Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Steffen Brauer

Konzeption und Realisierung einer Cloud-Anwendung zur Kapselung sozialer Netzwerke

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Steffen Brauer

Konzeption und Realisierung einer Cloud-Anwendung zur Kapselung sozialer Netzwerke

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Olaf Zukunft
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 25. August 2011

Steffen Brauer

Thema der Arbeit
Konzeption und Realisierung einer Cloud-Anwendung zur Kapselung sozialer Netzwerke

Stichworte
Cloud Computing, REST, soziale Netzwerke

Kurzzusammenfassung
Cloud Computing stellt den nächsten Schritt in der Entwicklung der Bereitstellung von IT-Ressourcen dar, doch auch soziale Netzwerke gewinnen in der Gesellschaft immer mehr an Bedeutung. Mit ihrer Popularität wächst auch die Anzahl und es fällt schwer den Überblick zu behalten. In dieser Arbeit soll eine Anwendung zur Kapselung sozialer Netzwerke entwickelt werden und verschiedene Varianten zur Portierung in die Cloud vorgestellt werden.

Steffen Brauer

Title of the paper
Design and implementation of a cloud application to encapsulate social networks

Keywords
Cloud Computing, REST, social Networks

Abstract
Cloud computing is the next step in the development of how to deploy it resources. Also social networks are becoming increasingly important in society. Their popularity grows as fast as their number, so it is hard to keep track. This thesis covers a development of an application to encapsulate social networks and introduces several ways to port the application into the cloud.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Zu Beginn dieser Bachelorarbeit soll die Wahl des Themas begründet und die Aufgabenstellung präzisiert werden.

1.1 Motivation

Soziale Netzwerke haben einen festen Platz im Leben zahlreicher Menschen. Dort senden sie ihren Freunden Nachrichten oder teilen das täglich Erlebte mit ihnen. Doch fällt häufig die Wahl des richtigen Netzwerkes schwer, da diese sich durch ihre Funktionalität, Anzahl der Nutzer sowie soziale Ausprägung unterscheiden. So sind laut Hargittai und li Patrick Hsieh (2010) nicht nur 52 Prozent aller Nutzer von sozialen Netzwerken mehrmals täglich aktiv, sondern sind auch in mehreren Netzwerken angemeldet. Dies könnte zum Verlust des Überblicks oder dem Verbreiten von Nachrichten in mehreren Netzwerken führen. Hierfür bieten zwar schon diverse soziale Netzwerke die Möglichkeit andere Netzwerke zu integrieren, jedoch fehlt eine unabhängige Plattform zur Interaktion mit mehreren Anbietern. Eine solche Plattform hätte jedoch nicht nur Vorteile für private Nutzer, auch Unternehmen können von sozialen Netzwerken nicht nur durch das Schalten von Werbung profitieren: Durch die Präsenz in sozialen Netzwerken können Unternehmen neue Marketingstrategen nutzen oder Meinungen zu den Produkten aus den sozialen Netzwerken beziehen. Die Entwicklung der Plattform als Cloud-Anwendung hat vielfältige Vorteile: Cloud-Computing ist zur Zeit einer der meist gehypten Trends in der Informatik, da es Computerressourcen, wie Speicher, Rechenleistung oder ganze Software über das Internet als Service verfügbar macht. Durch das Abrechnen von nur tatsächlich benötigten Ressourcen bieten sich erhebliche Vorteile für Neuentwicklungen, da es nicht nötig ist kostenintensive Server anzuschaffen oder bei stärkerer Nachfrage neue zu integrieren. Aber auch für die Betreiber von Rechenzentren bietet Cloud-Computing neue Möglichkeiten: Sie können nicht benötigte Ressourcen anderen anbieten und so auch mit von ihnen nicht genutzten Ressourcen Gewinn erzielen.

1.2 Aufgabenstellung

In dieser Bachelorarbeit soll eine Cloud Anwendung zur Kapselung sozialer Netzwerke entwickelt werden. Hierzu sollen zunächst verfügbare Schnittstellen von populären Netzwerken analysiert und auf Grundlage dieser Erkenntnisse eine gemeinsame Abstraktionsebene gefunden werden. Über diese Ebene soll es möglich sein, einzelne Objekte aus einem speziellen Netzwerk abzurufen. Auch wird es möglich sein, neuste Aktivitäten aus allen verfügbaren Netzwerken anzuzeigen und über eine Schnittstelle eine Nachricht in mehrere Netzwerke zu verbreiten. Als eine weitere Funktion sollen die Freundschaftsbeziehungen von Benutzer abgerufen werden können. Zur Authentifizierung sollen die von den Netzwerken angebotenen Möglichkeiten genutzt werden. Im Anschluss sollen verschiedene Varianten zur Portierung der Anwendung in die Cloud vorgestellt werden.

2 Grundlagen

2.1 Cloud Computing

In diesem Kapitel sollen die Grundlagen des Cloud Computing vermittelt werden. Hierzu wird zuerst in die Thematik der Cloud und ihrer Entwicklung eingeführt. Der nächste Teil des Kapitels beschäftigt sich mit den charakteristischen Eigenschaften der Cloud und definiert diese. Ebenso soll die Trennung zum Grid Computing vorgenommen und die verschiedenen Ebenen in der Cloud-Architektur erklärt werden. Am Ende werden die wirtschaftlichen und sicherheitsbezogenen Aspekte der Cloud betrachtet sowie eine Bewertung zum heutigen Stand des Cloud Computing abgegeben.

