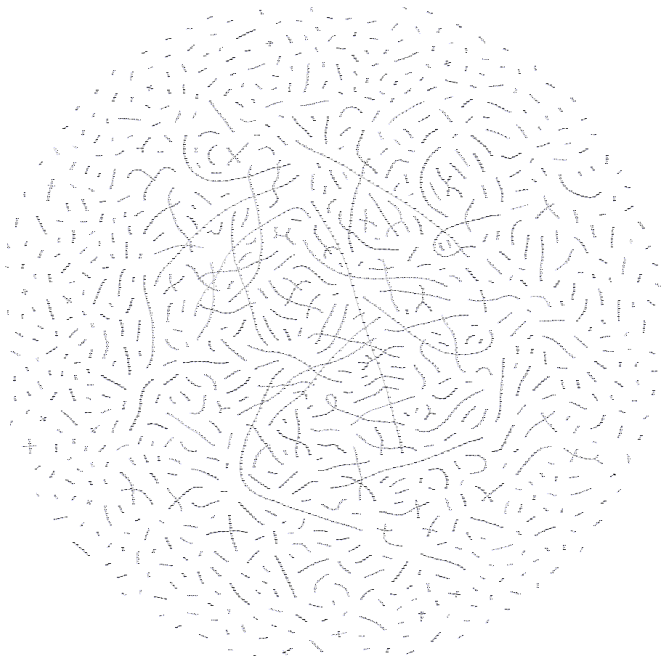


Abbildung 14: Zeitliche Relationen innerhalb des phpMyAdmin-Datensatzes

Das Ziel des *User Session Analyzers* ist die Generierung von Relationen zwischen Nachrichten, wenn diese innerhalb eines zu definierenden Zeitraumes durch einen Nutzer emittiert wurden *(Session)*. Hierfür soll ebenfalls auf die innerhalb der Aliaserkennung

71 Das Delta besteht meist aus einzelnen Revisionsnummern bzw. ähnlichen Bezeichnern.

(vgl. Abschnitt 3.4.3) extrahierten Identitäten zurück gegriffen werden. Auf diese Weise entsteht eine zeitliche Relationsebene, welche andere Relationen verstärken kann oder, je noch Systemumgebung, auch eigenständig existieren kann. Erwartungsgemäß konnte innerhalb einer Vorevaluation so bereits erkannt werden, dass einzelne Nutzer längerfristig aktiv sind (Abbildung 14, jeder Knoten entspricht einer Nachricht, lagen diese zeitlich eng beieinander und wurden vom gleichen Autor generiert, entspricht dies einer Kante). Es sollte jedoch innerhalb einer Evaluation überprüft werden, inwiefern eine zeitliche Relation für sich genommen durch Nutzer als ausreichend akzeptiert wird.

3.5.6 Link Analyzer

Das Konzept des Link Analyzers sieht vor, die in Abschnitt 2.5.2.1 geschilderten Beziehungsmerkmale hinsichtlich es Auftretens von Hyperlinks auszuwerten. Hierbei kann grundlegend zwischen zwei Formen unterschieden werden:

1. Nachricht A enthält einen direkten Link auf Nachricht B (Typ 1)
2. Nachricht A und B enthalten beide einen Link auf Ressource R (Typ 2)

Innerhalb der Vorverarbeitung wurden bereits alle Vorkommen von Links einer Nachricht extrahiert, welche nun innerhalb des Link Analyzers ausgewertet werden sollen. Zunächst wurde durch eine Vorevaluation überprüft, inwiefern sich eine URL-Normalisierung auf die Menge der detektierten Relationen auswirkt. Hierbei zeigte sich, dass innerhalb des phpMyAdmin-Datensatzes eine aufwändige, jedoch weitestgehend semantikerhaltende, Normalisierung [94] keine nennenswerten Vorteile bringt. Eine händische Analyse machte deutlich, dass meist Unterschiede im Protokollteil einer URL die eigentliche Ursache für Differenzen sind. Aus diesem Grund wurde eine invasivere Methode evaluiert, welche den gesamten Protokollanteil entfernt sowie die gesamten Zeichenketten auf Kleinbuchstaben abbildete. Mittels dieser Variante konnte die Anzahl der detektierten Relationen bereits merklich erhöht bzw. sogar verbessert werden.

Grad der Normalisierung	**Typ der Relation**	**Anzahl detektierter Relationen**
Keine Normalisierung	Typ 1	316
	Typ 2	43.734
Entfernung des Protokolls + Kleinbuchstaben	Typ 1	544
	Typ 2	44.013
Komplexe Normalisierung [94]	Typ 1	316
	Typ 2	44.264

Tabelle 2: Detektierte Link-Relationen in Abhängigkeit der Normalisierung

Eine Analyse der aufgetretenen URLs innerhalb der Nachrichten zeigte des Weiteren, dass diese tendenziell einer *Zipf-Verteilung* entsprechen, das heißt, einige wenige URLs

kommen extrem häufig vor. Dies hat zur Folge, dass Nachrichten, welche einen Link auf eine sehr häufig vorkommende Ressource besitzen, potentiell zu sehr vielen anderen Nachrichten in Relation stehen. Um dies zu vermeiden, soll das System dynamisch erkennen, welche Relevanz eine entsprechende URL hat. Es zeigte sich, dass hierfür eine *idf*-Berechnung zielführend ist, da das *tf-idf*-Maß innerhalb langer Nachrichten unnötig abwertet, entscheidend ist prinzipiell ob ein Link innerhalb einer Nachricht existiert, es ist eher unwahrscheinlich, dass dieser mehrfach vorkommt (sollte dies der Fall sein, erhöht oder vermindert dies nicht die Aussagekraft).

RDAS soll daher mittels eines konfigurierbaren Schwellenwertes dynamisch entscheiden, wann eine Relation zwischen zwei Nachrichten zu setzen ist, in Abhängigkeit des *idf*-Wertes der aufgetretenen URL. Dieses Konzept hat jedoch einen Nachteil: es kann aufgrund des dynamischen Ansatzes passieren, dass eine URL bis zum Zeitpunkt t einen *idf*-Wert größer des Schwellenwertes besitzt, jedoch beim nächsten Vorkommen einen zu niedrigen Wert. Die Generierung der Relationen könnte dann theoretisch stets alternieren auftreten. Des Weiteren ist es prinzipbedingt möglich, dass eine URL erst eine hohe Relevanz besitzt und bei entsprechend häufigen Vorkommen später eine relativ niedrige. Dieses Problem lässt sich nur umgehen, indem das System beispielsweise in zyklischen Abständen eine Neuberechnung der Relationen vornimmt.

3.5.7 Regular Expression Analyzer

Ziel dieser Komponente soll die Relationsdetektion aufbauend auf, durch den Domänenexperten zu definierenden, Syntaxmustern *(regulären Ausdrücken)* sein. Ein erstes Konzept sah vor, diese Komponente mittels des *Event Pattern Analyzers* zu realisieren, indem eingehende Nachrichten auf entsprechende reguläre Ausdrücke überwacht werden. Eine Vorevaluation zeigte jedoch, dass dies sehr schwierig ist, da nicht nur das reine Vorkommen eines Syntaxmusters entscheidend ist, sondern auch dessen konkreter Wert. Soll beispielsweise nach Bug Reports gesucht werden, könnte folgendes Syntaxmuster eingesetzt werden: „(bug|patch|item) ?(report){0,1} ?#?([0-9]{3,})“. Wird dies in Nachricht *A* detektiert, beispielsweise aufgrund des Vorkommens von „..bug #123..“ und anschließend in Nachricht *B*, jedoch aufgrund des Vorkommens von „..bug #456..“, so stehen diese Nachrichten definitiv nicht in Relation. Versuche dies mittels einer geeigneten EPL zu beschreiben, waren jedoch nicht zielführend, da reguläre Ausdrücke nur innerhalb der *WHERE-Bedingungen* formuliert werden konnten – deren konkrete Werte jedoch nicht zwischen Nachrichten für Vergleiche zur Verfügung stehen. Es ist zudem nicht ausreichend, die zurück gelieferten Ergebnisse zu vergleichen, diese könnten beispielsweise in Nachricht C „ ..bug #123..“ und in Nachricht D „ ..bug report 123..“ lauten. Aus diesem Grund muss das System Abschnitte der Ergebnisse (Zeichenketten) der regulären Ausdrücke extrahieren und erst basierend auf diesen Relationen generieren, im beschriebenen Szenario exemplarisch indem der resultierende Wert „123“ als Merkmal[72] der Relation verwendet wird (vgl. Abbildung 15).
Reguläre Ausdrücke bieten hierfür einen sogenannten *Backreference-Mechanismus* [33] an, welcher es erlaubt, Teile eines Resultates zu extrahieren. Das System hat jedoch keine Kenntnisse darüber, welcher Teil dies tatsächlich ist, in Folge dessen soll dies ebenfalls durch den Domänenexperten spezifiziert werden. Im dargestellten Szenario wäre dies beispielsweise der dritte Abschnitt innerhalb des regulären Ausdrucks: ([0-9]

72 Der Domänenexperte muss dabei sicherstellen, dass diese Merkmale nicht zu generalisierend sind, beispielsweise wäre eine Relationsdetektion basierend auf „123“ vermutlich nicht zielführend.

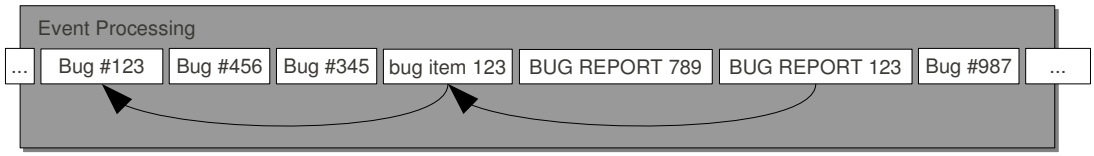


Abbildung 15: Relationen durch reguläre Ausdrücke in Abhängigkeit der Werte

{3,}) bzw. die konkrete Bug-ID. Es wäre theoretisch denkbar, dass das System selbstständig lernt, welche Teile eines Syntaxmusters sich überdecken. Konzeptionell ist hierbei aber fraglich, warum ein Domänenexperte den zu extrahierenden Abschnitt bei der Definition nicht auch direkt angeben sollte, da diese Angabe als trivial einzustufen ist.

3.5.8 Social Network Utilization

Nachdem innerhalb der Social-Network-Analysis-Komponente (vgl. Abschnitt 3.4.6) ein Graph sozialer Beziehungen generiert wurde, sollen innerhalb der Social-Network-Utilization-Komponente diese Informationen in detektierte Relationen einfließen bzw. verstärken, aus diesem Grund erfolgt dieser Schritt in der Abfolge als letztes. Das System soll hierbei bei einer eintreffenden (detektierten) Relation deren Autoren extrahieren und innerhalb des Graphen ermitteln, ob deren repräsentierende Knoten eine verbindende Kante besitzen. Ist dies der Fall, wird deren Gewicht als Multiplikator für den bereits ermittelten Confidence-Wert verwendet. Offen bleibt hierbei, in welchem Maß diese Verstärkung zu erfolgen hat, dies kann letztendlich nur durch eine Nutzerstudie bzw. Erfahrungswerte ermittelt werden. Innerhalb des Kontextes der Arbeit soll demnach zunächst nur eine entsprechende Information an den Endnutzer weitergeleitet werden, beispielsweise durch farbliche Hervorhebungen einer Relation, wenn diese aufbauend auf dem sozialen Graphen höher gewichtet wurde.

3.6 Activity Streams

Ein wesentliches Ziel des Systems ist die Generierung eines Aktivitätsstromes laut der Activity-Stream-Spezifikation (vgl. Abschnitt 2.2.2), die nachfolgenden Abschnitte beschreiben die entsprechenden Komponenten des zu konzipierenden Systems.

3.6.1 Analyse und Detektion von Aktivitäten

Um aus einer Nachricht beliebiger Struktur die zugrunde liegende Aktivität ableiten zu können, muss prinzipiell eine umfassende semantische Analyse (NLP) erfolgen. Innerhalb einer Vorevaluation wurden verschiedene Testreihen unter Einbeziehung von NLP-Systemen durchgeführt, der konzeptionelle Gedanken lautete hierbei wie folgt: mittels eines geeigneten *POS-Taggers* werden Substantive und Verben extrahiert und lemmatisiert, in einem weiteren Schritt werden diese gegen die innerhalb der Spezifikation erlaubten Ausprägungen verglichen. Dieser naive Ansatz scheitert jedoch bereits bei mehrfachen Vorkommen gültiger Terme und erscheint insbesondere bei längeren Texten völlig ungeeignet. Grundsätzliche Voraussetzung ist zudem, dass der Nachrichtentext auch Information über die zugrunde liegende Aktivität beinhaltet, dies kann aber nicht als gegeben vorausgesetzt werden. Ferner zeigte eine händische Analyse des Datensatzes, dass eine NLP-Semantikanalyse nur basierend auf dem Nachrichtentext, das heißt ohne zusätzli-

che Informationen, oft kaum möglich ist. Führt Autor A den Kommentar „Last Friday night I went out with my friends“ zu einem Forum „weekend“ hinzu, so müsste die korrekte Aktivität lauten: „Author A posted comment 'Last Friday...' to forum 'weekend' “.

Das heißt, es existiert eine *semantische Aktivität*, welche mittels NLP aus der Nachricht extrahiert werden könnte und eine *syntaktische Aktivität*, welche sich aufbauend auf den Metadaten und der Struktur ergibt. Die Activity-Stream-Spezifikation benötigt jedoch im Normalfall nur letztere, wie anhand des Beispiels gezeigt wurde.

Des Weiteren wurde deutlich, dass mittels entsprechendem Wissen über die Quellen und deren potentielle Nachrichten relativ einfach aktivitätsbezogene Nachrichten generiert werden können, beispielsweise indem bekannt ist, dass der Informationsstrom stets Kommentare beinhaltet. Im Gegensatz dazu kann ein NLP-System nur basierend auf dem Nachrichtentext nur schwer bis gar nicht erkennen, dass es sich um einen Kommentar in einem Forum handelt – dies kann prinzipiell nur dann erfolgen, wenn die Nachricht selbst bereits eine Beschreibung der Aktivität darstellt. Aufgrund dieser Erkenntnisse soll das zu konzipierende System daher nicht selbstständig die einer Nachricht zugrunde liegende Aktivität erkennen können, sondern die in Abschnitt 3.4.7 beschrieben Annotierung hierfür genutzt werden.

3.6.2 Erweiterung des Activity-Stream-Formates

Der von dem zu konzipierenden System generierte (aggregierte) Nachrichtenstrom muss die detektierten Relationen den abonnierenden *(Subscriber)* Anwendungen zur Verfügung stellen, damit diese innerhalb weiterer Verarbeitungsschritte entsprechende aufbereitete Informationen darstellen können. Grundsätzlich soll der Activity Stream lediglich alle notwendigen Informationen liefern, auf welche Art und Weise die Relationen durch externe Anwendungen visualisiert bzw. verarbeitet werden, soll nicht Bestandteil von RDAS sein. Das Activity-Stream-Format sieht für diese Modellierung von Nachrichtenrelationen jedoch keine Mechanismen, respektive dedizierte Felder, vor. Innerhalb der Dokumentation [6] wird lediglich auf ein sogenanntes *source*-Element im Rahmen der *Base Extension Properties* eingegangen[73]:

```
1    "source": {
2         "objectType": "collection",
3         "displayName": "Joe's Photo's",
4         "url": "http://example.org/joes/photos"
5    }
```

Listing 11: source element

Dieses soll dazu dienen, die Quelle eines Objektes bzw. einer Aktivität definieren zu können. Obwohl dies auf den ersten Blick adäquat erschien, soll in RDAS dennoch eine andere Lösung realisiert werden, nachfolgend die hierfür ausschlaggebenden Gründe:

- Die Semantik des Elementes *(source)* entspricht streng genommen nicht der einer verweisenden Relation.

73 http://activitystrea.ms/head/activity-schema.html#anchor3 [31.10.11]

- Die Dokumentation (von *source*) ist äußerst rudimentär und lässt beispielsweise offen, inwiefern mehrere Verweise enthalten sein dürfen.
- Eine Erweiterung des Atom- bzw. Activity-Stream-XML-Namensraumes ist stets möglich und vermeidet eine Zweckentfremdung vorhandener Konstrukte. Des Weiteren ist eine Abwärtskompatibilität hiermit gesichert.
- Auch das *source*-Element müsste zwingend erweitert werden, beispielsweise um die Gewichtung einer Relation abbilden zu können,

```
1  <xs:complexType name="feedType">
2  ...
3  <xs:element name="relations" type="rel:relations" minOccurs="0" maxOccurs ="unbo..
4  ...
5  <xs:complexType name="relations">
6  <xs:sequence>
7  <xs:complexType name="relation">
8    <xs:sequence>
9      <xs:element name="link" type="linkType"/>
10     <xs:element name="relationType" type="xs:string"/>
11     <xs:element name="confidence" type="xs:decimal"/>
12   </xs:sequence>
13 </xs:complexType>
14 </xs:sequence>
15 </xs:complexType>
```

Listing 12: XSD der Activity-Stream-Erweiterung

Dem Modell aus Abschnitt 2.5.1 entsprechend, besteht eine Relation aus einem Tripel: ein Verweis auf eine weitere Nachricht, der Typ der Relation sowie deren Wahrscheinlichkeitswert. Eine Nachricht A kann dabei beliebig viele Relationstripel besitzen. Eine entsprechende XSD[74] ist ist in Listing 12 dargestellt. Zur erkennen ist hierbei, dass für die Modellierung der Kanten das bereits vorhandene *link*-Element genutzt werden soll, streng genommen stellt dieses keinen eindeutigen Identifikator dar, die meisten Systeme nutzen jedoch eindeutige Adressen. Ein derartiger Verweis ist für die abonnierenden Systeme ausreichend, um Relationen verarbeiten zu können: sollte eine Nachricht sehr alt und dem Endsystem unter Umständen gar nicht bekannt sein, so müsste es diese initial nachladen. Die Alternative hierzu, jegliche Verweise (Nachrichten) bereits in die Nachrichten selbst einzufügen, würde dagegen insbesondere bei tiefen Verweishierarchien zu einem zu großen Datenaufkommen führen.

3.7 Visualisierung

Das zu konzipierende System soll neben der eigentlichen Detektion der Relationen und der Transformation in einen Activity Stream zusätzlich auch eine prototypische grafische Benutzeroberfläche bereitstellen. Diese soll im Wesentlichen folgen Aufgaben erfüllen können:

74 http://exyus.com/xcs/tasklist/source/?f=put_atom.xsd wurde hierbei als Vorlage genutzt [31.10.11]

- Ausgabe eines Activity Streams, welcher alle Nachrichten chronologisch geordnet anzeigt
- Visualisierung der Relationen zwischen den Nachrichten untereinander
- Ausgabe der Detail- und Metainformationen einer Nachricht, beispielsweise deren Inhalt, Datum oder Titel.
- Visualisierung der Korrelationen

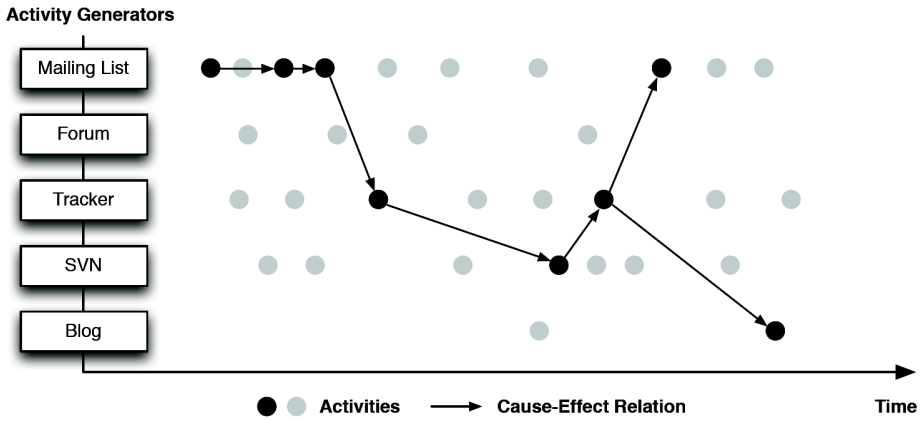


Abbildung 16: Abstrakte Visualisierung von Relationen und Korrelationen [49]

Bei der Darstellung des Activity Streams kann im Wesentlichen die verbreitete vertikale Darstellung wie sie beispielsweise innerhalb von *Facebook* (vgl. Abschnitt 2.2) zu finden ist, verwendet werden. Des Weiteren erscheint eine grafische Hervorhebung der SPO-Struktur (vgl. Abbildung 1) ebenfalls sinnvoll. Die Ausgabe der Datei- und Metainformationen ist im Prinzip beliebig und soll daher im Rahmen der prototypischen Umsetzung nicht weiter diskutiert werden. Die Visualisierung der Relationen und Korrelationen erfordert jedoch ausführlichere Überlegungen bzw. stellt technisch höhere Anforderungen und soll daher nachfolgend detaillierter diskutiert werden. Innerhalb von [49] wurde eine abstrakte Sicht auf den beschrieben Kontext bereits vorgestellt, siehe Abbildung 16, diese soll als Grundlage für die weitere Konzeption dienen.

Thread Arcs [57] sind ein Visualisierungskonzept, welches Konversationen innerhalb von E-Mail-Nachrichten im Vergleich zu einer rein linearen Abfolge besser verdeutlichen soll. Im Wesentlichen besteht das Konzept darin, Nachrichten mittels grafischer Bögen *(Arc)* zu verbinden, um auf diese Weise deren Beziehungen (beispielsweise Antwortstrukturen) abbilden zu können. Der entstehende Relationengraph (vgl. Abschnitt 2.5.1) kann somit analog abgebildet und im Rahmen des zu konzipierenden Systems auf alle relevanten Relationsarten erweitert werden (zusätzliche zu den in der Quelle betrachteten Konversationsstrukturen). Damit Korrelationen bzw. Ereignisketten deutlicher hervor treten, erscheint jedoch eine Erweiterung der Bögen zu gerichteten Kanten sinnvoll (indem zusätzlich Pfeilspitzen eingesetzt werden).
In [1] werden zahlreiche weitere, teilweise sehr abstrakte, Visualisierungskonzepte kurz vorgestellt. Einige dieser Vertreter verfolgen dabei sehr experimentelle Ansätze, welche im Kontext der Arbeit ungeeignet erscheinen. Ein Beispiel der vorgestellten Konzepte ist in Abbildung 18 dargestellt, soll jedoch mit Verweis auf die entsprechende Quelle nicht weiter erläutert werden.

Abbildung 17: Thread Arcs [57]

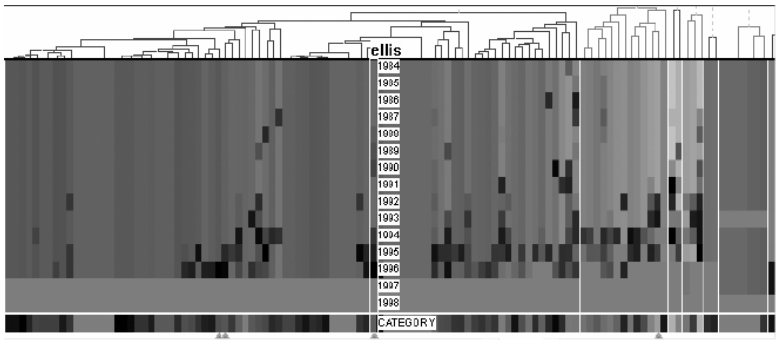


Abbildung 18: Rhythms of Relationships [1]

Fazit Da innerhalb des zu konzipierenden Systems nur eine prototypische Implementierung der Benutzeroberfläche erfolgen soll, erscheint das Prinzip der *Thread Arcs* sinnvoller, da hier wesentlich geringer technische Anforderungen an das Layout/Platzierung der Elemente existieren (da stets linear auf einer Ebene). Zukünftige Entwicklungen sollten dennoch eine komplexere Visualisierung betrachten, da die in Abbildung 16 dargestellte Sichtweise intuitiver im Kontext erscheint.

3.8 Zusammenfassung

Innerhalb des vorangegangenen Kapitels wurde die grundsätzliche Konzeption von RDAS vorgestellt. Hierbei wurde zunächst die logische Systemarchitektur mittels einer Übersicht dargestellt und anschließend einzelne Komponenten näher charakterisiert. Diese lassen sich in drei grundlegende Gruppen zusammenfassen: Import, Vorverarbeitung, Relationsdetektion sowie Ausgabe. Im Anschluss daran wurde jede Komponente des Systems im Detail erläutert, insbesondere in Hinblick auf die geplante Funktionsweise der Relationsdetektion. Daran anschließend wurden notwendige Erweiterungen der Activity-Stream-Spezifikation vorgestellt, welches eine Abbildung der Beziehungen innerhalb dieses Formats ermöglichen. Des Weiteren wurde in diesem Kontext auf die Problematik der Aktivitätsanalyse eingegangen, sowie ein möglicher Lösungsansatz präsentiert. Zum Abschluss wurden mögliche Visualisierungskonzepte kurz vorgestellt, welche innerhalb eines produktiven Systems zum Einsatz kommen könnten. Das nächste Kapitel beschreibt die Implementierung eines prototypischen Systems, welche die in Kapitel 3 skizzierten Konzepte realisieren soll.

4 Implementierung

Dieses Kapitel beschreibt die Umsetzung der in den vorherigen Abschnitten ausgearbeiteten Konzepte, dabei sollen aufgetretene Probleme und deren Lösungen beschrieben und die einzelnen entstandenen Komponenten näher charakterisiert werden.

4.1 Technologiebasis

RDAS wurde mittels Java implementiert, insbesondere im Backendbereich kann dies als die am weitesten verbreitete [82] objektorientierte Programmiersprache angesehen werden, welche für zahlreiche Aufgabenstellungen bereits existierende Lösungen liefert. Des Weiteren existiert für Java eine sehr große Zahl frei verfügbarer (Open Source) Bibliotheken, welche ohne Einschränkungen[75] in eigene Projekte eingebunden und bei Bedarf angepasst werden können. Neben diesen existiert auch eine sehr große Palette an ausgereiften Entwicklungsumgebungen und Werkzeugen, welche den gesamten Workflow abdecken - allen voran die IDE *Eclipse*[76], welche als De-Facto-Standard angesehen werden kann. Mittels entsprechender Application Server, wie beispielsweise *Tomcat*[77], können Anwendungen auf einfachem Wege auch Schnittstellen im Bereich des Webs anbieten bzw. Webanwendungen bereitstellen. GWT (vgl. Abschnitt 4.2.2) liefert für diesen Zweck alle erforderlichen Komponenten bereits standardmäßig mit.

4.2 Bibliotheken und Frameworks

Im Rahmen von RDAS kamen zahlreiche frei verfügbare Java-Bibliotheken bzw. Frameworks zum Einsatz, welche sich je nach Einsatzgebiet in die grundlegenden Bereiche Backend und Frontend unterteilen lassen. Nachfolgend sollen die wichtigsten Pakete bzw. Projekte kurz, auch bereits im Kontext von RADS, charakterisiert werden. Ausführlichere Erläuterungen erfolgen in den nachgelagerten Abschnitten, sofern dies für das Verständnis erforderlich ist.

4.2.1 Backend

Apache Commons[78] ist eine Sammlung von verschiedenartigen Java-Bibliotheken, welche das Ziel haben, weit verbreitete bzw. häufige Problemstellungen durch universell einsetzbare Klassen bzw. Methoden zu lösen, beispielsweise durch Java-API-Erweiterungen. Apache Commons besteht dabei aus zahlreichen kontextbezogenen Komponenten beispielsweise für die Verarbeitungen von Dateien (Commons IO[79]) oder Zeichenketten (Commons Lang[80]). Innerhalb von RDAS kamen insbesondere die Funktionalitäten der Language- (lang) und Input-Output-Komponente (IO) sowie DBCP (Database Connection Pools[81]) für die effiziente Verwaltung von Datenbankverbindungen zum Einsatz.

75 Unter Berücksichtigung entsprechender lizenzrechtlicher Bestimmungen.
76 http://www.eclipse.org/ [09.11.11]
77 http://tomcat.apache.org/ [09.11.11]
78 http://commons.apache.org/ [09.11.11]
79 http://commons.apache.org/io/ [09.11.11]
80 http://commons.apache.org/lang/ [09.11.11]
81 http://commons.apache.org/dbcp/ [09.11.11]

Neben diesen verwenden die eingesetzten Fremdpakete (beispielsweise *Esper*) ebenfalls weitere Commons-Bibliotheken, welche somit indirekt Bestandteil des Systems sind (dies betrifft zum Beispiel das Logging[82]).

JDBM[83] ist eine Persistenzengine für Java, welche neben Transaktionen insbesondere einfach zu verwendende B-Bäume und Hash-Tabellen anbietet. Der wesentliche Vorteil von JDBM liegt in der Tatsache begründet, dass die Speicherung der Daten nicht innerhalb des Arbeitsspeichers, sondern auf Festplatte erfolgt. Dies führt zu einer stark verbesserten Skalierbarkeit und ermöglicht, bezogen auf RADS, die Verwaltung und Verarbeitung von sehr großen Datenmengen und leistungsfähigen Operationen.

MySQL[84] kommt innerhalb von RDAS als relationales Datenbankverwaltungssystem für verschiedenartige Persistierungsaufgaben zum Einsatz und speichert bzw. exportiert in diesem Sinne jene Daten, welche durch andere Komponenten (beispielsweise *JDBM*) bereits auf Dateiebene gesichert wurden. Des Weiteren werden nicht zeitkritische Operationen (insbesondere initiiert durch das Frontend) ebenfalls mittels entsprechender SQL-Abfragen verarbeitet.

Lucene[85] ist ein durch die *Apache Software Foundation*[86] entwickeltes Suchmaschienen-Framework, welches die Entwicklung leistungsfähiger Information-Retrieval-Systeme ermöglicht. Lucene stellt dabei nicht selbst eine Suchmaschine dar, sondern liefert die grundlegenden Mechanismen mittels entsprechender Klassen und Methoden, beispielsweise für die Generierung von Indizes. Auf diesen können dann Suchanfragen ausgeführt werden, welche gewichtete Ergebnisse zurück liefern (vgl. Abschnitt 2.3.1). Innerhalb RDAS stellt Lucene eine der zentralen Kernkomponenten dar und kommt in Folge dessen bei zahlreichen Funktionalitäten zum Einsatz, beispielsweise zur Suche und Gewichtung von verwandten Nachrichten im Hinblick auf die extrahierten Stichwörter (vgl. Abschnitt 2.5.2.3).

ROME[87] stellt eine Java-Bibliothek dar, welche eine einheitliche Sichtweise auf die verschiedenartigen Feed-Formate (vgl. Abschnitt 2.1) auf Implementierungsebene ermöglicht und insbesondere die existierenden Versionsinkompatibilitäten ausblendet, indem es eine Abstraktion vornimmt. ROME bietet in diesem Sinne Funktionalitäten für den Import sowie Export von Web Feeds und wird innerhalb von RDAS als Ausgangsbasis für das Datenmodell bzw. die Abbildung von Nachrichten auf entsprechende Klassen verwendet.

Esper[88] ist ein Event-Processing-System (vgl. Abschnitt 2.6) für die Realisierung von CEP- bzw. ESP-Systemen. Es bietet eine sehr intuitive API, welche eine komfortable Entwicklung entsprechender Anwendungen ermöglicht. Ereignisse können hierbei auf Implementierungsebene in Echtzeit analysiert, gefiltert und konsumiert werden. Neben Lucene stellt Esper innerhalb von RDAS eine weitere Kernkomponente dar, insbesondere

82 http://commons.apache.org/logging/ [09.11.11]
83 http://jdbm.sourceforge.net [09.11.11]
84 http://www.mysql.com [09.11.11]
85 http://lucene.apache.org [09.11.11]
86 http://www.apache.org [09.11.11]
87 http://java.net/projects/rome/ [09.11.11]
88 http://esper.codehaus.org [09.11.11]

für die Definition und Erkennung von Ereignismustern (vgl. Abschnitt 2.5.3.2) sowie zur Annotierung von Nachrichten mit aktivitätsbezogenen Daten. Die durch Esper zur Verfügung gestellte Ereignisbeschreibungssprache kommt hierbei als wesentliches Konfigurationswerkzeug in Hinblick auf den Domänenexperten zum Einsatz.

JGraphT[89] dient der Verwaltung und Verarbeitung von Graphen und liefert entsprechende Klassen für zahlreiche mathematische Operationen über diesen, beispielsweise eine Implementierung des *Dijkstra-Algorithmus* [26]. Die Bibliothek kennt hierbei alle im Kontext relevanten Datenstrukturen: gerichtet und ungerichtete Graphen, gewichtete Kanten, DAGs und vieles mehr. Innerhalb von RDAS kommt JGraphT insbesondere für die Berechnung und Verarbeitung des *Social Graph* zum Einsatz. Des Weiteren ermöglicht JGraphT Berechnungen in Hinblick auf die Erreichbarkeit von einzelnen Nachrichten untereinander, dies wird beispielsweise eingesetzt, um Redundanzen in Relationen innerhalb von Threads zu vermeiden.

Stanford-POS-Tagger[90] ist eine frei verfügbare Java-Bibliothek, welche ein (bezogen auf die englische Sprache) „out of the box“ Part-of-Speech-Tagging liefert, indem bereits trainierte Modelle angeboten werden. RDAS verwendet dies, um Stichworte der einzelnen Nachrichten basierend auf den vorkommenden Subjekten generieren zu können. In diesem Zusammenhang kommt die, ebenfalls in der Bibliothek enthaltene, Lemmatisierung (vgl. 2.5.2.3) zum Einsatz, um entsprechende Grundformen generieren zu können.

cloning[91] stellt eine leichtgewichtige Java-Bibliothek dar, welche ein sogenanntes *Deep Cloning* (auch *Deep Copy* genannt) von Objekten ermöglicht, ohne dass diese die Java *Cloneable*-Schnitstelle[92] implementieren müssen. Um dies zu realisieren wird intern auf die *objenesis*[93] Bibliothek zurück gegriffen, welche eine stark erweiterte Introspection von Klassen ermöglicht. RDAS nutzt dies zur Generierung von sogenannte *virtuellen Nachrichten (Events)*, indem vorhandene ROME-Objekte kopiert und nur partiell modifiziert werden.

Prefuse[94] dient der Visualisierung von (beispielsweise) Graphen und kam im Kontext des zu konzipierenden Systems intensiv während der Entwicklung und als intuitiveres[95] Kommunikationsmittel zum Einsatz, um eine weitere Sicht auf die zu detektierenden Relationen erhalten zu können.

XStream[96] ist eine Bibliothek für die Serialisierung und Deserialisierung von von Java Objekten zu XML und JSON. Wie bereits im Zusammenhang mit der cloning-Bibliothek erwähnt wurde, kommt auch an hier eine Objekt-Introspection zum Einsatz, um diese ohne vorherige Definition entsprechender Schemata generieren zu können. Innerhalb von RDAS wird dies insbesondere für die Auswertung der Konfigurationsdateien eingesetzt,

89 http://www.jgrapht.org/ [09.11.11]
90 http://nlp.stanford.edu/software/tagger.shtml [09.11.11]
91 http://code.google.com/p/cloning/ [09.11.11]
92 http://download.oracle.com/javase/6/docs/api/java/lang/Cloneable.html [09.11.11]
93 http://code.google.com/p/objenesis/ [09.11.11]
94 http://prefuse.org/ [09.11.11]
95 Bezogen auf die Tatsache, dass die detektierten Relationen innerhalb des Systems lediglich als eine Menge von Quadrupeln abgebildet werden.
96 http://xstream.codehaus.org/ [09.11.11]

um so einen konsistenten Zugriff auf die Parameter ermöglichen zu können.

4.2.2 Frontend

Google Web Toolkit[97] (GWT) bezeichnet ein von Google entwickeltes Toolkit zur Entwickelung von vollständig JavaScript- bzw. AJAX-basierenden Webanwendungen. Eine Besonderheit stellt die Tatsache dar, dass die gesamte Entwicklung entsprechender Anwendungen konsistent mittels Java erfolgt, erst der enthaltene Java-to-JavaScript Compiler generiert aus den Quelldaten die eigentliche Webanwendung (analog beispielsweise einem „klassischen" Compiler, welcher exemplarisch C-Code in Maschinenbefehle transformiert). Dies bringt erhebliche Vorteile in der Entwicklung mit sich, zum einen da bewährte Entwicklungsumgebungen konsistent eingesetzt werden können und zum anderen, da ein Entwickler im Grunde genommen keine speziellen Kenntnisse im Bereich von Webanwendungen besitzen muss. RDAS verwendet GWT für die Realisierung des GUI-Prototypen bzw. des Frontends in Form einer Webanwendung. GWT stellt in diesem Sinne nicht nur einer einfache Bibliothek dar, sondern liefert die gesamte Entwicklungsinfrastruktur (unter anderem durch entsprechende Eclipse Plugins).

Smart GWT[98] ist ein auf GWT aufbauendes Framework, welches primär für die Bereitstellung von zahlreichen sogenannten *Widgets,* respektive GUI Elementen, verwendet wird. GWT selbst liefert nur einen rudimentären Satz an verfügbaren Elementen, welche nur einfache Anforderungen zufriedenstellend abdecken können. Mittels Smart GWT lassen sich sehr einfach Register, Baumstrukturen, dynamische Tabellen und vieles mehr realisieren, insgesamt besteht die Bibliothek aus über 300 derartiger Elementen für verschiedenste Anwendungsfälle. Neben diesen GUI Elementen liegt ein weiterer Schwerpunkt in bei der Bereitstellung von ausgereiften Datenintegrationsmechanismen, beispielsweise für die Backendanbindung.

gwt-links[99] ist eine weitere GWT Erweiterung, welche die Realisierung bzw. Visualisierung von graphenähnlichen Strukturen innerhalb von Webanwendungen ermöglicht. Die Bibliothek nutzt hierfür das HTML5 Canvas-Element, welches ein dynamisches Rendern von Bitmap-Grafiken ermöglicht, dies wird von RDAS genutzt, um Relationen zwischen Nachrichtenelementen optisch ansprechend und verständlich visualisieren zu können.

4.3 Projektstruktur

Die Strukturierung des Prototypen auf Implementierungsebene erfolgte in Anlehnung an die bereits vorgestellte Systemarchitektur (vgl. Abschnitt 3.1) bzw. durch Erweiterung dieser. In Folge dessen ergeben sich sechs Hauptkomponenten: *Import*[100], *Preprocessing*, *Relation*, *Output*, *Persistence* sowie *Configuration.*

4.4 Import

Die grundsätzliche Aufgabe der Imports besteht darin, einzelne Feeds zyklisch auf neue Nachrichten zu überprüfen, diese zu laden sowie in ein einheitliches Format zu überfüh-

97 http://code.google.com/webtoolkit [09.11.11]

98 http://code.google.com/p/smartgwt/ [09.11.11]

99 http://code.google.com/p/gwt-links/ [09.11.11]

100 Die Bezeichnung *import* in nicht zulässig innerhalb von Java Paketnamen, darum intern: *importing*

ren. Dabei werden alle Nachrichten in einer zentralen FIFO-Datenstruktur erfasst *(Queue)*, welche wiederum zyklisch durch das System abgefragt wird. Der Aufgabenbereich des Imports endet mit der Übergabe einer Nachricht an das *Preprocessing*. Das Einlesen der Feeds sowie die Extraktion der Nachrichten erfolgt mittels der ROME-Bibliothek, welche im Ergebnis Objekte der Klasse `SyndEntry` liefert, diese repräsentieren anschließend systemweit ein konkrete Nachricht welche eine einheitliche und abstrahierte Sicht auf Web-Feed-Elemente gewährleistet. Die Entscheidung, auf das durch ROME bereitgestellte Nachrichtenformat (bzw. Klasse) zu setzen wurde durch drei Gründe motiviert:

- Zukünftige Erweiterungen der ROME Bibliothek können ohne Änderungen in das System übernommen werden.
- Die Ausgabe des Activity Streams kann aufbauend auf die durch ROME angebotenen Mechanismen erfolgen.
- Die innerhalb von RDAS durchgeführten Prozesse greifen nur auf eine Untermenge der durch die Klasse `SyndEntry` bereitgestellten Attribute zu. Eine eigene Klasse müsste somit die gesamte Komplexität dieser abbilden (Mapping der Werte).

Durch die zusätzlichen Attribute (SPO-Struktur) des Activity Streams und durch verschiedenartige systeminterne Attribute (beispielsweise identifizierte Aliase, respektive Autoren) war es jedoch notwendig, die `SyndEntry`-Klasse zu erweitern, damit diese mehr Daten aufnehmen kann. Für diesen Zweck bietet ROME einen Erweiterungsmechanismus (vgl. Abschnitt 4.7.1) an, jedoch stellte sich dessen Umsetzung als problematisch heraus. Zum einen, da der Zugriff auf die neuen Attribute über diese sogenannten *Module* relativ umständlich[101] ist und zum anderen, da auf diese Weise ein Zugriff auf EPL-Ebene (vgl. Abschnitt 4.6.1) wesentlich aufwändiger wäre. Letztendlich sind ROME-Module konzeptionell auch kein Mechanismus, um zusätzliche Funktionalitäten zu implementieren - Module sollten nur tatsächlich vorhandene Elemente eines Namensraumes behandeln. Aus diesen Gründen erfolgte die Erweiterung der `SyndEntry`-Klasse mittels des *Decorator*[102] Entwurfsmusters.

Jeder Feed wird mittels der Klasse `Sensor` innerhalb des Systems durch einen eigenständigen Thread verwaltet, welcher auf diese Weise asynchron arbeiten kann. Innerhalb des Systems muss ausgeschlossen sein, dass eine Nachricht mehrfach verarbeitet wird, um dies zu erreichen wurde zwei Mechanismen implementiert: zum einen besitzt jeder `Sensor` einen eigenständigen (flüchtigen) Cache, welcher bereits verarbeitete Elemente erkennt, zum anderen überprüft das *Preprocessing* nochmals, ob eine Nachricht bereits existiert. Als Unterscheidungsmerkmal dient hierfür die URL einer Nachricht, dies stellt einen pragmatischen Ansatz dar und kann ggf. durch die Bildung von Hashwerten, bestehend dem tatsächlichen Inhalt einer Nachricht, erweitert werden.

101 Entsprechende Module müssen bei jedem Attributzugriff per URI geladen und instantiiert werden
102 http://www.dofactory.com/Patterns/PatternDecorator.aspx [09.11.11]

4.5 Preprocessing

Activity Stream Annotierung In den vorangegangen Kapiteln wurde herausgearbeitet, dass eine vollständig automatisierte Erkennung der einer Nachricht zugrunde liegenden Aktivität sehr schwierig bzw. unmöglich ist. Aus diesem Grund wurde innerhalb des Konzeptes eine *Annotator*-Komponente vorgesehen, welche (gesteuert durch eine Konfigurationsdatei) mittels Definitionen des Domänenexperten eingehende Nachrichten um aktivitätsbezogene Informationen anreichert. Auf Implementierungsebene existiert für den *Annotator* keine eigenständige Klasse, stattdessen ist dessen Funktionalität ein Bestandteil des *Event Pattern Analyzers* bzw. der Klasse `CEPListener` (vgl. Abschnitt 4.6.1). Eine wesentliche Zielstellung war, dass der Domänenexperte die Annotierungen möglichst komfortabel definieren kann, jedoch gleichzeitig eine ausreichend große Flexibilität gewährleistet ist. Entsprechend dieser Zielstellung bot sich eine Lösung mittels der bereits vorhandenen Esper EPL an. Der zugrunde liegende Gedanke war folgender: jede Nachricht stellt ein Event dar, der Domänenexperte definiert Filterkriterien für konkrete Nachrichtentypen und definiert gleichzeitig die für diese Nachrichten zu setzenden Annotierungen. Hierbei wurde ausgenutzt, dass EPL analog SQL innerhalb von SELECT -Statements die Angabe von Konstanten ermöglicht. Im Gegensatz zu den bereits diskutierten Ereignismustern betrifft eine derartige EPL-Anweisung also stets nur eine Nachricht und analysiert nur Merkmale dieser.

```
1   SELECT a AS annotateFeedItem, 'comment' AS objectObjectType,
2   a.text AS objectTitle, 'add' AS verb, 'thread' AS targetObjectType,
3   a.title AS targetTitle
4   FROM pattern
5   [ EVERY (a=FeedItem (
6   title LIKE  '%[Phpmyadmin-devel]%' AND title LIKE  '%Re:%'
7   ) ) ]
```

Listing 13: Activity Stream Annotierung

Das Beispiel in Listing 13 illustriert die Annotierung einer Nachricht, dies soll nachfolgend kurz erläutert werden: in Zeile 1 liefert die Anweisung im Feld `annotateFeedItem` das vollständig `FeedItem` Objekt zurück, RDAS benötigt dies, um anschließend die Daten dem Objekt anfügen zu können. Des Weiteren werden in Zeile 1 bis 3 die konkreten Werte der Activity Stream Felder angegeben, beispielsweise wird der Objekttyp *comment* sowie das Prädikat *add* festgelegt (als Konstanten). Deutlich wird auch (Zeile 2 und 3), dass für den Titel des *target*- sowie des *object*-Elementes die tatsächlichen Textfelder der Nachricht vergeben werden. In Zeile 6 werden die Bedingungen definiert, jede Nachricht, welche innerhalb ihres Titels die Zeichenkette „[Phpmyadmin-devel]“ sowie „Re:“ enthält, wird hierbei erfasst. Dies stellt eine sehr einfache Annotierungsanweisung dar, die Esper EPL ermöglicht jedoch weitaus komplexere Angaben, beispielsweise Bedienungen mittels regulärer Ausdrücke, Unterabfragen und vieles mehr. Ein mögliches Ergebnis der in Listing 13 dargestellten Annotierung lautet beispielsweise „Peter add comment 'I can't reproduce the bug' to thread 'Bug #123 – wrong picture'“[103]. Des Weiteren wird in Listing 13 verdeutlicht, dass mittels der Esper EPL der Zugriff auf die entsprechenden Attribute der `FeedItem`-Klasse direkt durch deren Angabe erfolgt. Esper bildet in diesem Sinne die Struktur der POJOs transparent ab. An dieser Stelle wird die

103 Der RDAS Prototyp unterstützt keine Konjugation von Verben.

Entscheidung verdeutlicht, die ROME `SyndEntry`-Klasse mittels eines Decorator-Entwurfsmusters durch die Klasse `FeedItem` zu erweitern: die Zugriffe auf Nachrichtenattribute erfolgen mittels impliziter Aufrufe der *Getter*-Methoden (Zeile 3) `a.text` bzw. `FeedItem.getText()`. Wäre für die Erweiterung der ROME `SyndEntry`-Klasse der Modulmechanismus (vgl. Abschnitt 4.7.1) eingesetzt worden, wären diese zusätzlichen Attribute nicht direkt über *Getter*-Methoden ansprechbar bzw. für Esper nicht sichtbar.

Die Implementierung der weiteren Vorverarbeitungsschritte gestaltete sich weitestgehend problemlos und soll daher nur kurz erläutert werden. Innerhalb der **Content-Extraction-Komponente** konnte die Feststellung gemacht werden, dass Trennzeichen oft nicht eindeutig für die Markierung von Signaturen ausreichen, da diese mehrfach vorkommen. Das System versucht dieses Problem zu lösen, indem für verschiedenartige Trennzeichenkombinationen die Anzahl der resultierenden Abschnitte erfasst wird, welche durch Aufsplitten des Textes entstehen. Diese werden gegeneinander verglichen und der größte Abschnitt wird als Signatur betrachtet. Dieses Vorgehen ist ein pragmatischer Ansatz, welcher versucht, den Nutzanteil einer Nachricht möglichst umfassend zu extrahieren. Dieser Ansatz zeigte ausreichend gute Ergebnisse, könnte jedoch noch zusätzlich abgesichert werden, indem die extrahieren Signaturen im Korpus gesucht werden und nur im positiven Fall (Treffer) eine Reduktion erfolgt.

Die **Keyword Extraction** wurde mittels des *Stanford POS-Taggers* innerhalb der `POS`-Klasse realisiert. Dieser liefert für die englische Sprache bereits ein trainiertes Modell (mit einer Genauigkeit von über 96% [85]), welches innerhalb des Kontextes der Arbeit ausreichend ist. Da das POS-Tagging vollständig im Arbeitsspeicher verarbeitet wird, zeigte sich, dass bei sehr langen Texten, welche teilweise reine Codeblöcke darstellten, Exceptions generiert werden. Dieses Problem konnte gelöst werden, indem zu lange Texte in Blöcke aufgeteilt wurden. Die gegenwärtige Lösung beachtet hierbei jedoch keine Sätze, in Folge dessen kann es passieren, dass an der Grenze zwischen zwei Blöcken fehlerhafte Annotierungen entstehen. Zukünftige Entwicklungen könnten daher das Aufsplitten nur an Satzgrenzen ermöglichen.

4.6 Relation

Der nachfolgende Abschnitt soll einige wesentliche Implementierungsmerkmale der Relationsdetektion näher erläutert, indem einzelne Analyzer charakterisiert werden.

4.6.1 Event Processing

Innerhalb einer Vorevaluation wurden verschiedenartige Event-Processing-Systeme analysiert, ein direkter Vergleich deren Leistungsmerkmale[104] ist jedoch schwierig [11, 41, 62]. Aus diesem Grund standen primär technische Gründe in Bezug auf eine möglichst einfache Integration in das zu konzipierende System im Vordergrund. In diesem Sinne bot das Event-Processing-Framework *Esper* die meisten Vorteile und überzeugte insbesondere durch eine sehr ausführliche Dokumentation [25]. Wie bereits in den Abschnitten 2.6.3 und 3.5.2 erwähnte wurde, kommt aus diesem Grund innerhalb von RDAS dieses System zum Einsatz, welches konzeptionell durch den *Event Pattern Analyzer* (vgl. Abbildung 8) repräsentiert wird. Aufgrund der intuitiven Architektur gestaltete sich die

104 Beispielsweise Bearbeitungszeit, Speicherverbrauch, Systemanforderungen etc.

Integration von Esper problemlos und konnte so im Wesentlichen innerhalb einer einzigen Klasse realisiert werden (`CEPListener`), welche eine Spezialisierung der Esper `UpdateListener`-Klasse darstellt. Diese besitzt im Wesentlichen eine zentrale Methode (`update`), welche bei Detektion eines Ereignismusters aufgerufen wird. Diese werden durch den Domänenexperten in einer XML Konfigurationsdatei definiert (`InteractionPatterns.xml`), welche durch das System verarbeitet wird. Eine wichtige Erkenntnis im Rahmen der Implementierung stellte die Verarbeitungsreihenfolge bzw. Priorisierung dar: standardmäßig ist diese nicht festgelegt, das heißt, Esper gibt keine Garantie, dass das erste Muster auch an erster Stelle verarbeitet wird. Das Konzept von RDAS sah jedoch vor, dass der Domänenexperte innerhalb der Konfigurationsdatei durch die Anordnung der Elemente implizit auch deren Verarbeitungsreihenfolge definiert, da dies einem intuitiven Vorgehen entspricht. Um dies in Esper zu ermöglichen, muss in einen Prioritäsmodus gewechselt werden:

```
1   Configuration esperConfig = new Configuration();
2   esperConfig.getEngineDefaults().getExecution().setPrioritized(true);
```

Listing 14: Esper Priorisierung

Anschließend kann Esper mittels einer EPL-Annotierung mitgeteilt werden, welche Abfragen zu priorisieren sind (`@Priority(Integer)`). Da die Konfigurationsdatei durch XStream absteigend eingelesen wurde, liegen die Ereignismuster bereits korrekt sortiert vor, es musste nun lediglich deren Summe ermittelt werden und jedes EPL Statement mit dem dekrementierten Wert dieser als Präfix erweitert werden. Diese Funktionsweise ist insbesondere innerhalb der Annotierungen notwendig, da der Domänenexperte auf diesem Wege erst globale Regeln definieren kann (beispielsweise allen Nachrichten einer Quelle das Prädikat *post* zuweisen) und erst später spezialisierte Anweisungen, welche ggf. vorherige überschreiben (beispielsweise indem das Prädikat *post* durch *add* überschrieben wird, wenn eine Notiz als Kommentar erkannt wurde). Auf diese Weise wird gewissermaßen eine Vererbungshierarchie ermöglicht, ähnlich der objektorientierten Programmierung.

4.6.2 Data-Mining

Die `ThreadAnalyzer`-Klasse ist für die Detektion von Thread-artigen Strukturen innerhalb der Nachrichtenströme verantwortlich. Dies wurde im Wesentlichen durch zwei Mechanismen realisiert: zum einen iteriert der **Thread Analyzer** über jegliche zuvor extrahieren *chunks* (vgl. Abschnitt 3.4.2) und zum anderen werden die Titelpräfixe analysiert. Wird innerhalb einer älteren Nachricht eine zu einem *chunk* deckungsgleiche Zeichenkette gefunden, so generiert das System zu dieser eine Relation. Theoretisch bedeutet dies, dass auch Relationen gesetzt werden, wenn die entsprechenden Zeichenketten rein zufällig erscheinen. Um diesen Faktor zu minimieren, bietet das System einen Konfigurationsparameter an, welcher eine untere Schranke der zu vergleichenden Wörter vorgibt, so dass zu kurze Vergleiche gemindert werden können. Beide Analysen erfolgen durch Suchoperationen innerhalb des Lucene-Index, standardmäßig bietet Lucene jedoch keine *Exact-Match*-Zeichenkettenvergleiche an, das heißt, wird nach einer Wortkombination *s* gesucht, werden auch Treffer zurück geliefert, welche *s* lediglich als Bestandteil enthalten (beispielsweise infix). Um dennoch exakte Übereinstimmungen zu suchen bie-

tet Lucene sogenannte *PhraseQuerys*, *MultiPhraseQuerys* und *SpanQuerys* an [42]. Deren Konstruktion ist jedoch aufwändiger, es zeigte sich des Weiteren, dass eine einfache Suche nach allen Übereinstimmungen (also auch fälschliche Infixvorkommen) mit einem nachträglichen Zeichenkettenvergleich ausreichend performant ist. Des Weiteren liefert eine Lucene-Suchanfrage stets eine nach Relevanz gewichtete Ergebnisliste zurück, dies ist aber innerhalb des *Thread Analyzers* nicht zielführend, da eine zeitlich geordnete Liste benötigt wird bzw. die neueste Nachricht.

```
1   Sort dateSorter = new Sort(new SortField[]{
2      new SortField(„published“, SortField.STRING, true),
3      new SortField(„idFeedItem“, SortField.STRING, true)
4   });
```

Listing 15: Lucene Date Sort

Aus diesem Grund wurde die entsprechende `Search`-Klasse um eine Methode `searchSortByDate` erweitert, welche mittels der Lucene `Sort`-Klasse die Ergebnissortierung steuert. In Listing 15 ist ein konzeptioneller Auszug diesbezüglich zu finden, im Wesentlichen wird ein `Sort`-Objekt instantiiert, welches nachfolgend Suchergebnisse nur noch nach Datum und anschließend nach ID sortiert ausgibt.

Die Klasse `PatternAnalyzer` repräsentiert den **Regular Expression Analyzer** und wertet die durch den Domänenexperten definierten regulären Ausdrücke aus und sucht innerhalb des Korpus nach diesen Mustern. Dieser Mechanismus wurde noch erweitert, indem bei jedem Vorkommen überprüft wird, ob alle vorherigen Vorkommen transitiv miteinander in Relation gesetzt sind. Lautet ein Muster beispielsweise „bug #123“ und ein Autor schreibt in einer Nachricht „ buf #123“ (Schreibfehler), so wird dies damit dennoch erfasst. Des Weiteren greift dieser Mechanismus auch, wenn der Wert einer ID beispielsweise innerhalb einer URL kodiert ist, welche ansonsten nicht durch den regulären Ausdruck erfasst werden würde. Die innerhalb der Konzeption erläuterte Notwendigkeit, nicht nur nach den Ausdrücken selbst, sondern letztendlich nach deren Rückgabewerten zu suchen, wurde nicht mittels des *Backtracking*-Mechanismus implementiert, sondern indem der Domänenexperte einen weiteren regulären Ausdruck angeben muss. Diese Variante vermeidet die teilweise recht komplexe Angabe von sogenannten Subsections und erschien komfortabler, da der zusätzliche reguläre Ausdruck zwangsläufig ein Teil des ursprünglichen sein muss und demnach einfach kopiert werden kann.

Bei der Realisierung des **Content Analyzers** wurde eine grundlegende Fragestellung nochmals deutlich: trotz des zu definierenden Schwellenwertes kommt es sehr oft vor, dass zu einer Nachricht A zahlreiche positive Nachrichten ermittelt werden können, welche inhaltlich ähnlich zu A sind. Hierbei stellt sich die Frage, ob der Nutzer jegliche Nachrichten angezeigt bekommt, oder ob es ausreichend ist, wenn diese transitiv erreichbar sind. Jegliche positive Nachrichten einfach in Relation zu setzen führt in vielen Fällen zu einer sehr großen Relationsanzahl, was wiederum einer zentralen Zielstellung von RDAS widerspricht, das Problem des *Information Overload* zu vermindern – es würde das genaue Gegenteil eintreten. Ist Nachricht A beispielsweise sehr ähnlich zu (absteigend nach Datum angegeben) B, C und D, so könnte es ausreichend sein, eine Relation zu B zu generieren. B wiederum generierte bereits eine Relation zu C und so weiter. Es konnte jedoch gezeigt werden, dass dies nicht in jedem Fall zutrifft: enthält A die Wör-

ter *Ice, Tea, Cub*e und *Cross*, *B* die Wörter *Ice* und *Tea* sowie *C* die Wörter *Cube* und *Cross*, so genügt es nicht, eine Relation nur zu B zu setzen, da *B* keine Relation zu *C* besitzt. Aus diesem Grund war es notwendig, dass die `ContentAnalyzer`-Klasse bei allen Treffern überprüft, inwiefern diese untereinander transitiv erreichbar sind. Dies wurde mittels *JGraphT* realisiert, indem für jeden Relationstyp ein eigenständiger (Erreichbarkeits-) Graph konstruiert wird, welcher anschließend beispielsweise mittels des *Dijkstra-Algorithmus* analysiert werden kann. Im skizzierten Beispiel bedeutet dies, dass erkannt werden kann, dass Nachricht *C* nicht von *B* aus erreichbar ist und in Folge dessen die Relation $A \rightarrow B$ generiert werden sollte. Dieser Algorithmus wird in Abbildung 19 nochmals grafisch dargestellt.

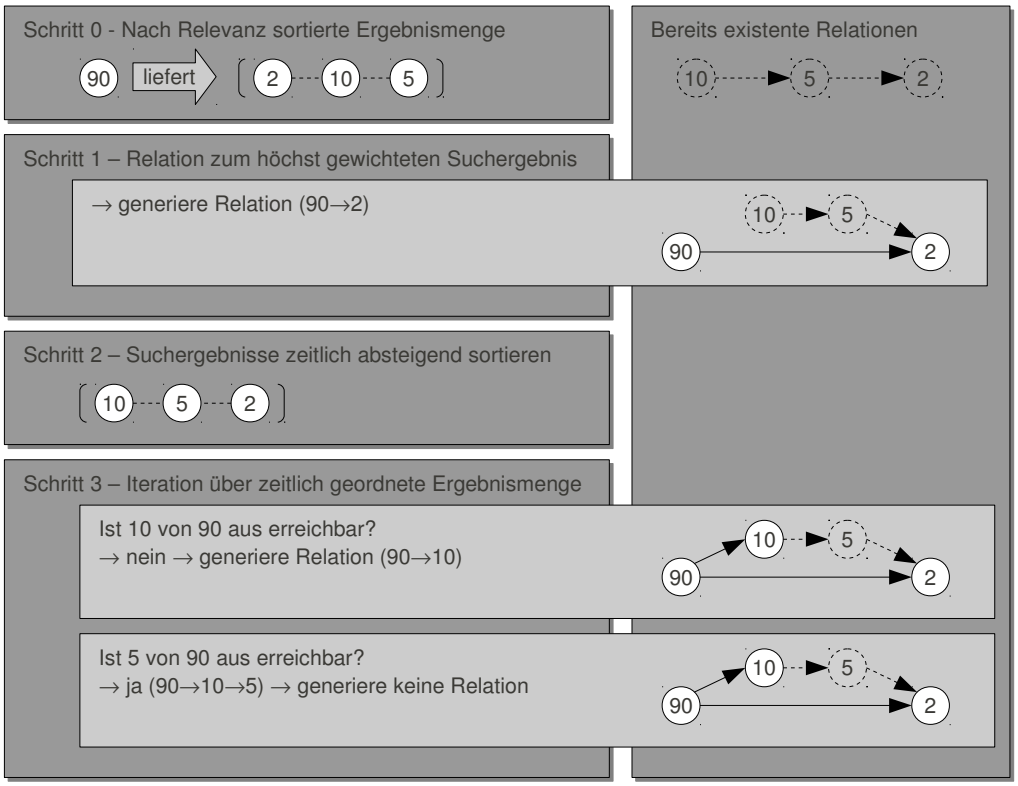


Abbildung 19: Erreichbarkeitsgraph der Relationen

Analog wird dieses Prinzip auch bei anderen Relationen angewendet, jedoch in einer leicht abgeänderten Funktionsweise: detektiert der *Pattern Analyzer* beispielsweise ein Muster, so generiert nur Relationen zu Nachrichten, welche nicht innerhalb eines Threads bereits in Beziehung stehen.

RDAS verwaltet mittels *JGraphT* für jeden Relationstyp jeweils eigenständige Graphen, dies ermöglicht eine wesentlich performantere Erreichbarkeitsanalyse, da die Graphen weniger komplex werden und Kanten nicht gefiltert werden müssen. Dennoch wurde auch ein Mechanismus geschaffen, welcher alle Relationen auf eine Sicht abbildet, gegenwärtig wird diese im System jedoch nicht benötigt.

4.7 Output

Das Primärziel von RDAS ist die Bereitstellung eines Nachrichten-Feeds im Activity--Stream-Format, neben diesem sollte des Weiteren durch eine prototypische Benutzeroberfläche ein Frontend in Form einer Webanwendung realisiert werden.

4.7.1 Activity Stream

Standardmäßig wird das Activity-Stream-Format von ROME in der Version 1.0 nicht unterstützt, das heißt, der Activity-Stream-Namensraum bzw. dessen neue Elemente wie beispielsweise Verben werden bei der Generierung eines Atom Web Feeds nicht berücksichtigt. Für die Generierung des Activity Streams kamen prinzipiell zwei Möglichkeiten in Betracht, zum einen die manuelle Erstellung der entsprechenden XML-Ausgabe und zum anderen eine Erweiterung von ROME. Aufgrund der Tatsache, dass ROME einen Erweiterungsmechanismus mittels sogenannter *Module* [83] bereitstellt, wurde die erst genannte Möglichkeit nicht weiter verfolgt. Ein ROME-Modul besteht hierbei aus vier Komponenten: Schnittstelle, Implementierung, Parser sowie Generator. Innerhalb der Schnittstelle wird die URI (Namensraum) sowie die *Getter* und *Setter* der neu hinzukommenden Attribute definiert – über die URI erfolgt anschließend die Identifizierung des Moduls. Die Modulimplementierung implementiert anschließend diese Schnittstelle sowie eine `copyFrom`-Methode, welche für die Replikation der Nachrichten notwendig ist. Wie der Name bereits impliziert, enthält ein ROME-Modul des Weiteren einen Parser, in welchem detailliert definiert werden muss, wie ein geladenes XML-Dokument (welches die innerhalb des Moduls neu hinzugefügten Elemente enthält) zu verarbeiten ist. Das heißt, dem System wird im Wesentlichen mittels XML-DOM-Operationen mitgeteilt, wie bzw. an welcher Stelle im Dokumentenbaum die Elemente auftreten und wie deren Werte zu extrahieren sind. Diese Funktionalität wurde innerhalb des Prototypen jedoch nur exemplarisch implementiert, da ein vollständiger Activity Stream als Eingabeformat zum gegenwärtigen Zeitpunkt noch unwahrscheinlich erscheint bzw. nicht berücksichtigt wurde, siehe auch Abschnitt 6.3. Innerhalb von RDAS stellt somit der Generator die wichtigste Klasse dar, da diese die eigentliche Generierung des Activity Streams realisiert. Hierbei wird im Wesentlichen analog dem Parser (jedoch invers) die Erzeugung des XML-Dokumentes gesteuert, indem definiert wird, an welchen Positionen innerhalb des Dokumentenbaums die neuen Elemente einzufügen sind.

4.7.2 Web Application

Die Realisierung des GUI-Prototypen erfolge mittels des Google Web Toolkits und darauf aufbauenden Bibliotheken. Die grafische Ausgabe ist hierbei dreigeteilt: im oberen Bereich werden die Nachrichtenrelationen mittels der Thread-Arc-Metapher dargestellt (vgl. Abschnitt 3.7), im linken unteren Bereich befindet sich die Ausgabe des eigentlichen Activity Streams und im rechten unteren Bereich werden Detail- und Metainformationen einzelner selektierter Nachrichten ausgegeben.

Für die Realisierung der *Thread Arcs* wurde die die *GWT* Bibliothek *gwt-links* eingesetzt, deren API für die innerhalb von RDAS auftretenden Anforderungen jedoch zu eingeschränkt war. Beispielsweise bietet die Klasse `StraightArrowConnection` (welche die Kanten bzw. Bögen repräsentieren) keinen Zugriff auf deren Farbgebung oder Linienbreite. Aus diesen Gründen musste der Quellcode entsprechend analysiert und erweitert

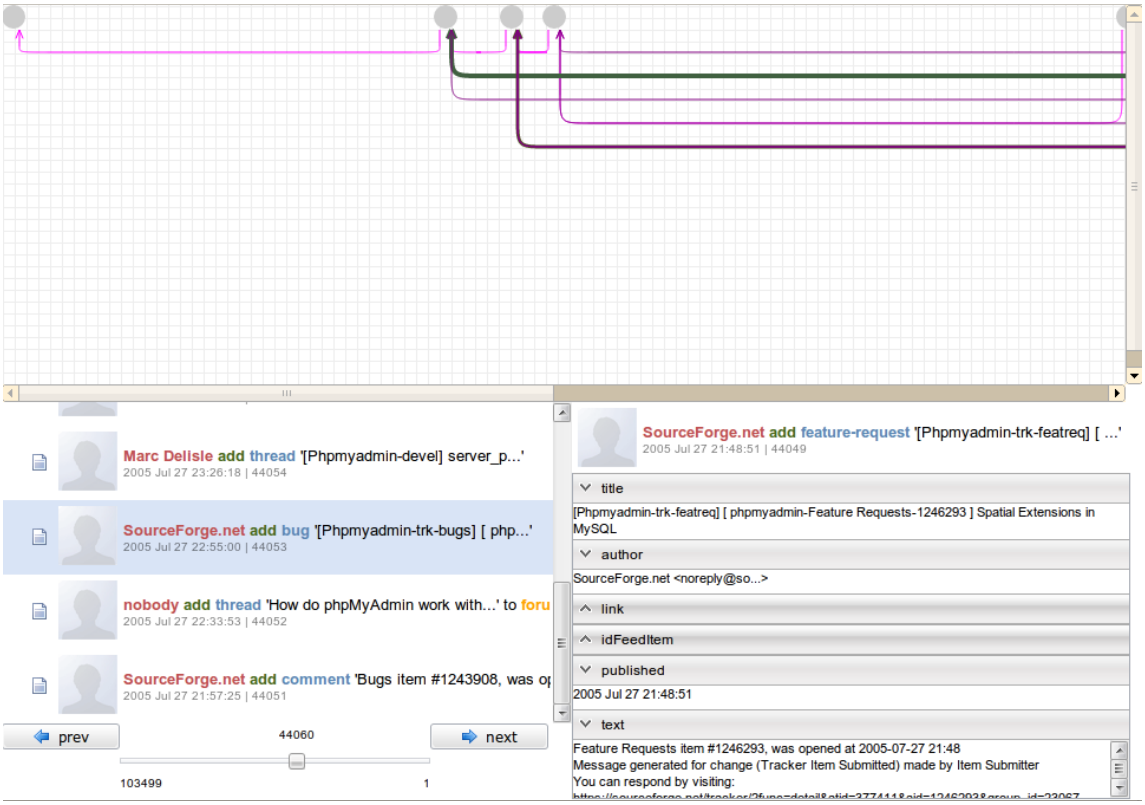


Abbildung 20: GUI-Prototyp

werden. Neben *gwt-links* wurden weitere Bibliotheken analysiert, welche prinzipiell für eine Darstellung von graphenähnlichen Strukturen geeignet erschienen, einer dieser Vertreter war *gwt-connectors*[105] welches jedoch nur mit einer veralteten GWT-Version lauffähig ist und augenscheinlich als obsolet zu betrachten ist. Eine weitere Bibliothek stellt *gwt-diagrams*[106] dar, welche jedoch noch gravierendere Probleme aufweist und als nicht lauffähig einzustufen ist. Die Realisierung der restlichen Komponenten (Ausgabe des Activity Streams, Detailansicht und Steuerungselemente, beispielsweise ein dynamischer *Slider* für eine zeitliche Navigation innerhalb der Nachrichten) gestaltete sich mittels *GWT* und *Smart GWT* problemlos und entspricht in der Vorgehensweise bzw. Konzeption bekannten Java-GUI-Toolkits wie beispielsweise *Swing,* das heißt, bis zu einem gewissen Grad sind innerhalb von GWT keinerlei Kenntnisse im Bereich *Web Development* (HTML, CSS, JavaScript etc.) notwendig. Die Client-Server-Kommunikation gestaltet sich innerhalb der GWT Entwicklung sehr einfach mittels einer Schnittstelle für *Remote Procedure Calls*, bei welcher aufgrund des ganzheitlichen Einsatzes von Java auf Implementierungsebene keinerlei Datenanpassungen notwendig sind – es werden stets vollständige Objekte serialisiert und per JSON übertragen, dies geschieht jedoch für den Entwickler vollkommen transparent. GWT kann jedoch nur Objekte in JavaScript-Code transformieren, welche keine serverseitigen Bibliotheken bzw. Objekte referenzieren. Aus diesem Grund war es nicht möglich, die Nachrichtenobjekte der Klasse `FeedItem` direkt innerhalb des Clients zu nutzen, da diese ROME `SyndEntry`-Objekte referenzieren. Dies wurde durch Containerobjekte gelöst (`FeedItemContainer`), welche nur die für den Client notwendigen Daten bzw. Datenprimitive wie Integer und String enthalten.

105 http://code.google.com/p/gwt-connectors/ [22.11.11]
106 http://code.google.com/p/gwt-diagrams/ [22.11.11]

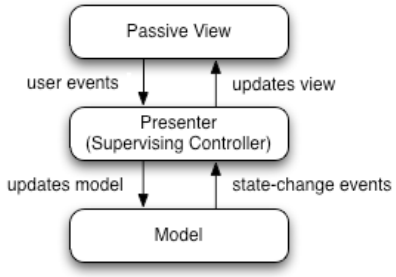


Abbildung 21: Model View Presenter

Innerhalb der Entwicklung des Prototypen wurde das sogenannte *Model-View-Presenter*-Entwurfsmuster[107] eingesetzt, welches unter anderem eine striktere Trennung zwischen Ausgabe und Logik ermöglicht. Aus diesem Grund ergeben sich innerhalb des Client-Paketes stets eine *view-* sowie *presenter*-Klasse. Die Auslieferung der erstellten Webanwendung erfolgte anschließend mittels eines *jetty-Webservers*[108], welcher unter anderem den Vorteil bietet, dass keine Installation notwendig ist, da dieser selbst rein Java-basierend ist.

4.8 Persistence

Innerhalb von RDAS wurde ein Konglomerat an verschiedensten Persistierungsmechanismen eingesetzt, hierzu zählen unter anderem eine relationales DBMS (MySQL), mehrere Indizes (Lucene) sowie verschiedenartige Disk-basierte Datenstrukturen. Diese Aufteilung ist im Wesentlichen dem Ziel einer ausreichenden Verarbeitungsgeschwindigkeit geschuldet. Das System stellt hierfür eine zentrale `Database`-Klasse zur Verfügung, welche das Entwurfsmuster *Facade* sowie *Singleton*[109] realisiert, das heißt, alle Zugriffe werden über diese verarbeitet, die zugreifenden Instanzen haben jedoch keine Kenntnis über die zugrunde liegenden Persistierungsmechanismen (Schichtenmodell) – auf diese Weise wird ein Austausch einzelner Komponenten problemlos ermöglicht. Erste Ansätze wurden unter ausschließlichem Einsatz von MySQL umgesetzt, hierbei zeigte sich jedoch schnell, dass die dadurch resultierende Antwortzeit viel zu groß war. Unabhängig hiervon bietet MySQL auch keine fortgeschrittenen Information-Retrieval-Mechanismen an, dass heißt, es findet kein Ranking von Suchergebnissen nach beispielsweise *tf-idf* bzw. *Kosinus-Maß* statt. Aus diesem Grund kommt innerhalb von RDAS für jegliche Suchanfragen Lucene zum Einsatz, welches diese um Größenordnungen schneller bearbeiten kann. Ein Benchmark bzw. Vergleich der Dauer einer Suchanfrage zwischen MySQL und Lucene findet sich in [89]. Die beiden Systeme ergänzen sich somit, da der Lucene-Index im Gegensatz zu einer relationalen Datenbank für eine sichere Speicherung der Daten ungeeignet ist.

Lucene Innerhalb der Data-Mining-Komponente muss das System zahlreiche Suchprozesse und Vergleiche gegen den gesamten Datenbestand durchführen, dies betrifft nicht nur den *Content Analyzer*, sondern wird von fast allen Komponenten unterschiedlich intensiv genutzt. Um Suchanfragen in ausreichender Geschwindigkeit verarbeiten zu können, wird jede eintreffende Nachricht (bzw. alle Daten dieser) in einem zentralen

107 http://msdn.microsoft.com/en-us/magazine/cc188690.aspx [22.11.11]
108 http://www.eclipse.org/jetty/ [10.08.11]
109 http://www.oodesign.com/singleton-pattern.html [20.11.11]

Index gespeichert. Hierbei stehen die Daten dieser (neuen) Nachricht jedoch nicht sofort zur Verfügung, das heißt, diese erscheinen innerhalb nachfolgender Suchanfragen erst, wenn die Änderungen am Index auch gespeichert bzw. geschrieben wurden *(commit)*. Erste Ansätze basierten darauf, diesen Commit somit nach jeder eintreffenden Nachricht durchzuführen, dies verursacht jedoch bei jedem Aufruf schreibende Zugriffe auf den Disk-basierten Index und in Folge dessen zu inakzeptabel großen Verarbeitungszeiten. Um das Problem eines dynamischen Index zu lösen, bietet Lucene eine sogenannten *Near-Realtime-Search*-Mechanismus [65] an, welcher zum Ziel hat, Änderungen am Index (dies schließt das Hinzufügen von neuen Dokumenten ein) nahezu in Echtzeit [15] nachfolgenden Suchanfragen zur Verfügung zu stellen.

```
1   IndexWriter writer; // create IndexWriter
2   Document doc = null; // create document
3   writer.addDocument(doc); // update document
4   IndexReader reader = writer.getReader(); // get reader with the new document
5   Document addedDoc = reader.document(0);
```

Listing 16: Lucene Near Realtime Search Code-Auszug

Dieser Mechanismus wird unter anderem auch innerhalb von *Twitter* (in einer erweiterten Form) genutzt und löste innerhalb von RDAS das Problem des dynamischen Indexes.

Der Einsatz des relationalen DBMS **MySQL** dient primär der sicheren Persistierung der Daten, da weder die Disk-basierte Speicherung der Lucene-Indizes noch die JDBM-Datenstrukturen als ausreichend sicher bzw. formatunabhängig angesehen werden können. Wie bereits beschrieben wurde, erfolgt eine Speicherung in zyklischen Abständen. Hierbei zeigte sich, dass die Angabe bzw. Gruppierung einzelner Statements zu Performance-Engpässen führt, aus diesem Grund erfolgt die Speicherung mittels des MySQL-Batch-Modus [64], welcher eine Gruppierung von Anweisungen ermöglicht.

JDBM Obwohl mit Lucene und MySQL bereits zwei Persistierungsmechanismen zur Verfügung standen, zeigte sich, dass diese nicht vollständig ausreichend sind. Beide Systeme speichern lediglich die konkreten Attributwerte der Objekte (Nachrichten), jedoch nicht diese selbst. Aufgrund der Größe der zu verarbeitenden Datenmenge zeigte sich frühzeitig, dass eine Speicherung der Objekte innerhalb der Arbeitsspeichers als äußert kritisch einzustufen ist und des Weiteren nicht skaliert. Die Persistenzengine JDBM schließt diese Lücke, indem Objekte innerhalb von B-Bäumen und Hash-Tabellen verwaltet werden könnten. Des Weiteren werden Transaktionsmechanismen bereitgestellt, welche aber im Kontext von RDAS nur eine untergeordnete Rolle spielen. Innerhalb von RDAS wird JDBM konkret innerhalb folgender Bereiche eingesetzt:

- Speicherung jeglicher extrahierter Hyperlinks aller Nachrichten (B-Baum)
- Speicherung der URLs aller Nachrichten (B-Baum)
- Speicherung der `FeedItem` Nachrichtenobjekte (Hash-Tabelle)
- Speicherung der Graphen (Hash-Tabelle)

Eine typische Aufgabe innerhalb von RDAS stellt beispielsweise der direkte Zugriff auf ein Nachrichtenobjekt dar, dessen ID als Ergebnis von Suchoperationen (Lucene) zurück geliefert wurde. Durch die Disk-basierte Speicherung innerhalb einer Hash-Tabelle entfällt ein Datenbankzugriff bzw. steht das Objekt transparent zur Verfügung. Die B-Bäume ermöglichen sehr schnelle Suchanfragen, welche prinzipiell auch mittels MySQL oder Lucene durchgeführt werden könnten, jedoch zeitaufwändiger wären. Ein wesentlicher Vorteil von JDBM liegt des Weiteren in der Tatsache begründet, dass die zu speichernden Objekte vollständig transparent serialisiert werden, das heißt, es müssen keine zusätzlichen Methoden oder Schnittstellen implementiert werden, beispielsweise `Serialize`. JDBM stellt innerhalb von RDAS somit eine essentielle Komponente dar, welche innerhalb der prototypischen Realisierung intensiv genutzt wird, um jegliche Daten verfügbar zu halten.

RDAS führt automatisiert einen *Commit* der im Arbeitsspeicher gehaltenen Daten durch, dies betrifft jegliche Persistierungsmechanismen, welche ausnahmslos erst nach einem *Commit* geschrieben werden. Das System kennt hierfür zwei Schwellenwerte: eine obere Zeitschranke und eine maximale Anzahl an Nachrichten. Trifft also nach dieser Zeitspanne keine neue Nachricht ein, wird automatisch ein *Commit* durchgeführt, analog geschieht dies, wenn die maximale Anzahl (beispielsweise 1000 Nachrichten) erreicht wurde.

4.9 Configuration

Während der Konzeption und der Implementierung wurde berücksichtigt, dass RDAS zahlreiche Konfigurationsparameter besitzt, welche teilweise stark domänenabhängig sind. Aus diesem Grund wird RDAS durch drei separate XML-Dateien parametrisiert, diese enthalten die zu detektierenden Ereignismuster, Annotierungsvorschriften sowie globale Systemparameter. Hierbei wurde ein Objekt-Mapping mittels *XStream* verwendet, dies ermöglicht eine typsichere Definition von Attributen und erfordert kein händisches Parsen der XML-Dokumente. Erste Ansätze verwendeten den *Java-Properties-Mechanismus*[110], welcher jedoch mit zunehmender Komplexität ungeeignet erschien.

110 http://openbook.galileodesign.de/javainsel5/javainsel11_006.htm [20.10.11]

5 Validierung

Ziel des folgenden Kapitels soll die Validierung der im Rahmen der Arbeit entstandenen Software RDAS sein. Diese gliedert sich in zwei grundlegende Bereiche: einer Nutzerstudie bezüglich ausgewählter Aspekte der Relationsdetektion sowie einer anschließenden kurzen Analyse der Leistungsfähigkeit bzw. der Skalierbarkeit. Problematisch für eine umfassende Analyse der Konzepte sowie des entstandenen Prototypen ist die Tatsache, dass keine markierten Datensätze im Sinne eines sogenannten *Gold Standards* [61] existieren, respektive gefunden werden konnten, das heißt, es standen keine Vergleichsdaten zur Verfügung, welche bereits nach verschiedenen Gesichtspunkten detektierte Relationen enthalten.

5.1 Nutzerstudie

Mittels einer Evaluationsanwendung sollte im Rahmen einer Nutzerstudie untersucht werden, inwiefern die innerhalb des phpMyAdmin-Datensatzes detektierten Relationen durch Dritte bestätigt oder widerlegt werden können bzw. inwiefern Nutzer unter Umständen Beziehungen zwischen Nachrichten erkennen, welche innerhalb der Konzeption nicht berücksichtigt wurden. Um einen umfassenden Gold Standard generieren zu können, müsste theoretisch jede Nachricht im Datensatz zu jeder älteren in Relation gesetzt werden und anschließend eine Bewertung erfolgen. Dies ist jedoch praktisch kaum realisierbar, da beispielsweise innerhalb des phpMyAdmin-Datensatzes auf diese Weise theoretisch über 30 Milliarden Vergleiche[111] bzw. händische Bewertungen notwendig wären. Die Zahl der tatsächlichen Relationen innerhalb dieser dürfte jedoch im Vergleich extrem niedrig sein. Des Weiteren sind auf diese Weise komplexe Ereignismuster nicht erkennbar. Aus diesem Grund wurde für die Nutzerstudie ein anderer Ansatz gewählt: zum einen sollten die durch RDAS detektieren Relationen beurteilt werden und zum anderen durch Hinzufügen von zufälligen Relationen erkannt werden, inwiefern möglicherweise weitere Beziehungsmerkmale zu berücksichtigen sind. In beiden Fällen sollten die Nutzer jedoch frei entscheiden bzw. wurde ihnen nicht mitgeteilt, ob es sich zum eine zufällige oder eine durch RDAS detektierte Relation handelt. Insbesondere sollte verhindert werden, dass sich die Nutzer zu stark auf die innerhalb der Konzeption entworfenen Relationsmechanismen konzentrieren. Für diesen Zweck wurde eine webbasierte Evaluationsanwendung erstellt (Abbildung 22), welche den Nutzern erlaubte, Bewertungen für konkrete Relationen abgeben zu können. Aufgrund der bereits innerhalb der Implementierung vorhandenen Infrastruktur bzw. den erlangten Erkenntnissen, erfolgte die Erstellung ebenfalls mittels GWT. Dies ermöglichte eine effiziente Realisierung bzw. Wiederverwendung der vorhandenen Codebasis. Eine vorherige (grobe) Evaluation existierender Umfrageanwendungen[112] zeigte schnell, dass der damit einhergehende Anpassungsaufwand an die vorhanden Datenstrukturen den einer eigenständigen Implementierung übersteigen würde. Die Evaluationsanwendung sollte möglichst ohne Vorkenntnisse bzw. Einführungen bedient werden können, aus diesem Grund wurde in die Anwendung selbst ein sogenannter *Splash Screen* eingefügt, welcher beim initialen

111 30 Milliarden, da jede Nachricht n zu jeder älteren Nachricht m mit $datum(m) < datum(n)$ in Relation stehen könnte. $MaxRelations = \frac{|AllMessages|^2 - |AllMessages|}{2}$

112 Beispielsweise http://www.limesurvey.org/ [14.11.11]

Ladevorgang angezeigt wird und dem Nutzer die wesentlichen Vorgänge erläutert. Dies und der Verzicht auf eine Nutzeranmeldung sollte die Einstiegshürde möglichst niedrig halten. Eine Herausforderung stellten hierbei die Formulierungen dar: einerseits sollten die Nutzer ergebnisoffen Entscheidungen treffen, andererseits musste jedoch die grundlegende Zielstellung erläutert werden. Aus diesem Grund wurde verstärkt darauf geachtet, dass die Frage- bzw. Hilfestellungen nicht zu suggestiv formuliert wurden. Die Funktionsweise der Evaluationsanwendung wurde bewusst sehr einfach gehalten: dem Nutzer werden stets zwei Nachrichten bzw. eine Relation präsentiert, welche mittels eines Zufallsmechanismus aus dem Datensatz ausgewählt wurde. Hierbei sieht der Nutzer jegliche Metadaten sowie den Inhalt der beiden Nachrichten, mittels entsprechender Radio-Buttons kann anschließend eine Bewertung bezüglich der Korrektheit der Relation abgegeben werden.

Hierfür kam die sogenannte **Likert-Skala**[113] [81] zum Einsatz. Diese stellt ein Skalierungsverfahren dar, welches dazu dient, die individuelle Einstellung (Meinung) von Versuchspersonen bezogen auf ein konkretes Objekt zu ermittelt. Sie wird sehr oft im Zusammenhang mit der Erstellung von Fragebögen verwendet. Ein Objekt entspricht hierbei meist einer konkreten Aussage, zu welcher die Versuchsperson eine Entscheidung der Form “trifft zu”, “trifft eher zu”, “weder noch”, “trifft eher nicht zu” und “trifft nicht zu” treffen muss (abweichende Bezeichner und Umfang dieser sind ebenfalls gebräuchlich).

Abschließend sollen noch einmal die Primärziele der Nutzerstudie aufgelistet werden:

- Beurteilung der Qualität der detektierten Relationen
- Erkennen von nicht berücksichtigten Relationen
- Ableiten von Konfigurationsparametern bzw. Gewichtungsfaktoren

113 Nach dem amerikanischen Psychologen Rensis Likert benannt.

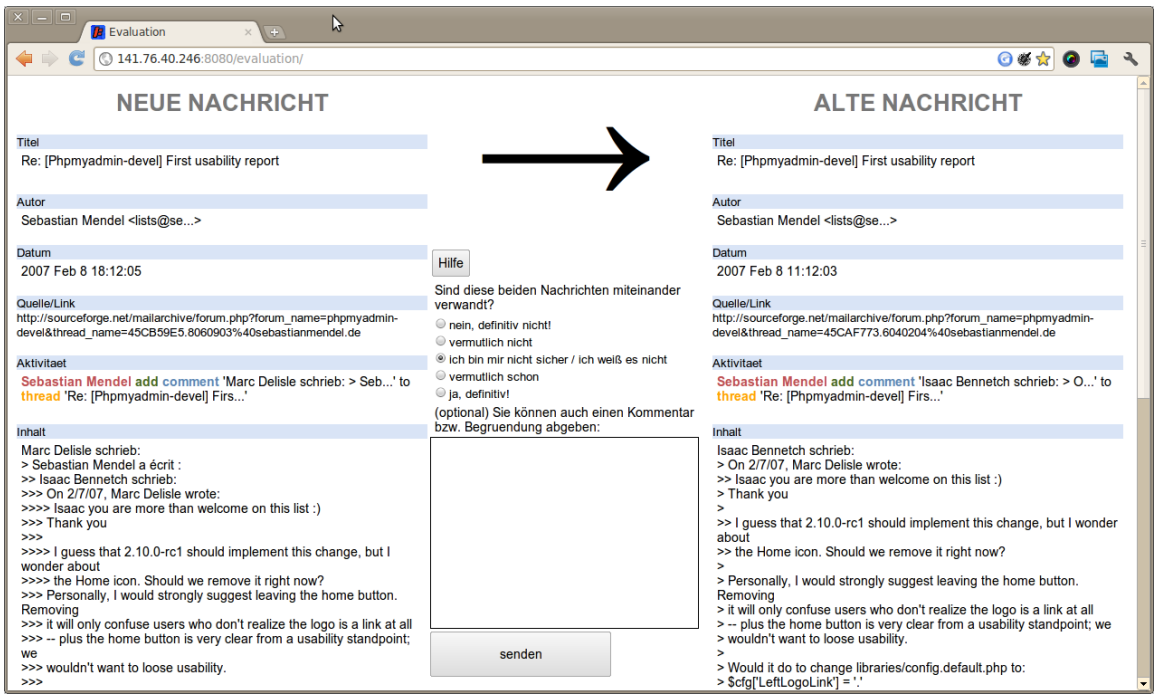


Abbildung 22: Evaluationsanwendung

5.1.1 Datenbasis

Aus einer Untermenge von rund 50.000 Nachrichten des phpMyAdmin-Datensatzes wurden circa 2.000 Relationen zu möglichst gleichen Teilen extrahiert, zu diesen wurden anschließend rund 800 zufällige Relationen hinzufügt. Um die Wahrscheinlichkeit einer positiven Beziehung zwischen zwei zufälligen Relationen zu erhöhen, wurden ein *Schiebefenstermechanismus* mit einer Größe von 100 Nachrichten angewendet, das heißt, zu einer Zufallsnachricht x wurde eine Zufallsnachricht y mit $getId(y) > getId(x)\text{-}100$ ausgewählt.

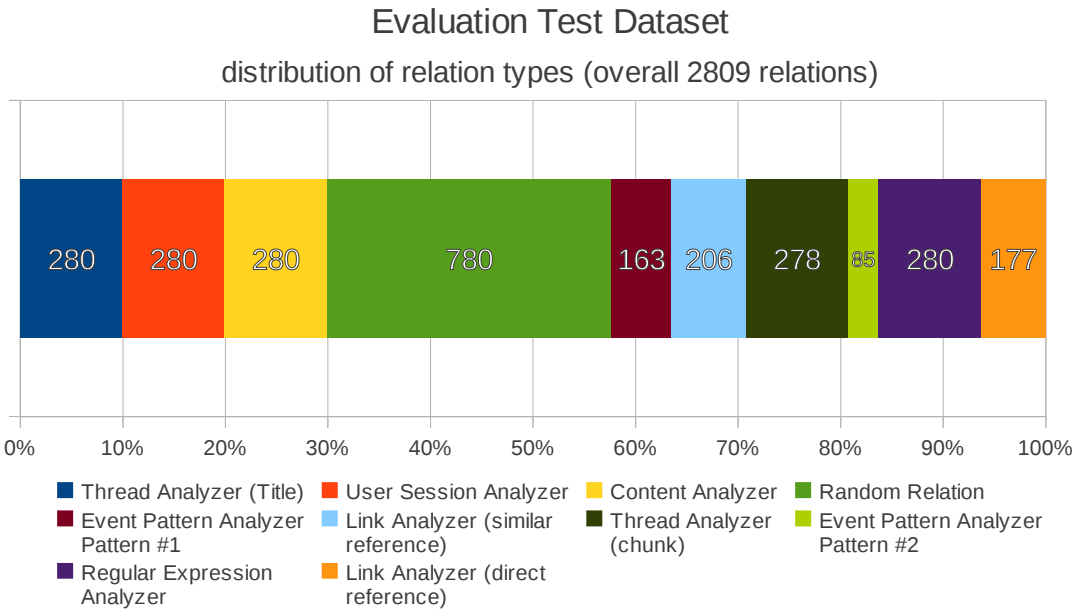


Abbildung 23: Verteilung Evaluationsdatensatz

Die zufälligen Relationen sind im Vergleich zu den anderen Arten bewusst überrepräsentiert, um auf diese Weise möglichst viele derartiger Bewertungen erhalten zu können. Die Anzahl von rund 800 Nachrichten ist hierbei frei gewählt bzw. stellt dies grob ein Drittel der Relationen dar. Dieser Anteil wurde bewusst nicht zu hoch gewählt, da davon ausgegangen wurde, dass positive Relationen in zufälligen Kombinationen eher selten auftreten und die evaluierenden Nutzer durch immer wiederkehrende Negativ-Relationen demotiviert werden und diese in Folge dessen weniger Bewertungen abgeben.

Bezogen auf die vorgestellte Konfusionsmatrix (vgl. Abschnitt 2.3.2) bedeutet dies, dass mittels der zufälligen Relationen potentielle falsch/richtig-negativ Ergebnisse erkannt werden können, sofern ein Nutzer eine dieser als korrekt/positiv markiert. Dieser Sachverhalt ist innerhalb nachfolgender Tabelle bzw. Konfusionsmatrix nochmals strukturiert dargestellt.

	Nutzer entscheidet: Nachricht A steht in Relation zu Nachricht B	**Nutzer entscheidet: Nachricht A steht *nicht* in Relation zu Nachricht B**
Relation wurde durch RDAS erkannt/gesetzt	richtig positiv *Eine detektierte Relation wird als korrekt markiert*	falsch positiv *Eine detektierte Relation wird als falsch markiert*
Relation wurde durch RDAS nicht erkannt/gesetzt	falsch negativ *Eine zufällige Relation wird als korrekt markiert*	richtig negativ *Eine zufällige Relation wird als falsch markiert*

Tabelle 3: Konfusionsmatrix im Kontext der Evaluation

Die Ergebnisse der zufälligen Relationen sind somit ein wichtiger Indikator für die Bewertung der Genauigkeit *(precision)* bzw. der Trefferquote *(recall)* (vgl. Abschnitt 2.3.2). Die jeweilige Menge der relevanten Dokumente ist aufgrund des fehlenden Gold Standards unbekannt, die Bewertung der zufälligen Relationen liefert jedoch zumindest ein Indiz für deren Umfang: werden die meisten zufälligen Relationen als negativ klassifiziert, so kann zumindest geschlussfolgert werden, dass die Menge der relevanten Dokumente nicht wesentlich größer als die durch RDAS ermittelte ist. Sollte dies nicht der Fall sein, das heißt, die Nutzer markieren sehr viele zufällige Relationen als korrekte/positive Relation, so lässt dies die Schlussfolgerung zu, dass die durch RDAS detektierten Mengen der relevanten Relationen zu klein sind, respektive die Relationsdetektion zu ungenau bzw. „zu gering“ ist.

Die Bezeichner der Legende bzw. der Relationstypen in Abbildung 23 entspricht hierbei weitestgehend den Formulierungen aus Abschnitt 3.5. Die Relationen „Event Pattern Analyzer #1/2“ bezeichnen hierbei zwei im Rahmen der Arbeit erstellte Ereignismuster. Da die Nutzer deren Definition nicht kennen, ist es jedoch sehr schwierig bis unmöglich diese auch zu erkennen – beispielsweise wenn das Muster aus mehr als zwei Nachrichten besteht (da die Evaluationsanwendung stets nur zwei Nachrichten präsentiert). In Folge dessen ist diese Komponente von RDAS mit dem gewählten Evaluierungsansatz nur

bedingt überprüfbar, des Weiteren sind für komplexe Muster auch tiefer gehende Kenntnisse der entsprechenden Domäne notwendig, welche jedoch innerhalb des Nutzerkreises nicht als gegeben vorausgesetzt werden konnte.

Des Weiteren wurden zielgerichtet alleinstehende Relationen verwendet, dies soll nachfolgend erläutert werden: steht Nachricht A aufgrund einer Thread-artigen Struktur zu Nachricht B in Relation, beispielsweise durch Vorkommen von Zitaten, so stellt dies eine relativ starke Beziehung dar. Gleichzeitig kann A und B jedoch auch hinsichtlich weiterer Relationstypen in Beziehung stehen, beispielsweise aufgrund der Tatsache, dass A einen Link auf B enthält *(direct link)*. Innerhalb der Nutzerstudie würden die meisten Nutzer vermutlich in Folge dessen nicht die Direct-Link-Relation erkennen und bewerten, sondern die Thread-Relation. Aus diesem Grund wurde der Datensatz gefiltert, so dass (exemplarisch) eine Direct-Link-Relation nicht durch eine andere Relation überdeckt wird. Dies erklärt auch, dass innerhalb des Datensatzes einige Typen eine geringere Anzahl besitzen – in diesen Fällen war es nicht möglich, ausreichend alleinstehende Relationen zu extrahieren bzw. existierten nicht genügend.

5.1.2 Ergebnisse

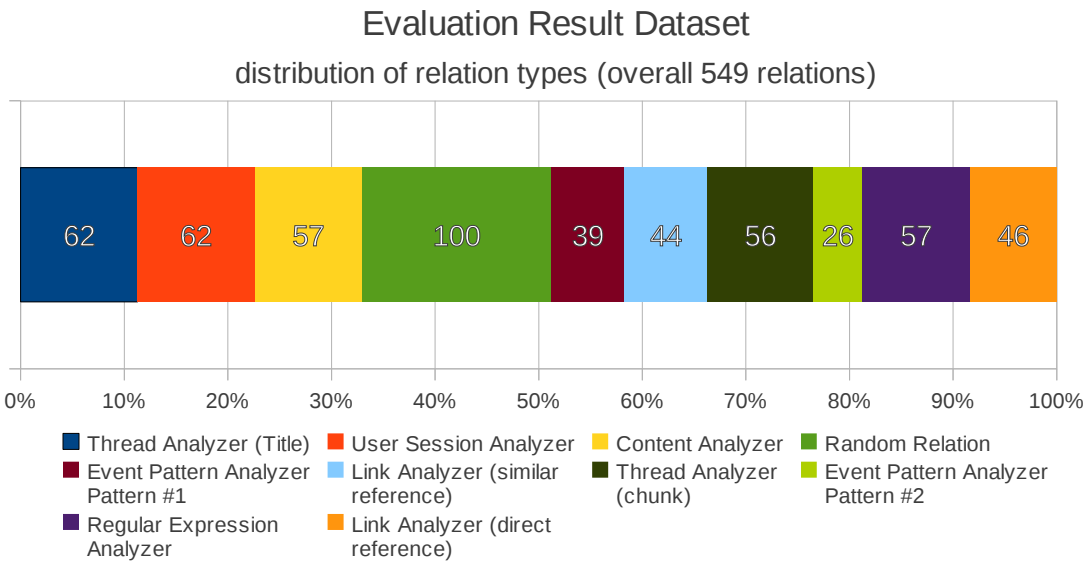


Abbildung 24: Verteilung Evaluationsergebnisdatensatz

Die Nutzerstudie wurde über den Zeitraum von einer Woche durchgeführt und führte insgesamt zu 549 bewerteten Relationen. Ihre Verteilung ist in Abbildung 24 dargestellt und entspricht erwartungsgemäß dem der Ausgangsdaten, das heißt, die zufällige Auswahl bevorzugte nicht bestimmte Relationen. Aufgrund der Tatsache, dass für eine Teilnahme an der Evaluation keine Anmeldung erforderlich war, kann nur ungefähr ermittelt werden, wie viele unterschiedliche Nutzer die Bewertungen abgegeben haben. 12 Nutzer gaben freiwillig eine E-Mail-Adresse an, so dass von mindestens 12 unterschiedlichen Nutzern ausgegangen werden kann. Durch Analyse der Zeiträume der Bewertungen kann weiterhin auf individuelle Nutzer geschlossen werden, im Datensatz lassen sich circa 30 zusammenhängen Sitzungen (das heißt zeitlich eng beieinander liegende Bewertungen) extrahieren, so dass die tatsächliche Anzahl der Nutzer vermutlich bei 20 bis 30 liegt, möglicherweise sogar noch höher.

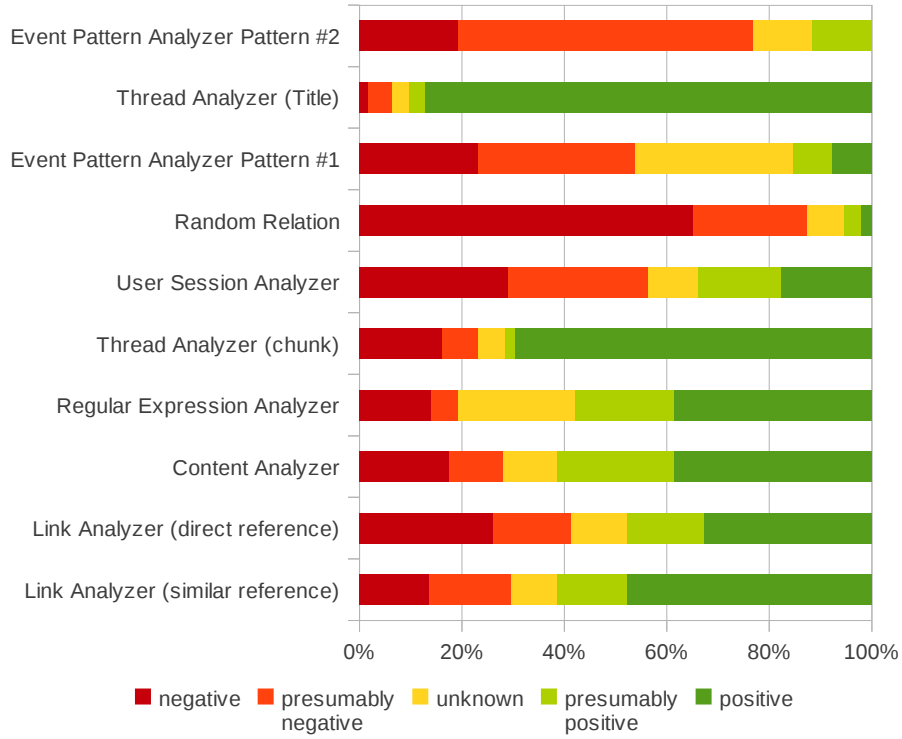


Abbildung 25: Bewertung der Relationstypen

Die Ergebnisse der Nutzerstudie sind in Abbildung 25 abgebildet, gemäß der vorgestellten *Likert-Skala* ergeben sich auf diese Weise je Relationstyp fünf unterschiedliche Kategorien, welche durch die farbliche Hervorhebung nochmals verdeutlicht werden. Die einzelnen Kategorien beziehen sich hierbei auf die Fragestellung: „Sind diese beiden Nachrichten miteinander verwandt?“. Aufgrund der Tatsache, dass die verschiedenartigen Relationstypen stets von einer größeren Menge von Nutzern evaluiert wurden, kann des Weiteren untersucht werden, inwiefern diese übereinstimmten bzw. gestreut sind. Hierfür kann beispielsweise die sogenannte *Urteilerübereinstimmung* (auch *Reliability of Agreement* oder *Interrater-Reliabilität* genannt) verwendet werden [38]. Diese hat zum Ziel, sogenannte *Konkordanzen,* das heißt das Ausmaß an Übereinstimmungen (zwischen verschiedenen Nutzern bzw. *Ratern*), in den Ergebnissen zu bewerten bzw. zu untersuchen. Für die Bestimmung der *Urteilerübereinstimmung* existieren verschiedene Berechnungsverfahren, beispielsweise mittels des sogenannten *Cohens Kappa,* respektive *Fleiss' Kappa*. Letzteres berücksichtigt im Gegensatz zu *Cohens Kappa* auch Berechnungen mit mehr als zwei Beurteilungen. Nachfolgend ist die Berechnungsformel für *Cohens Kappa* dargestellt:

$$\kappa = \frac{p_0 - p_c}{1 - p_c}$$

Je höher der resultierende Wert $\varkappa$ ist, desto besser ist die Übereinstimmung der Beurteiler. Der Wert $\varkappa$ berechnet sich hierbei aus dem Übereinstimmungswert p_0 sowie dem zufällig zu erwartenden Übereinstimmungswert p_c. Stimmen beispielsweise alle Beurteiler überein, resultiert dies in $\varkappa = 1$. Die exakte Berechungsvorschrift für *Fleiss' Kappa* findet sich in [30]. Bezüglich der Interpretation existieren verschiedenartige Skalen, eine dieser wird in [55] vorgestellt und setzt sich wie folgt zusammen (die Zahlenwerte beziehen sich jeweils auf das zu berechnende $\varkappa$):

- <0 → schlechte Übereinstimmung
- 0 - 0,20 → geringe Übereinstimmung
- 0,21 - 0,40 → ausreichende Übereinstimmung
- 0,41 - 0,60 → mittelmäßige Übereinstimmung
- 0,61 - 0,80 → beachtliche Übereinstimmung
- 0,81 - 1,00 → (fast) vollkommene Übereinstimmung

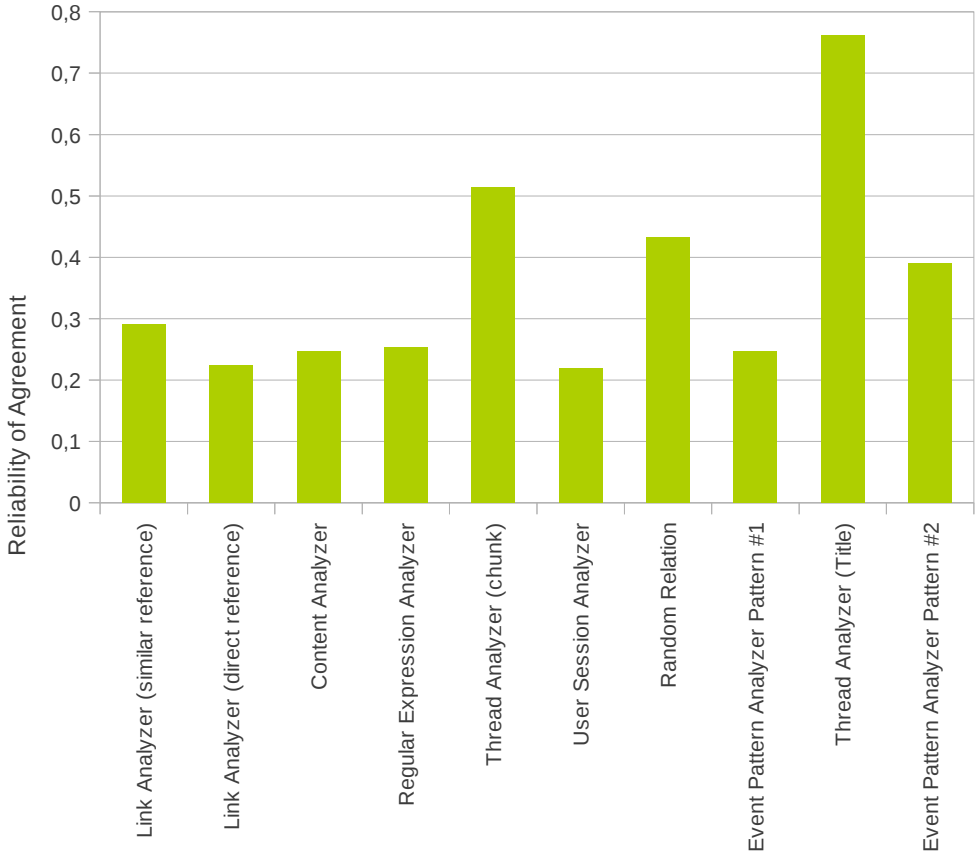


Abbildung 26: Urteilerübereinstimmung

Da *Fleiss' Kappa* von einer gleichen Anzahl Bewertungen je Objekt ausgeht, wurde der Ergebnisdatensatz auf 100 Bewertungen normalisiert, das heißt, jeder Relationstyp wird durch je 100 Beurteilungen bewertet. Die Ergebnisse zeigen, das bezogen auf das in [55] vorgestellte Maß von einer ausreichend verlässlichen Beurteilung ausgegangen werden kann bzw. die Ergebnisse nicht zu stark gestreut sind, da diese ausnahmslos größer 0,21 sind (ausreichende Übereinstimmung).
Mittels des sogenannten *Konfidenzintervalls* wurde des Weiteren ermittelt, wie verlässlich die erzielten Ergebnisse sind. Unter einem *Konfidenzintervall* versteht man einen Intervallbereich, welcher mittels des Standardfehlers und einer festzulegenden Wahrscheinlichkeit einen Bereich eines Messwertes anzeigt, in welchem dieser vermutlich liegt. Hierfür wurde festgelegt, dass eine Relation als falsch bewertet wurde, wenn diese entweder als *negative* oder *presumably negative* klassifiziert wurde. Des Weiteren wurde ein Konfidenzwert von 90% festgelegt, wodurch sich der Wert 1,644854 als Konstante inner-

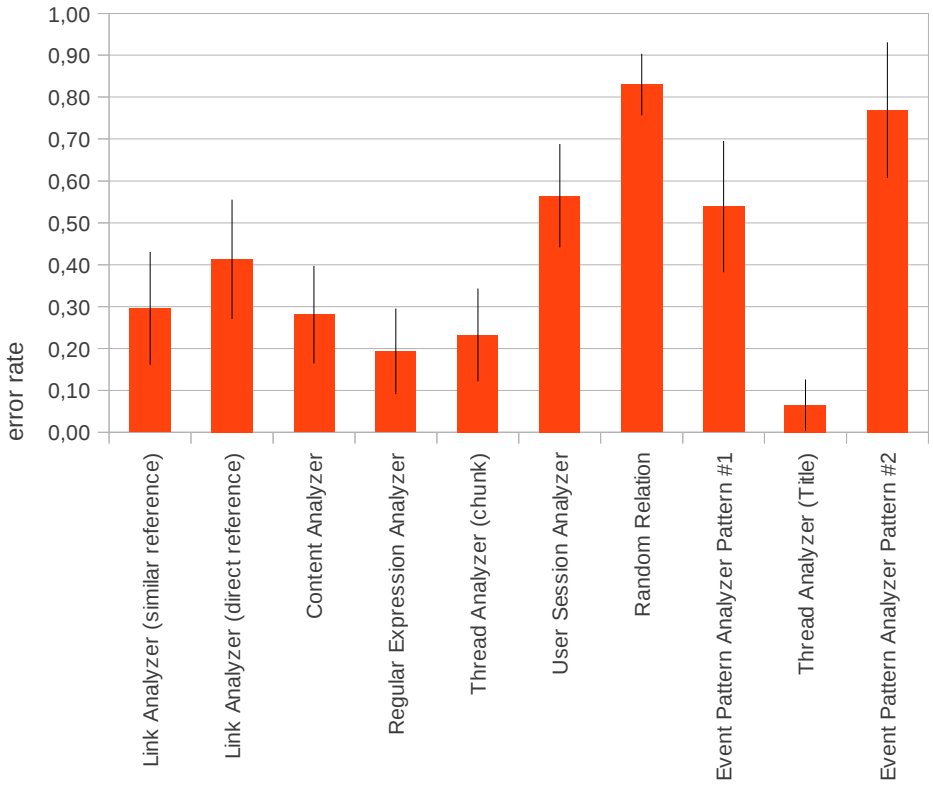


Abbildung 27: Fehlerraten und zugehörige Konfidenzintervalle

halb der Berechnungsvorschrift ergibt [60]. Bezogen auf die durch den *Link Analyzer (similar reference)* detektierten Relationen lässt sich somit beispielsweise folgende Aussage treffen: mit 90-prozentiger Wahrscheinlichkeit liegt Fehlerrate im Bereich von 16% bis 43%. Die vollständigen Ergebnisse aller Relationstypen befinden sich in Abbildung 27.

5.1.3 Diskussion

Betrachtet man die Ergebnisse in Abbildung 25, so wird deutlich, dass die Nutzer die durch RDAS extrahierten Relationen im Wesentlichen bestätigen können, bereinigt man die Nutzerstudie von den Ereignismusterrelationen *(Event Pattern Analyzer Pattern #1/2)*, so wurde im Mittel nur in circa einem Drittel aller Fälle den detektieren Relationen widersprochen.

Dies lässt die Schlussfolgerung zu, dass die Nutzer im Wesentlichen die detektierten Relationen bestätigen können. Die ausgewertete Urteilerübereinstimmung (Abbildung 26) zeigte des Weiteren, das die Nutzer bezogen auf die vorgestellte Skala, tendenziell auch eine einheitliche Meinung diesbezüglich vorweisen, respektive eine geringe Streuung erkennbar ist. Dies ist insofern relevant, als dass Bewertungen mit zu starken Tendenzen zu den jeweiligen Extrema (Relation positiv bzw. negativ) als kritisch eingestuft werden müssten bzw. deren Aussagegehalt hierdurch vermindert werden würde. Die innerhalb von Abbildung 27 analysierten Fehlerraten und deren Konfidenzintervalle zeigten, dass diese (Fehlerraten) in den meisten Fällen (>50%) mit hoher Wahrscheinlichkeit (90%) unter 50% liegen. Umgekehrt lässt dies die Schlussfolgerung zu, dass die Anzahl der falsch positiv Relationen (vgl. Abschnitt 2.3.2) zufriedenstellend gering ist. Deutlich wird, dass der *Thread Analyzer (title)* mit Abstand die besten Ergebnisse erzielte, über

90% der Nutzer konnten die Relationen bestätigen. Dieses Ergebnis ist leicht nachvollziehbar, da die meisten Nutzer vermutlich intuitiv zu erst auf den Titel einer Nachricht schauen – welches das primäre Auswertungsmerkmal dieser Relationsart darstellt.

Eine wichtige Fragestellung im Rahmen der Nutzerstudie war, inwiefern innerhalb der Konzeption weitere potentielle Relationstypen nicht beachtet wurden. Aus diesem Grund wurden den Nutzern zufällige Nachrichtenkombinationen präsentiert. Das Ergebnis zeigte jedoch deutlich, dass innerhalb dieser kaum Relationen existierten: in über 90% der Fälle werteten die Nutzer die Relationen auch als falsch bzw. negativ. Eine detaillierte Analyse der als positiv gewerteten Relationen führte jedoch zu keinen weiteren Erkenntnissen, das heißt, es konnte nicht ermittelt werden, welche Faktoren in diesen Fällen ausschlaggebend waren. Letztendlich kann ggf. auch davon ausgegangen werden, dass diese Bewertungen fälschlicherweise abgegeben wurden. Aufschluss darüber könnte letztendlich nur eine individuelle Befragung der Probanden bzgl. der zugrunde liegenden Motivation geben. Die Tatsache, dass die innerhalb der Evaluationsdaten vorhandenen Ereignismuster vergleichsweise schlechte Ergebnisse lieferten, wurde bereits ansatzweise erläutert. *Event Pattern Analyzer Pattern #1* detektierte beispielsweise das innerhalb von [19] vorgestellte Muster: „Wenn Autor A einen Checkin innerhalb des Repository tätigt und zeitnah von ihm ein Bugreport als gelöst markiert wird, so stehen diese beiden Ereignisse vermutlich in Beziehung zu einander“. Diesen Zusammenhang semantisch erfassen zu können, ist für Studienteilnehmer ohne Vorwissen bzw. domänenspezifische Kenntnisse kaum möglich.

Betrachtet man die User-Session-Analyzer-Relationen so zeigt sich, dass diese vergleichsweise selten als positiv erkannt wurden. Dies lässt die Schlussfolgerung zu, dass diese Relationsart für sich genommen nicht ausreichend ist, demnach tendenziell eher zur Verstärkung anderer Relationen genutzt werden könnte.

Eine wichtige Relationsart stellt der *Content Analyzer* bzw. die durch diesen detektierten Relationen dar. Die Ergebnisse zeigten, dass obwohl die Funktionsweise dieser Komponente relativ naiv konzipiert wurde, in fast zwei Dritteln der Fälle, die Nutzer die extrahierten Relationen bestätigen konnten. Der damit zusammenhängen Schwellenwert (vgl. Abschnitt 3.5.4) könnte somit durchaus erhöht werden, zu erwarten wäre dann in einer möglichen zweiten Nutzerstudie, dass der positive Wert weiter ansteigt. Letztendlich kann die Funktionsweise des *Content Analyzer* auch beliebig ausgebaut werden, beispielsweise durch die in [50, 77, 86, 93] vorgestellten Mechanismen, um nur einige zu nennen.

5.1.4 Kritik

Ein grundlegendes Problem der Nutzerstudie wurde bereits angedeutet: da die Nutzer keine Kenntnis über die Art der Relationen hatten, ist deren positiv oder negativ Bewertung letztendlich nur ein Indiz für deren Qualität, jedoch keine umfassende Bestätigung. Deutlich wird dies beispielsweise durch die *Link-Analyzer-Relationen*: enthält Nachricht A eine (möglicherweise sehr lange/komplexe) URL auf Nachricht B, so ist fraglich, ob ein Nutzer dies erkannt hat oder die Relation aufgrund anderer Merkmale markierte. Ähnlich verhält es sich innerhalb der durch den *Regular Expression Analyzer* detektierten Relationen, in sehr langen Texten sind die extrahieren Identifikatoren nur schwer auszumachen. Da alle Daten innerhalb das phpMyAdmin-Datensatzes letztendlich aus

einer Domäne (eben dem Projekt selbst) stammen, könnte ein Nutzer beispielsweise ausnahmslos jede Relation aufgrund dieser Tatsache als positiv kennzeichnen. In Folge dessen müsste eine konkretere Evaluierung innerhalb eines produktiven Systems erfolgen, welche von einem wesentlich umfassenderen Kontextwissen der Nutzer ausgehen könnte.

Die eingangs aufgeworfene Fragestellung bzw. die Evaluation von zufällig generierten Relationen ist ebenfalls nicht umfassend aussagekräftig bzw. ist der Umkehrschluss (im Sinne der Implikation) nicht möglich. Das heißt, die Tatsache, dass innerhalb der (größtenteils als negativ markierten) zufälligen Relationen keine positiven Relationen entdeckt wurden, bedeutet nicht, dass dies global gültig ist. Dies ist allein schon dadurch begründet, dass die Anzahl der evaluierten zufälligen Relationen nur rund 0,0000003% der Gesamtmenge darstellen. Zum Vergleich: RDAS detektiert mit den gegenwärtigen Mechanismen grob 0,03% aller theoretisch möglichen Relationen.

5.2 Intra- und Interreferentialität

Eine weitere wichtige Analyse beschäftigte sich mit der Fragestellung, in welchem Maße Relationen zwischen verschiedenartigen bzw. gleichen Informationsströmen (Quelle) auftreten. Es lassen sich somit konzeptionell zwei Arten von Relationen bezogen auf dieses Merkmal unterscheiden:

- Intrareferentielle Relationen, Nachricht $A \rightarrow B$ mit $quelle(A)=quelle(B)$
- Interreferentielle Relationen, Nachricht $A \rightarrow B$ mit $quelle(A)\neq quelle(B)$

Eine Auswertung eines Subsets des phpMyAdmin-Datensatzes ergab hierbei die in Abbildung 28 dargestellten Verteilungen. Im Mittel sind demnach mit den verwendeten Konfigurationsparametern circa 40% interreferentielle Relationen. Dieser Wert ist jedoch sehr stark kontextabhängig, beispielsweise könnte der Domänenexperte zahlreiche Ereignismuster definieren, welche explizit interreferentiell sind. Grundsätzlich bleibt bezüglich der interreferentiellen Relationen festzuhalten, das gerade dieser Relationstyp durch einen einfachen Aggregator bzw. den Nutzer selbst, sehr wahrscheinlich nicht detektierbar wäre, jedoch durch RDAS erkannt werden kann.

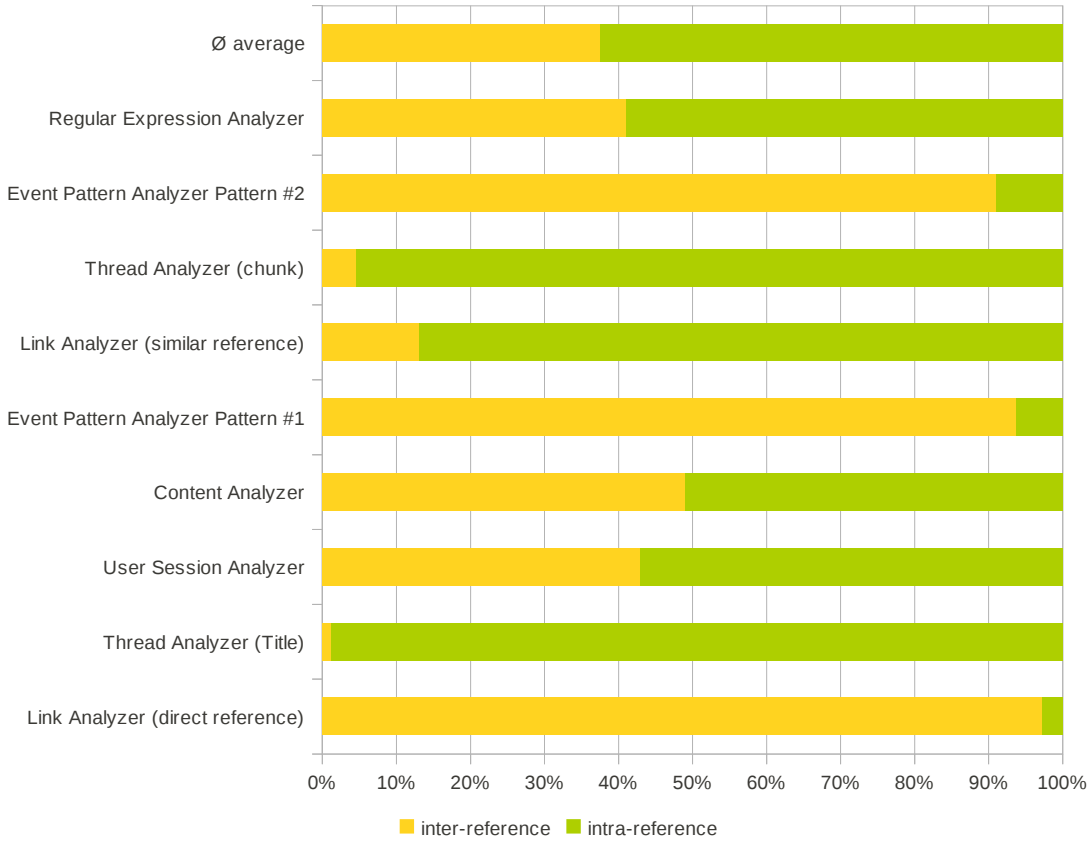


Abbildung 28: Verteilung von intra- und interreferentiellen Relationen

5.3 Performance

Neben der vorgestellten Nutzerstudie wurde des Weiteren eine Analyse der Datenverarbeitungsleistung[114] durchgeführt. Die grundsätzliche Fragestellung lautete hierbei, wie viele Nachricht das System innerhalb welcher Zeit verarbeiten kann. Dabei wird unter Verarbeitung der gesamte Prozess, das heißt jegliche Vorverarbeitungsschritte sowie die eigentliche Relationsdetektion verstanden. Eine Analyse der Ausgabe (Activity-Stream-Komponente) fand nicht statt, da deren Zeitverhalten als unkritisch eingestuft werden kann. Zunächst wurde das System mit einer 50.000 Nachrichten umfassenden Untermenge des Datensatzes gestartet und gemessen, wie lange RDAS für einen vollständigen Durchlauf benötigt, dies ist in Abbildung 29 dargestellt.

Das Ergebnis zeigte zwei Dinge: zum einen, dass die Verarbeitungsgeschwindigkeit ausreichend hoch ist: im Mittel benötigt das System pro Nachricht 1,3 Sekunden. Jedoch zeigt der Verlauf in Abbildung 29 auch, dass diese Leistung mit der Zeit abnimmt. Das dieser Effekt auftritt, ist jedoch nachvollziehbar: mit zunehmender Zeit bzw. Datenmenge vergrößern sich auch die Indizes bzw. Datenbanken generell, hierdurch steigt der Aufwand, beispielsweise für Vergleichsoperationen sowie Suchanfragen. Damit das System in zukünftigen Erweiterungen optimiert werden kann, sollte herausgefunden werden, welche Verarbeitungsschritte die Verlangsamung vergleichsweise stark beeinflussen. Eine erste Analyse versuchte daher, die Größe des Lucene Index mittels eines Schiebefensteralgorithmus auf eine maximale Größe zu beschränken, beispielsweise indem nur Nach-

114 Systemumgebung: Dual-Core AMD Opteron(tm) Processor 2224 SE, 8GB RAM, Ubuntu 10.04.2 LTS

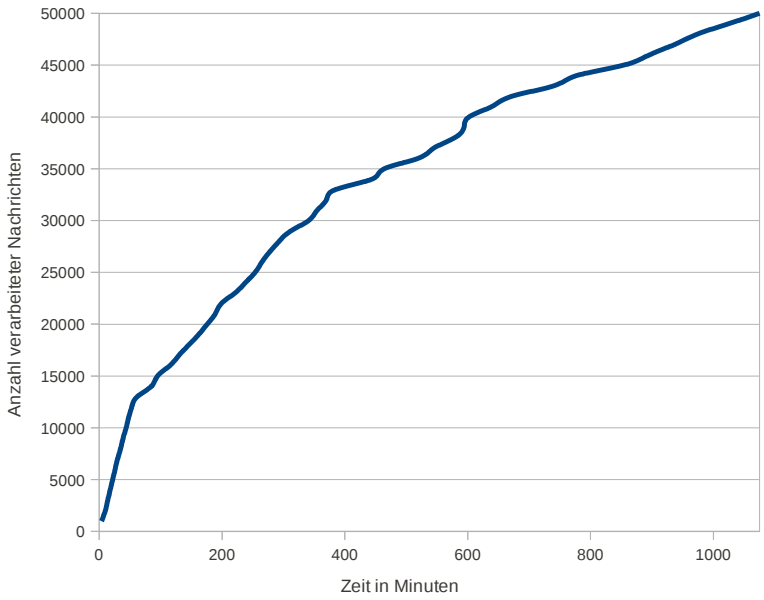


Abbildung 29: Verarbeitungsgeschwindigkeit in Abhängigkeit des Zeitverlaufs

richten eines Jahres abrufbar gehalten werden. Es stellte sich jedoch heraus, dass dies nicht zielführend war, da die zusätzlichen Operationen (löschen veralteter Einträge in zyklischen Abständen) das System sogar etwas verlangsamten. Um herauszufinden, welche Komponente ursächlich für die sinkende Verarbeitungsgeschwindigkeit verantwortlich ist, wurde anschließend ein Durchlauf mit deaktivierter Relationsdetektion sowie deaktivierter Duplikatenentfernung durchgeführt, hierbei zeigte sich, dass die innerhalb Abbildung 29 deutlich gewordene Verlangsamung auch ohne die genannten Mechanismen auftritt. Dies lies die Schlussfolgerung zu, dass andere Faktoren hierfür ausschlaggebend waren. Eine Untersuchung der Nachrichtenanzahl bzw. deren Größe machte deutlich, das beide Faktoren starken Schwankungen unterliegen und dies sehr wahrscheinlich als Ursache für die Verlangsamung in Frage kommen kann. In Anhang A.4 befinden sich vier Diagramme, welche das Zeitverhalten darstellen. Das Zeitintervall des eingangs vorgestellten 50K Subsets liegt zwischen August 2005 und November 2010, respektive Monat 55 bis 75. Der verstärke Geschwindigkeitseinbruch kann also insbesondere anhand der größeren Nachrichtenlänge erklärt werden. Diese Vermutung konnte durch einen Durchlauf über einen größeren Datenbestand anschließend bestätigt werden, dargestellt in Abbildung 30. Der erste Datensatz ist hierbei Teil der Analyse und beginnt circa bei Nachricht 45K. Es wird somit deutlich, dass die Verlangsamung hier erst bei Nachricht 60K eintritt.

Grundsätzlich kann jedoch festgehalten werden, das die theoretisch erforderliche Datenverarbeitungsgeschwindigkeit weitaus niedriger liegt: betrachtet man das Zeitverhalten im phpMyAdmin-Datensatz so zeigt sich, dass in diesem im Mittel (bezogen auf den gesamten Zeitraum von circa 10 Jahren) pro Sekunde rund 0,0007 Nachrichten auftraten, respektive aller 22 Minuten eine Nachricht emittiert wurde. In diesem Kontext erscheint die gegenwärtige Leistung von RDAS mit grob einer Nachricht pro Sekunde als ausreichend.

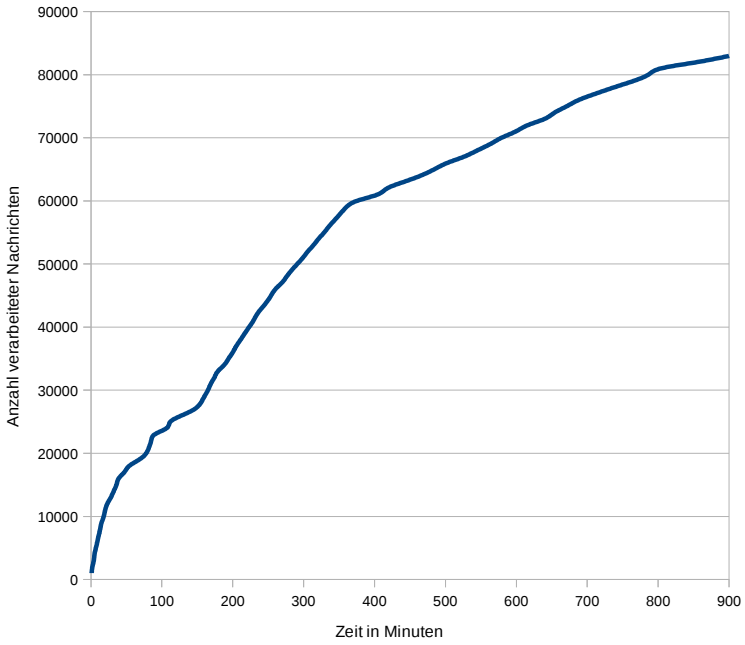


Abbildung 30: Verarbeitungsgeschwindigkeit in Abhängigkeit des Zeitverlaufs

5.4 Zusammenfassung

Das vorangegangene Kapitel untersuchte einige der Kernaspekte von RDAS mittels einer Nutzerstudie sowie einer Analyse der Datenverarbeitungsgeschwindigkeit. Des Weiteren erfolgte eine kurze Betrachtung der Interreferentialität der detektierten Relationen.

Innerhalb der Nutzerstudie konnte gezeigt werden, dass die durch RDAS ermittelten Relationen zu überwiegenden Teilen von den Nutzern erkannt bzw. akzeptiert werden. Eine Untersuchung der Urteilerübereinstimmung zeigte hierbei, dass die Streuung der Ergebnisse in einem akzeptablen Bereich liegt – die Nutzer sich demnach überwiegend einig waren. Eine Analyse der Konfidenzintervalle machte zudem deutlich, dass die aus der Nutzerstudie resultierenden Fehlerraten von RDAS in den meisten Fällen (mit hoher Wahrscheinlichkeit) unter 50% liegen – bezogen auf die Trefferquote bedeutet dies, dass die erkannten Relationen größtenteils als korrekt angesehen wurden. Dennoch sind diese Werte stark vom jeweiligen Relationstyp abhängig, das heißt, eine Berücksichtigung dieser Ergebnisse sowie eine Filterung des *confidence* Wertes (vgl. Abschnitt 3.5.1) würden in einem produktiven System weitere Verbesserungen ermöglichen.

Die Analyse der Datenverarbeitungsgeschwindigkeit machte deutlich, dass das System grundsätzlich ausreichend performant ist, jedoch in Abhängigkeit der Nachrichtenstruktur Optimierungsbedarf bestehen könnte, sofern an die Verarbeitungsleistung höhere Anforderungen bestehen.

Ferner wurde durch eine Untersuchung der Interreferentialität deutlich, dass ein wesentlicher Teil der erkannten Relationen nicht innerhalb eines Informationsstroms, sondern quellenübergreifend auftritt. Dies ist insofern relevant, als dass gerade dieser Relationstyp durch einen einfachen Aggregator bzw. den Nutzer sehr wahrscheinlich nicht detektierbar ist – durch RDAS jedoch potentiell erkannt werden kann.

6 Zusammenfassung

Ziel des folgenden Kapitels ist die rückblickende Betrachtung der erarbeiteten Konzepte bzw. deren Umsetzung. Weiterhin soll ein Ausblick gegeben werden, welcher Erweiterungsmöglichkeiten aufzeigt und Grenzen des Systems beschreiben soll.

6.1 Retrospektive

Innerhalb des Einleitungskapitels wurde die der Arbeit zugrunde liegende Motivation und die Ziele, welche sich aus dieser ergebenen, dargestellt. Darauf aufbauend wurden einige Anwendungsfälle herausgearbeitet um mittels diesen zentrale Fragestellungen formulieren zu können. Abschließend konnten mit diesen wesentliche Anforderungen, an dass im Rahmen der Arbeit zu realisierende System, konkretisiert werden.

Kapitel 2 lieferte zunächst ein Überblick über verschiedene Web-Feed-Formate und deren Merkmale sowie eine historische Betrachtung. Im darauf folgenden Abschnitt wurde das daraus hervorgegangene Activity-Stream-Format im Detail erläutert um im Anschluss daran grundlegende Begrifflichkeiten des Information Retrieval vorzustellen. Die darauf folgenden Kapitel 2.4 sowie 2.5 beschäftigen sich mit Struktur und Merkmalen relevanter Quelldaten sowie deren potentielle Nachrichtenbeziehungen. Abschließend erfolgte eine Analyse von vorhandenen Systemen (State of the Art), welche eine Einordnung des konzipierten Systems ermöglichte.

Im nachfolgenden Kapitel wurde die Konzeption des RDAS-Systems vorgestellt, indem zunächst die Systemarchitektur und daran anschließend die wesentlichen Mechanismen sowie deren Komponenten im Detail diskutiert wurden. Dies beinhaltete eine umfassende Beschreibung der Relationsdetektion sowie deren Umsetzung innerhalb der einzelnen Komponenten.

Eine Beschreibung der prototypischen Umsetzung der zuvor erarbeiteten Konzepte erfolgte anschließend in Kapitel 4. Hierbei wurde die zugrunde liegende Technologiebasis vorgestellt und auf welche Art und Weise die einzelnen Komponenten auf diese aufbauen bzw. wie diese realisiert wurden. Neben einzelnen Designentscheidungen wurde dabei auch auf aufgetretene Probleme und deren Lösungen eingegangen.

Kapitel 5 beschrieb die Evaluation des entstandenen Systems aufbauend auf einer durchgeführten Nutzerstudie sowie deren Auswertung. Diese hatte zum Ziel, die wesentlichen Mechanismen bzw. Konzepte der Relationsdetektion durch Dritte bewerten zu lassen, um aufbauend auf diesen Erkenntnissen, Hinweise auf die Qualität sowie Gebrauchstauglichkeit der analysierten Relationsarten erhalten zu können. Des Weiteren wurde eine Betrachtung der Datenverarbeitungsleistung des entstandenen Systems vorgestellt.

6.2 Ergebnisse

Zu Beginn dieser Arbeit wurden innerhalb des Abschnitts 1.4 die wesentlichen Fragestellungen vorgestellt. Diese sollen nun nachfolgend mit den gewonnenen Erkenntnissen dis-

kutiert werden.

Welche Strategien können angewendet werden, um die den Nachrichten zugrunde liegenden Aktivitäten extrahieren und modellieren zu können?

Der erste Teil dieser Fragestellung wurde in Abschnitt 3.6.1 diskutiert, mit dem Ergebnis, dass eine vollständig automatisierte Erkennung von Relationen selbst mit komplexen NLP-Systemen letztendlich nur eine „semantische Aktivität“ extrahieren kann, jedoch nicht die innerhalb der Activity-Stream-Spezifikation benötigte „syntaktische Aktivität“. Aus diesem Grund wurde ein Event-Processing-Annotierungskonzept vorgestellt, welches es dem Domänenexperte ermöglicht, Nachrichten semi-automatisch mit aktivitätsbezogener Semantik anzureichern. Dies beantwortet auch die Fragestellung nach der Modellierung – die entsprechenden Anweisungen können mittels der vorgestellten EPL-Beschreibungssprache (vgl. Abschnitt 2.6.4) definiert werden.

Wie können Relationen zwischen einzelnen Nachrichten detektiert werden?

Innerhalb der Grundlagen wurden in Abschnitt 2.5 zahlreiche Relationsarten vorgestellt, um anschließend innerhalb der Konzeption, in Abschnitt 3.5, mögliche Detektionsmechanismen zu erörtern. Dabei wurde herausgearbeitet, dass zahlreiche unterschiedliche Relationsmerkmale existieren, welche jeweils eigenständige Erkennungsmechanismen benötigen. Grundsätzlich kann eine Detektion auf zwei Wegen erfolgen: mittels Event-Processing-Mechanismen sowie mittels Data-Mining-Mechanismen. Erstere sind quasi Echtzeitsysteme, welche die Definition von komplexen Ereignismustern erlauben, wohingegen letztere Such- und Vergleichsoperationen innerhalb des gespeicherten Datenbestandes vornehmen können. Abschließend bleibt festzuhalten, dass RDAS so konzipiert wurde, dass beliebige weitere Relationsarten detektiert und verarbeitet werden können.

Welche Arten von Relationen existieren?

Diese Fragestellung wurde in Abschnitt 2.5 sowie 3.5 diskutiert, siehe auch vorheriger Abschnitt. Grundsätzlich kann zwischen domänenabhängigen- und domänenunabhängigen Relationen unterschieden werden. Auf konzeptioneller Ebene werden die einzelnen Relationstypen von RDAS jedoch als gleichwertig betrachtet. Zukünftige Erweiterungen könnten hier eine Typisierung integrieren, das heißt, Relationen werden Kategorien (beispielsweise der Form „ist-verwandt-zu“ oder „ausgelöst-durch“) zugeordnet, welche beispielsweise innerhalb der Benutzeroberfläche visualisiert werden könnten. Diese Typisierung könnte als weiteres Attribut innerhalb einer EPL-Anweisung abgebildet werden.

Welche Möglichkeiten existieren, um domänenspezifische Relationsstrukturen beschreiben zu können?

Die Adaptierbarkeit von RDAS auf verschiedenartige Domänen und deren Anforderungen wurde mittels einer Ereignismusterbeschreibungssprache (EPL) sowie zahlreichen Konfigurationsparametern sichergestellt. Grundsätzlich existieren im Bereich der domänenabhängigen Relationen zwei Mechanismen: einerseits die deklarative Beschreibung von besagten Ereignismustern und anderseits die Definition komplexer Syntaxmuster mittels regulären Ausdrücken.

Welche Voraussetzungen müssen geschaffen werden, damit das System in verschiedenartigen Umgebungen und deren Strukturen domänenunabhängig instantiiert werden kann?

Durch den Einsatz der im vorherigen Abschnitt erwähnten Ereignismusterbeschreibungssprache, wurde eine weitreichende Domänenunabhängigkeit sichergestellt. Des Weiteren wurde ein Konzept vorgestellt und implementiert, welche deren Mechanismen für eine Annotierung der Nachrichten mit aktivitätsbezogener Semantik ausnutzt und wieder verwendet. Jegliche Angaben erfolgen in einer externen XML-Konfigurationsdatei, auf diese Weise ist eine Adaptierbarkeit gewährleistet. Eine konkrete Instantiierung bestünde demnach aus zwei Schritten: Analyse der Beziehungsmerkmale der Domäne sowie Formalisierung dieser in konkreten EPL-Anweisungen. Für den Fall, dass in einer Domäne potentielle Beziehungsmerkmale nicht bekannt sind, würden dennoch die domänenunabhängigen RDAS-Mechanismen greifen, insbesondere in Form des Content Analyzers. Sollten zukünftige Entwicklungen einen Schwerpunkt stärker auf ein unüberwachtes (RDAS) System legen, welches ohne Ereignismusterdefinitionen auskommt, so wäre eine Integration von Mechanismen beispielsweise aus [77, 93] empfehlenswert.

> *Sind die von den Nachrichtenquellen gelieferten Informationen für ein automatisiertes System ausreichend oder müssen zusätzlichen (Meta-) Daten bereitgestellt werden?*

Eine umfassende Antwort dieser Fragestellung hängt letztendlich von der konkreten Nachrichtenstruktur der jeweiligen Domäne ab. Innerhalb des in dieser Arbeit verwendeten phpMyAdmin-Datensatzes konnte gezeigt werden, dass die Nachrichten ausreichend Information bereitstellen, um Aktivitäten (beispielsweise Objekttypen, Target etc. siehe Abschnitt 2.2.2) modellieren zu können. Grundsätzlich sieht das Konzept von RDAS die Angabe von Annotierungsvorschriften als zentrale Komponente vor.

6.3 Ausblick

Während der Konzeption bzw. der anschließenden Implementierung wurden verschiedene mögliche Erweiterungen und Verbesserungen des RDAS Systems erkannt. Nachfolgend soll auf diese Bezug genommen werden, auch im Hinblick auf Grenzen des Systems.

- Aufgrund der Tatsache, dass das Activity-Stream-Format gegenwärtig eine nur geringe Verbreitung vorweist, berücksichtigt das konzipierte System nicht den Umstand, dass als Eingabestrom bereits ein Activity Stream vorliegen kann, welcher eine aktivitätsbezogene Annotierung überflüssig machen würde.
- Die Konfiguration von RDAS, insbesondere die Definition der Annotierungen sowie Ereignismustern, erfolgt mittels einer manuellen Angabe entsprechender Informationen in einer XML-Datei. Diese Konfiguration sollte in einem produktiven System mittels einer grafischen Benutzeroberfläche erfolgen können, welche die Wahrscheinlichkeit von fehlerhaften Angaben (Syntax) stark minimieren würde.
- Ein Problem von RDAS stellen einige Information-Retrieval-basierende Konfigurationsparameter dar. Deren Werte basieren letztendlich auf Heuristiken bzw. Erfahrungswerten und müssten in einem produktiven System angepasst werden. Grundsätzlich sind diese Werte aber nur bedingt nachvollziehbar bzw. dürfte es für einen Domänenexperten schwer sein, einen konkreten Bezug zwischen Parameter und Wirkung herzustellen, dies könnte durch Machine-Learning-Ansätze

verbessert werden.

- Die grundlegende Unterteilung des Systems in eine Data-Mining und Event-Processing-Komponente stößt in einigen Anwendungsfällen an ihre Grenzen, insbesondere wenn Elemente beider Komponenten miteinander kombiniert werden sollen. Beispielsweise wäre ein Szenario denkbar, in welchem eine Ereignismuster nur positiv sein soll, wenn zwei Nachrichten auch inhaltlich ähnlich sind, das heißt, das Ergebnis des Content Analyzers soll als Parameter innerhalb der Ereignismusterbeschreibungssprache zur Verfügung stellen. Um dies zu ermöglichen, müsste die Esper EPL bzw. Esper selbst tiefer gehend analysiert und erweitert werden.
- Die gegenwärtigen Konzepte der einzelnen Relationsarten können beliebig ausgebaut werden, insbesondere stellt der realisierte Mechanismus des Content Analyzers ein vergleichsweise einfaches Konzept dar. Es existieren zahlreiche Ansätze auf dem Gebiet des Information Extraction, beispielsweise die in [77] skizzierten Mechanismen, welche Relationen basierenden auf abstrakteren Ebenen ermöglichen. Hierbei zeigte sich auch, dass im Kontext von beispielsweise Softwareentwicklungsprojekten die entsprechenden Daten zum einen sehr oft identische Schlüsselbegriffe enthalten sowie abstrakte/technische Bezeichnungen verwenden. Dies stellt höhere Anforderungen an eine Entitätserkennung als tendenziell natürlichsprachlichere Texte, Extraktionsmechanismen wie beispielsweise in [91] vorgestellt, erscheinen in diesem Kontext nur bedingt hilfreich.
- Die prototypische Implementierung hat in Hinblick auf den produktiven Einsatz noch Verbesserungspotential, beispielsweise sind gegenwärtig im laufenden Betrieb keine Erweiterungen der Ereignismusterdefinitionen möglich. Eine Lösung der meisten dieser Sachverhalte erscheint jedoch unproblematisch.
- Wie bereits in Abschnitt 3.7 angedeutet wurde, stellt das gegenwärtige Visualisierungskonzept mittels Thread Arcs einen eher pragmatischen Ansatz dar, welcher mit steigender Anzahl an Relationen problematisch erscheint, beispielsweise in Hinblick auf die benötige Darstellungsfläche. Eine detaillierter Evaluation weiterer potentieller Konzepte erscheint demnach sinnvoll.
- Steht innerhalb einer Domäne eine detaillierte Ontologie (vgl. Abschnitt 2.5.3.2) zur Verfügung, könnte RDAS dahingehend erweitert werden, dass deren Beziehungsmerkmale automatisiert verarbeitet werden und eine Beschreibung innerhalb der EPL auf diese Weise entfällt. Dies setzt jedoch eine ausreichende Annotierung der Nachrichten voraus. Grundsätzlich könnten die Informationen einer Ontologie jedoch auch mittels einer EPL formuliert werden.

Mit dem entwickelten System konnte gezeigt werden, dass innerhalb der betrachteten Domäne zahlreiche Beziehungen zwischen Nachrichten detektierbar sind, einerseits reichen hierfür bereits (partiell) sehr einfache Ansätze aus - andererseits könnten Erweiterungen einen stärkeren Semantikbezug herstellen. Grundsätzlich wurde jedoch mittels der prototypischen Umsetzung und einer anschließenden Nutzerstudie die Gebrauchstauglichkeit der konzipierten Mechanismen gezeigt bzw. konnte bestätigt werden.

A Anhang

A.1 Web-Feed-Formate

```
1   <?xml version="1.0" encoding="UTF-8" ?>
2   <rss version="2.0">
3   <channel>
4      <title>RSS Title</title>
5         <description>This is an example of an RSS feed</description>
6         <link>http://www.someexamplerssdomain.com/main.html</link>
7         <lastBuildDate>Mon, 06 Sep 2010 00:01:00 +0000</lastBuildDate>
8         <pubDate>Mon, 06 Sep 2009 16:45:00 +0000 </pubDate>
9
10        <item>
11           <title>Example entry</title>
12           <description>Here is some text</description>
13           <link>http://www.wikipedia.org/</link>
14           <pubDate>Mon, 06 Sep 2009 16:45:00 +0000 </pubDate>
15        </item>
16
17  </channel>
18  </rss>
```

Listing 17: Beispiel eines RSS 2.0 Feeds

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <feed xmlns="http://www.w3.org/2005/Atom">
3    <author>
4      <name>Autor des Weblogs</name>
5    </author>
6    <title>Titel des Weblogs</title>
7    <id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0af6</id>
8    <updated>2003-12-14T10:20:09Z</updated>
9
10   <entry>
11     <title>Titel des Weblog-Eintrags</title>
12     <link href="http://example.org/2003/12/13/atom-beispiel"/>
13     <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
14     <updated>2003-12-13T18:30:02Z</updated>
15     <summary>Zusammenfassung des Weblog-Eintrags</summary>
16     <content>Volltext des Weblog-Eintrags</content>
17   </entry>
18 </feed>
```

Listing 18: Beispiel eines Atom Feeds

```
<entry xmlns="http://www.w3.org/2005/Atom"
  xmlns:activity="http://activitystrea.ms/spec/1.0/">
  <id>tag:photopanic.example.com,2009:/activity/4859/1643</id>
  <title>Geraldine posted a photo to the My Pets album.</title>
  <published>2010-06-21T00:28:35Z</published>
  <link rel="alternate" type="text/html"
   href="http://example.com/geraldine/activities/1643" />
  <author>
    <name>Geraldine</name>
    <uri>http://example.com/geraldine</uri>
    <id>tag:photopanic.example.com,2009:person/4859</id>
    <activity:object-type>person</activity:object-type>
    <link rel="alternate" type="text/html" href="http://example.com/geraldine" />
  </author>
  <activity:verb>post</activity:verb>
  <activity:object>
    <id>tag:photopanic.example.com,2009:photo/1643</id>
    <title>My Cat</title>
    <link rel="alternate" type="text/html" href="/geraldine/photos/1643" />
    <link rel="preview" type="image/jpeg"
     href="/geraldine/photos/1643/thumb.jpg" />
    <link rel="enclosure" type="image/jpeg"
     href="/geraldine/photos/1643/full.jpg" />
    <activity:object-type>photo</activity:object-type>
  </activity:object>
  <activity:target>
    <id>tag:photopanic.example.com,2009:photo-album/2519</id>
    <title>My Pets</title>
    <link rel="alternate" type="text/html" href="/geraldine/albums/pets" />
    <activity:object-type>photo-album</activity:object-type>
  </activity:target>
  <content type="xhtml">...</content>
</entry>
```

Listing 19: Beispiel eines Activity-Stream-Eintrages

A.2 Activity Stream

Komponente	Beschreibung
Time	Definiert den Zeitpunkt, zu welchem die entsprechende Aktivität stattgefunden hat, dies muss nicht mit der Generierung des Eintrages innerhalb des Activity Streams korrespondieren.
Actor	Ein Akteur entspricht einem einem Objektkonstrukt (siehe nachfolgender Abschnitt) welcher die Entität definiert, welche die Aktivität ausführt.
Verb	Identifiziert die eigentliche Aktion einer Aktivität. Die möglichen Ausprägungen dieses Elementes sind durch Verwendung von IRIs[115] innerhalb des Namensraumes klar abgegrenzt bzw. definiert und können demnach nicht wahlfrei gesetzt werden.
Objekt	Identifiziert das primäre Objekt einer Aktivität (ebenfalls mittels eines Objektkonstruktes)
Target	Beschreibt ein Zielobjekt (Objektkonstrukt), auf welches die entsprechende Aktivität angewendet wird. Beispielsweise „Bob löschte ein Bild in dem Album Urlaub“, das „Album Urlaub“ ist in diesem Fall das Target
Title	Eine HTML Repräsentation der Aktivität, welche als Fallback genutzt werden kann, falls eine Anwendungen keine Activity-Stream-Einträge versteht bzw. kennt.
Summary	Ähnlich dem Titel Element, liefert es einen Fallback Mechanismus. Das Summary Element ist aber komplexer, beispielsweise enthält dieses auch Bilder.

Tabelle 4: Komponenten des Aktivitätskonstrukt

115 Ein IRI ist eine internationalisierte Form der Uniform Resource Identifier (URI), welche im RFC 3987 spezifiziert ist. IRIs umfassen einen größeren Zeichensatz, wodurch wesentlich mehr Sprachen korrekt abgebildet werden können.

Komponente	Beschreibung
ID	Mittels einer eindeutigen IRI wird jedes Objekt identifiziert. Dieses Feld ist nicht obligatorisch, es kann auf die schwächere Permalink URL zurück gegriffen werden, welche jedoch keine umfassende Eindeutigkeit des Objektes garantiert.
Name	Der Name des Objektes in für lesbarer Form. Objekte müssen nicht zwingend einen derartigen Namen haben.
Summary	Analog Name repräsentiert Summary eine Zusammenfassung des Objektes in lesbarer Form, ähnlich der Summary eines Aktivitätskonstruktes.
Representative Image	Eine visuelle Repräsentation des Objektes, welche mittels einer IRI definiert wird. Beispielsweise das Profilbild eines Akteurs.
Permalink URL	Dieses bereits unter ID erwähnte Feld ist ebenfalls eine IRI Referenz, welche auf eine HTML Repräsentation der Aktivität verweist. Beispielsweise die Profilseite eines Akteurs.
Object Type	Identifiziert den Typ des Objektes mittels einer eindeutigen IRI innerhalb des Activity-Stream-Namensraumes. Entgegen der intuitiven Annahme, ist die Angabe eines Objekttyps nicht obligatorisch, jedoch hat die konsumierende Endanwendung des Activity Stream in diesem Fall keine Kenntnis darüber, was genau das Objekt darstellt. Der Objekttyp 'photo' ermöglicht beispielsweise die Formulierung der Aktivität „Bob postet a photo 'My Dog Rex'“. Wäre der Typ nicht definiert, könnte die Anwendungen lediglich „Bob postet 'My Dog Rex'“ generieren.

Tabelle 5: Komponenten des Objektkonstruktes

A.3 Zusammensetzung phpMyAdmin-Datensatz

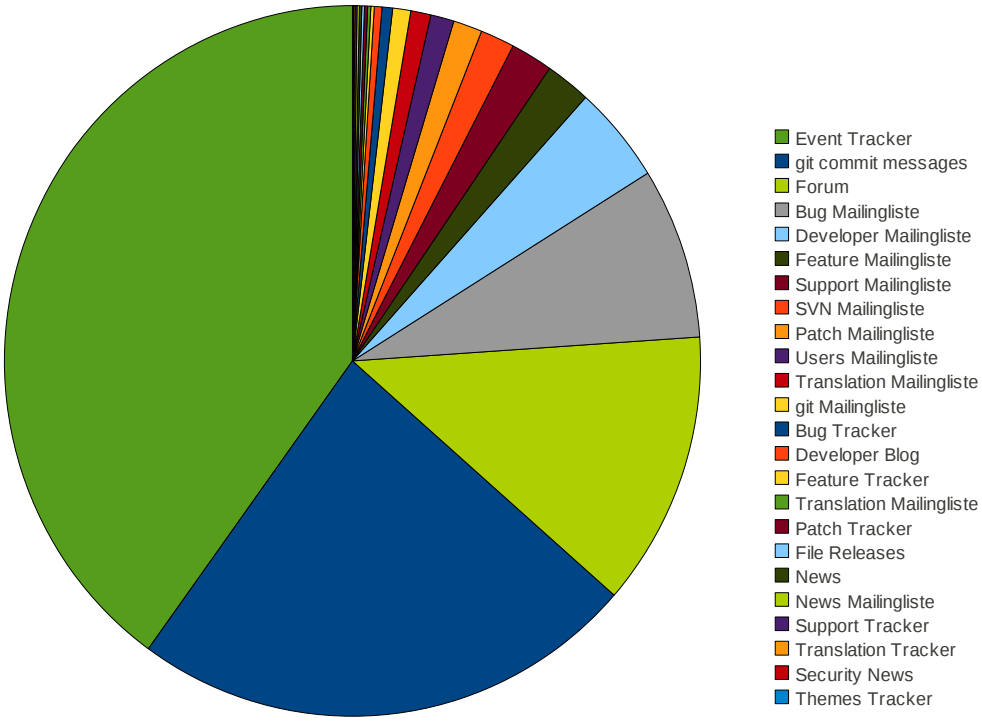


Abbildung 31: Anzahl der Nachrichten im phpMyAdmin-Datensatz nach Quelle

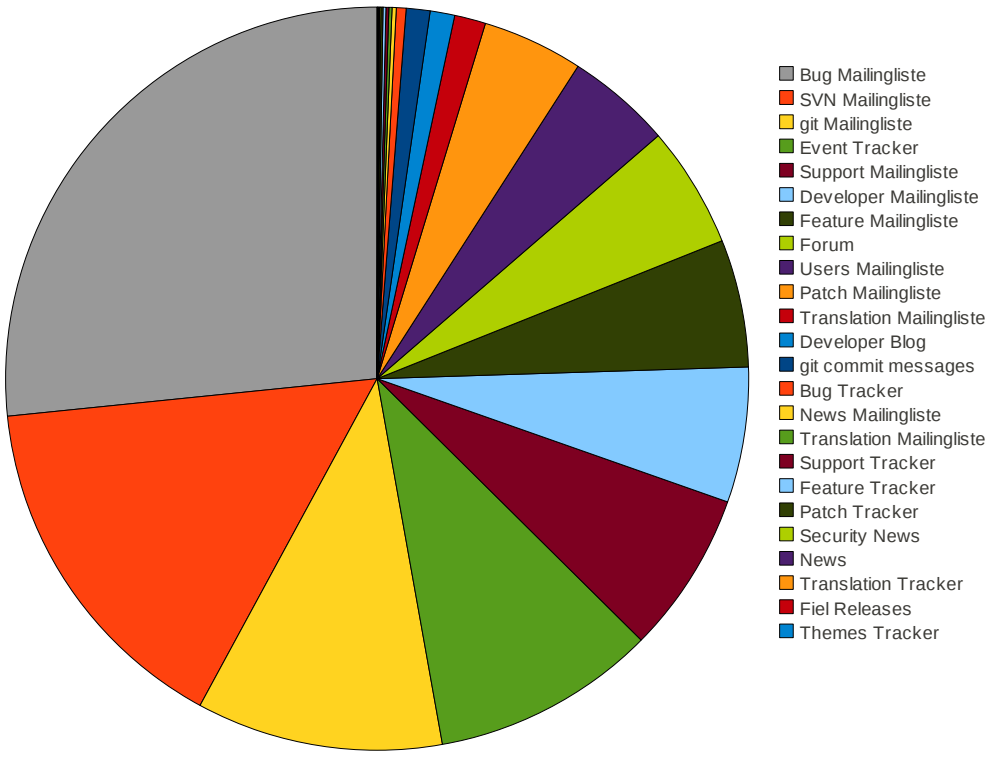
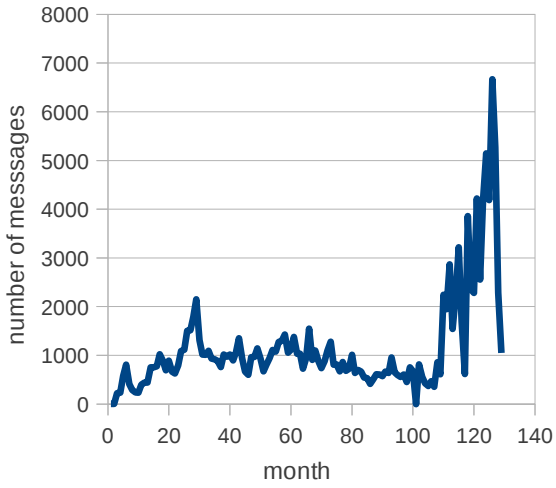
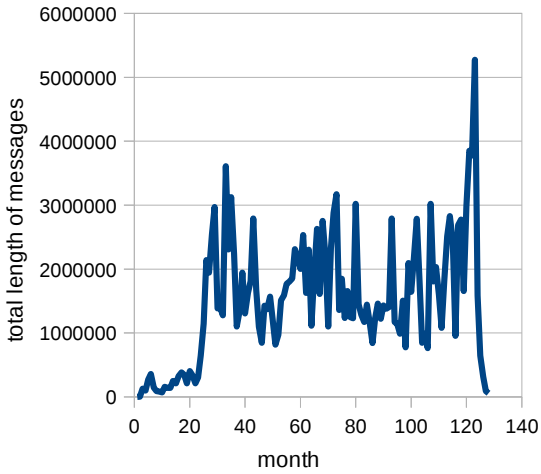
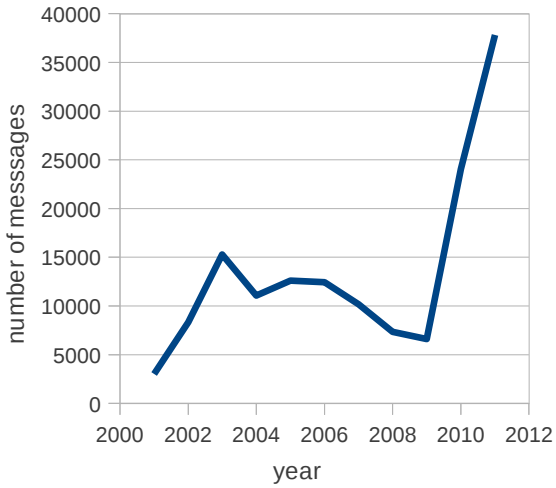


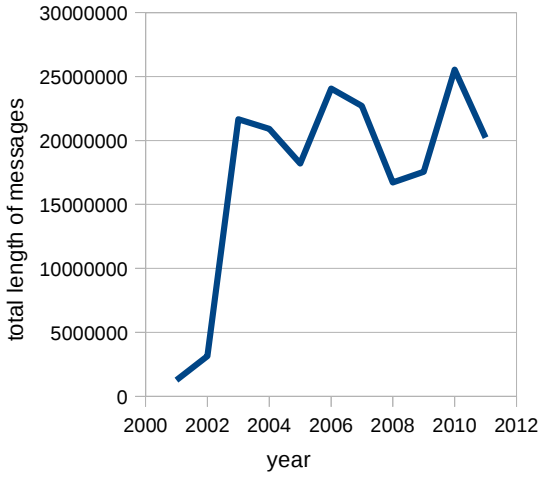
Abbildung 32: Kumulierte Nachrichtengröße im phpMyAdmin-Datensatz nach Quelle

A.4 Zeitverhalten phpMyAdmin-Datensatz









Listingverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CED	Complex Event Detection
CEP	Complex Event Processing
CSV	Comma Separated Values
DAG	Directed Acyclic Graph
DBCP	Database Connection Pool
DBMS	Datenbankmanagementsystem
DTD	Document Type Definition
EPL	Event Processing Language
ERD	Entity-Relationship-Diagramm
ESP	Event Stream Processing
FIFO	First In – First Out
FLOSS	Free/Libre Open Source Software
GNU	rekursives Akronym von GNU's Not Unix
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDF	Inverse Dokumentfrequenz
IETF	Internet Engineering Task Force
IRI	Internationalized Resource Identifier
ITS	Issue Tracking Systemen
JSON	JavaScript Object Notation
NER	Named Entity Recognition
NLP	Natural Language Processing
RDAS	Relation Detection [and] Activity Stream [Transformation]
RDF	Resource Description Framework
REST	Representational State Transfer
RFC	Request for Comments
RSS	unter anderem: Really Simple Syndication
SEP	Simple Event Processing
SPO	Subjekt Prädikat Objekt
SQL	Structured Query Language
SVN	Subversion
URL	Uniform Resource Locator
XHTML	eXtensible HyperText Markup Language
XML	Extensible Markup Language
XSD	XML Schema Definition

Literaturverzeichnis

[1] 21 Ways to Visualize and Explore Your Email Inbox: *http://flowingdata.com/2008/03/19/21-ways-to-visualize-and-explore-your-email-inbox/*. Accessed: 2011-11-15.

[2] 35 Ways to Stream Your Life: *http://www.readwriteweb.com/archives/35_lifestreamin_apps.php*. Accessed: 2011-07-18.

[3] A Survey of Event Processing Languages (EPLs), October 7, 2006: 2006. *http://www.slideshare.net/TimBassCEP/a-survey-of-event-processing-languages-epls-october-7-2006-presentation*. Accessed: 2011-10-26.

[4] Activity Base Schema (Draft): *http://activitystrea.ms/head/activity-schema.html*. Accessed: 2011-10-31.

[5] Activity Streams - a format for syndicating social activities around the web: *http://activitystrea.ms/*. Accessed: 2011-06-26.

[6] Activity Streams Working Group: Atom Activity Streams 1.0: 2011. *http://activitystrea.ms/specs/atom/1.0/*. Accessed: 2011-07-13.

[7] Andronikos, T. et al. 2009. Adding Temporal Dimension to Ontologies via OWL Reification. *Informatics, Panhellenic Conference on* (Los Alamitos, CA, USA, 2009), 19-22.

[8] Banos, E. et al. 2006. PersoNews: A Personalized News Reader Enhanced by Machine Learning and Semantic Filtering. *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*. R. Meersman and Z. Tari, eds. Springer Berlin Heidelberg. 975-982.

[9] Bar-Yossef, Z. et al. 2009. Do not crawl in the DUST. *ACM Transactions on the Web*. 3, 1 (Jan. 2009), 1-31.

[10] Bartelme Design | RSS Reading Preferences, Part 2: *http://bartelme.at/journal/archive/rss_reading_preferences_part_2*. Accessed: 2011-11-24.

[11] Benchmarks & CEP | cloudeventprocessing.com: *http://blog.cloudeventprocessing.com/2010/01/05/benchmarks-cep/*. Accessed: 2011-10-26.

[12] Bewertung von Webseiten durch Google: *http://homepage.univie.ac.at/Franz.Embacher/Lehre/aussermathAnw/Google.html*. Accessed: 2011-10-31.

[13] Bhavnani, S.K. and Wilson, C.S. 2010. Information Scattering. *Encyclopedia of Library and Information Sciences, Third Edition*. (2010), 2564.

[14] Bruns, R. et al. 2010. *Event-Driven Architecture: Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse*. Springer.

[15] Changing Bits: Lucene's near-real-time search is fast!: *http://blog.mikemccandless.com/2011/06/lucenes-near-real-time-search-is-fast.html*. Accessed: 2011-11-23.

[16] Comparison of issue-tracking systems - Wikipedia, the free encyclopedia:

http://en.wikipedia.org/wiki/Comparison_ of_ issue_ tracking_ systems. Accessed: 2011-09-07.

[17] Complex Event Processing: *http://www.complexevents.com/*. Accessed: 2011-10-22.

[18] Cornell Database Group - Cayuga: *http://www.cs.cornell.edu/bigreddata/cayuga/*. Accessed: 2011-09-04.

[19] Cubranic, D. et al. 2005. Hipikat: a project memory for software development. *IEEE Transactions on Software Engineering*. 31, 6 (Jun. 2005), 446-465.

[20] Dang, Q.V. et al. 2009. Supporting Situation Awareness in FLOSS Projects by Semantical Aggregation of Tools Feeds. *Signal-Image Technologies and Internet-Based System, International IEEE Conference on* (Los Alamitos, CA, USA, 2009), 423-429.

[21] Daring Fireball: An Improved Liberal, Accurate Regex Pattern for Matching URLs: *http://daringfireball.net/2010/07/improved_ regex_ for_ matching_ urls*. Accessed: 2011-10-30.

[22] Edmunds, A. and Morris, A. 2000. The problem of information overload in business organisations: a review of the literature. *International Journal of Information Management*. 20, 1 (Feb. 2000), 17-28.

[23] Endsley, M. 1995. Toward a theory of situation awareness in dynamic systems: Situation awareness. *Human factors*. 37, 1 (1995), 32-64.

[24] Erera, S. and Carmel, D. 2008. Conversation detection in email systems. *Proceedings of the IR research, 30th European conference on Advances in information retrieval* (Berlin, Heidelberg, 2008), 498–505.

[25] Esper - Event Stream and Complex Event Processing for Java: *http://esper.codehaus.org/esper-4.3.0/doc/reference/en/html_ single/index.html*. Accessed: 2011-10-22.

[26] Esser, A. 2009. *Der Dijkstra-Algorithmus zur Berechnung kürzester Wege in Graphen: Vorgehen bei der Implementierung des Algorithmus, mögliche Fehlerquellen, Datenstrukturänderungen, Beobachtungen zur Laufzeit*. GRIN Verlag.

[27] Facebook Gets a Facelift | Facebook: *http://blog.facebook.com/blog.php?post=2207967130*. Accessed: 2011-09-04.

[28] Ferber, R. 2003. *Information Retrieval. Suchmodelle und Data-Mining-Verfahren für Textsammlungen und das Web*. Dpunkt Verlag.

[29] File:Facebook mobile.png - Wikipedia, the free encyclopedia: *http://en.wikipedia.org/wiki/File:Facebook_ mobile.png*. Accessed: 2011-09-04.

[30] Fleiss' kappa - a knol by Benoît Curdy: *http://knol.google.com/k/beno%C3%AEt-curdy/fleiss-kappa/32fm4g387q1y7/1#*. Accessed: 2011-11-14.

[31] Fogel, K. 2005. *Producing Open Source Software: How to Run a Successful Free Software Project*. O'Reilly Media.

[32] For, N.D. et al. 2004. The Enron Corpus: *IN ECML*. (2004), 217--226.

[33] Friedl, J. 1997. *Mastering Regular Expressions*. O'Reilly/VVA.

[34] Fuller, M. 2008. RSS Feed Complex Event Detection. Brown University.

[35] Gamma, E. et al. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.

[36] German, D. 2004. Mining CVS repositories, the softchange experience. *Proceedings*

of the First International Workshop on Mining Software Repositories (2004), 17-21.
[37] Giarlo, M. 2005. A Comparative Analysis of Keyword Extraction Techniques. (2005).
[38] Grouven, U. et al. 2007. Der Kappa-Koeffizient. *DMW - Deutsche Medizinische Wochenschrift*. 132, (2007), e65-e68.
[39] Gupta, T. et al. 2009. Characterization of FriendFeed - A Web-based Social Aggregation Service. *International AAAI Conference on Weblogs and Social Media* (Mar. 2009).
[40] Hammersley, B. 2003. *Content Syndication with RSS*. O'Reilly Associates.
[41] Hasan, M. et al. 1999. A Conceptual Framework for Network Management Event Correlation and Filtering Systems. *IN SLOMAN ET AL*. 22, (1999), 233--246.
[42] Hatcher, E. 2005. *Lucene in Action*. Manning.
[43] Hong, L. et al. 2010. FeedWinnower: layering structures over collections of information streams. *Proceedings of the 28th international conference on Human factors in computing systems* (Atlanta, Georgia, USA, 2010), 947–950.
[44] How many feeds do you subscribe to? - Alex Barnett's blog - Site Home - MSDN Blogs: *http://blogs.msdn.com/b/alexbarn/archive/2006/02/01/522004.aspx*. Accessed: 2011-11-24.
[45] Howison, J. et al. 2006. FLOSSmole: A collaborative repository for FLOSS research data and analyses. *International Journal of Information Technology and Web Engineering*. 1, (Jul. 2006), 17-26.
[46] Hölzer, R. et al. 2005. Email alias detection using social network analysis. *Proceedings of the 3rd international workshop on Link discovery - LinkKDD '05* (Chicago, Illinois, 2005), 52-57.
[47] Institut für Informationsarchitektur: Learning IA - Institut für Informationsarchitektur: *http://iainstitute.org/de/translations/000551.html*. Accessed: 2011-10-31.
[48] Jen, L. and T, G. 2010. Survey on Complex Event Processing and Predictive Analytics. *Networks*. (2010).
[49] Katz, P. 2011. Causal Relation Detection for Activities from Heterogeneous Sources. *ICWE 2011, 11th International Conference on Web Engineering* (Paphos, Cyprus, 2011).
[50] Katz, P. 2010. NewsSeecr - Clustering und Ranking von Nachrichten zu Named Entities aus Newsfeeds. Dresden University of Technology.
[51] Kleinberg, J.M. et al. 1999. The web as a graph: measurements, models, and methods. *Proceedings of the 5th annual international conference on Computing and combinatorics* (Berlin, Heidelberg, 1999), 1–17.
[52] Koch, M. and Richter, A. 2009. *Enterprise 2.0: Planung, Einführung und erfolgreicher Einsatz von Social Software in Unternehmen*. Oldenbourg Wissenschaftsverlag.
[53] Krishnan, V. and Ganapathy, V. 2005. Named Entity Recognition. (2005).
[54] Königer, P. and Janowitz, K. 1995. Drowning in information, but thirsty for knowledge. *International Journal of Information Management*. 15, 1 (Feb. 1995), 5-16.
[55] Landis, J.R. and Koch, G.G. 1977. The Measurement of Observer Agreement for

Categorical Data. *Biometrics*. 33, 1 (Mar. 1977), 159-174.
[56] Lewandowski, D. 2005. *Web Information Retrieval: Technologien zur Informationssuche im Internet*. Deutsche Gesellschaft f. Informationswissenschaft u. Informationspraxis.
[57] Lewis, D.E. et al. 1997. Threading Electronic Mail: A Preliminary Study. (1997).
[58] List of e-mail subject abbreviations - Wikipedia, the free encyclopedia: *http://en.wikipedia.org/wiki/E-mail_ subject_ abbreviations*. Accessed: 2011-09-06.
[59] Luckham, D.C. and Schulte, R. 2008. *Event Processing Glossary - Version 1.1*.
[60] Ludwig-Mayerhofer, W. Konfidenzintervalle so einfach wie möglich erklärt. Universität Siegen.
[61] Manning, C.D. et al. 2008. *Introduction to Information Retrieval*. Cambridge University Press.
[62] Mendes, M.R.N. et al. 2010. Benchmarking event processing systems: current state and future directions. *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering* (New York, NY, USA, 2010), 259–260.
[63] Messina, C. 2010. ActivityStreams.
[64] MySQL :: MySQL 5.1 Reference Manual :: 3.5 Using mysql in Batch Mode: *http://dev.mysql.com/doc/refman/5.1/en/batch-mode.html*. Accessed: 2011-11-08.
[65] NearRealtimeSearch - Lucene-java Wiki: *http://wiki.apache.org/lucene-java/NearRealtimeSearch*. Accessed: 2011-11-07.
[66] Novak, J. et al. 2004. Anti-aliasing on the web. *Proceedings of the 13th conference on World Wide Web - WWW '04* (New York, NY, USA, 2004), 30.
[67] One down side to an SQL EP approach « Hans Gilde's weblog: *http://hansgilde.wordpress.com/2007/11/10/one-down-side-to-an-sql-ep-approach/*. Accessed: 2011-10-22.
[68] POS Tagging (State of the art) - ACLWiki: *http://aclweb.org/aclwiki/index.php?title=POS_ Tagging_ %28State_ of_ the_ art%29*. Accessed: 2011-10-24.
[69] Pera, M.S. and Ng, Y.-kai Synthesizing Correlated RSS News Articles Based on a Fuzzy Equivalence Relation.
[70] RDF Site Summary 1.0 Modules: *http://web.resource.org/rss/1.0/modules/*. Accessed: 2011-09-04.
[71] RFC 3986, Section 6: Normalization and Comparison: *http://tools.ietf.org/html/rfc3986#page-38*. Accessed: 2011-09-06.
[72] Raymond, E.S. 2001. *The Cathedral and the Bazaar: Musings On Linux And Open Source By An Accidental Revolutionary*. O'Reilly Media.
[73] Rentsch, D. 2011. Konzept und Realisierung einer Architektur zur automatisierten und konsistenten Integration von Dokumentationsartefakten in webbasierten Hilfesystemen. Technische Universität Dresden.
[74] Resnick, P.W. 2008. *Internet Message Format*.
[75] Rss20AndAtom10Compared - Atom Wiki: *http://www.intertwingly.net/wiki/pie/Rss20AndAtom10Compared*. Accessed: 2011-07-11.
[76] Samper, J.J. et al. 2008. NectaRSS, an intelligent RSS feed reader. *J. Netw. Comput. Appl.* 31, 4 (Nov. 2008), 793–806.
[77] Sayyadi, H. et al. 2009. Event Detection and Tracking in Social Streams.

Proceedings of International Conference on Weblogs and Social Media (ICWSM) (2009).

[78] Second, W.N. 2009. Automatic Keyword Extraction for Database Search. *l3sde*. (2009), 17-20.

[79] Shen, D. et al. 2006. Thread detection in dynamic text message streams. *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2006), 35–42.

[80] Sittig, A. c/o F. and Zuckerberg, M. c/o F. 2008. SYSTEMS AND METHODS FOR GENERATING A SOCIAL TIMELINE. Sep. 24, 2008.

[81] Statista-Lexikon: Likert-Skala - Definition im Lexikon.: *http://de.statista.com/statistik/lexikon/definition/82/likert-skala/*. Accessed: 2011-09-21.

[82] TIOBE Software: Tiobe Index: *http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html*. Accessed: 2011-10-21.

[83] TWiki . Javawsxml . Rome05TutorialSampleModule: *http://wiki.java.net/twiki/bin/view/Javawsxml/Rome05TutorialSampleModule?TWIKISID=32258c62255945c685c9360117899e75*. Accessed: 2011-11-07.

[84] The Penn Treebank Tag Set: *http://www.ims.uni stuttgart.de/projekte/CorpusWorkbench/CQP-HTMLDemo/PennTreebankTS.html*. Accessed: 2011-10-24.

[85] The Stanford NLP (Natural Language Processing) Group: *http://nlp.stanford.edu/software/pos-tagger-faq.shtml*. Accessed: 2011-11-09.

[86] Urbansky, D. et al. WebKnox: Web Knowledge Extraction.

[87] Vayghan, J.A. et al. 2007. The internal information transformation of IBM. *IBM Systems Journal*. 46, 4 (2007), 669-683.

[88] Weltkarte der Sozialen Netzwerke 2011 » Von markus » netzpolitik.org: *http://netzpolitik.org/2011/weltkarte-der-sozialen-netzwerke-2011/*. Accessed: 2011-09-04.

[89] Whatever....: Benchmarking results of mysql, lucene and sphinx...: *http://jayant7k.blogspot.com/2006/06/benchmarking-results-of-mysql-lucene.html*. Accessed: 2011-11-07.

[90] Wilson, C. et al. 2009. User interactions in social networks and their implications. *Proceedings of the 4th ACM European conference on Computer systems* (New York, NY, USA, 2009), 205–218.

[91] Wunderwald, M. 2011. NewsX: Eventextraktion aus Nachrichtenbetirägen. Dresden University of Technology.

[92] Yi, L. 2003. Eliminating Noisy Information in Web Pages for Data Mining. *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. (2003), 296--305.

[93] Zhao, Q. et al. 2007. Temporal and information flow based event detection from social text streams. *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2* (2007), 1501–1506.

[94] url rewriting - How to normalize a URL in Java? - Stack Overflow: *http://stackoverflow.com/questions/2993649/how-to-normalize-a-url-in-java/4057470#4057470*. Accessed: 2011-11-03.