2.1.1 Einführung in die Thematik

In Carr (2009) vergleicht der Autor die Entstehung des Stromnetzes und der damit einhergehenden Zentralisierung der Stromgenerierung mit der aktuellen Entwicklung in der Informatik: Dem Trend des Cloud Computing. Auch hier bewegt sich die Rechenleistung von vielen dezentralen Standorten, den PCs, hin zu großen Rechenzentren mit mehreren tausend Servern. Da Cloud Computing erst am Anfang seiner Entwicklung steht, ist noch nicht bekannt, ob es ebenfalls dieselben Folgen auf die Wirtschaft und Gesellschaft haben wird, wie die Elektrifizierung. Die Entwicklung zur Cloud begann bereits in den 1970er Jahren. Nachdem in den 1960er Jahren die ersten kommerziellen Mainframes auf den Markt kamen, die jeweils nur von einem Benutzer zur gleichen Zeit verwendet werden konnten, wurden so genannte time-shared Systeme entwickelt. Hierbei wurden die Grundsteine zur Virtualisierung von Hardware gelegt, da nun jeder Benutzer des Computers eine virtuelle Maschine zur Verfügung gestellt bekam, sodass es für ihn den Anschein hatte, er sei der einzige Nutzer. Da in den folgenden Jahrzehnten die Computer immer schneller und kleiner wurden, konzentrierte man sich weniger auf zentrale als vielmehr auf verteilte Computersysteme. Dieser Trend gipfelte in den sogenannten thick-client Anwendungen, wie zum Bespiel Microsoft Office. In den 1980er Jahren wurde mit dem TCP/IP Standard der Grundstein für die einfache Vernetzung von einer großen Anzahl an Computern, dem Internet, gelegt. Mit der Entwicklung des Web und somit

auch des HTTP Protokolls in den 1990er Jahren begann das Zeitalter des Cloud Computing. Der Begriff der Cloud, also der Wolke, hat seinen Ursprung in Diagrammen, welche Netzwerktopologien beschreiben. Hier soll die Wolke darstellen, dass nicht bekannt ist, wo sich eine Ressource oder ein Nutzer befindet.

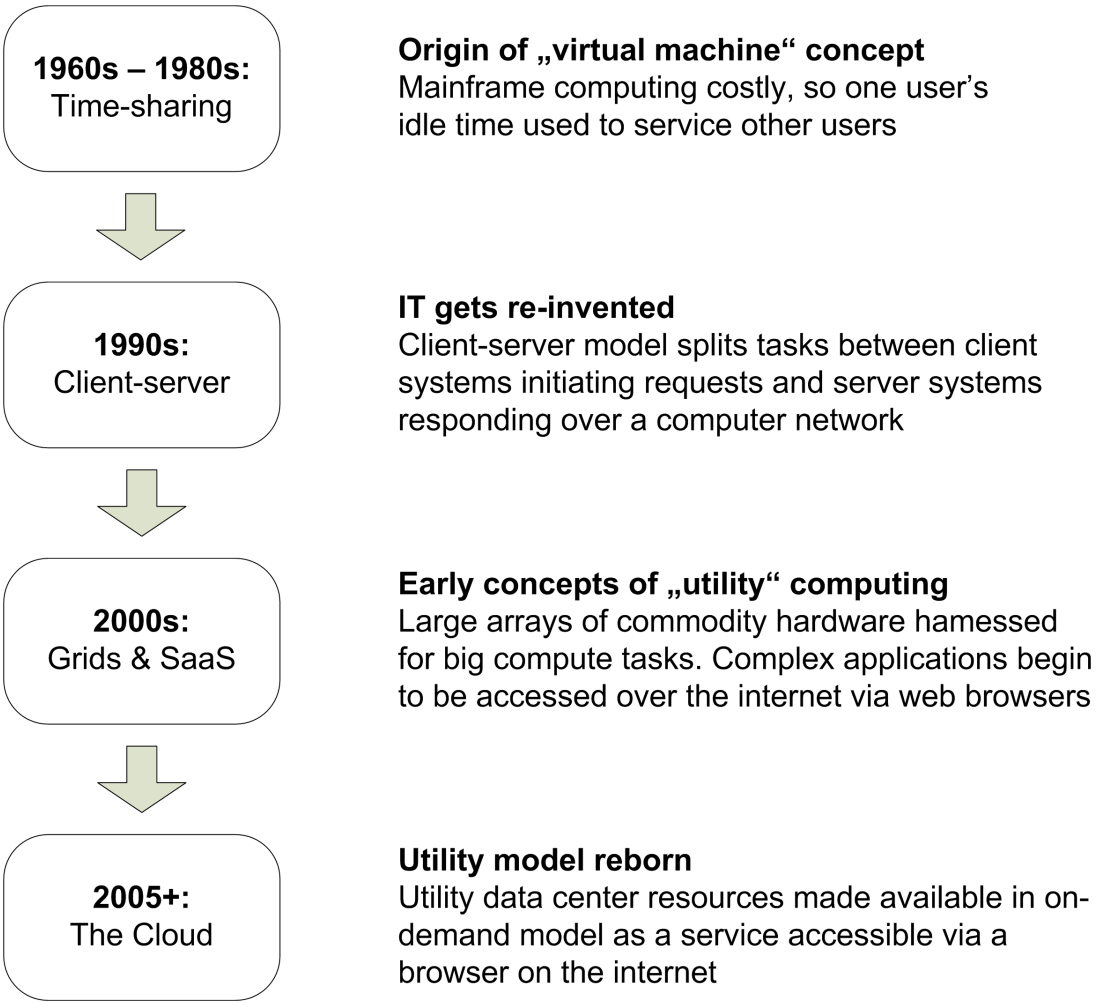

Abbildung 2.1: Entwicklung zum Cloud Computing nach Rosenberg und Matoes (2011)

2.1.2 Definition

Nachdem die Herkunft des Begriffes der Cloud erläutert wurde, stellt sich die Frage, worum es sich bei Cloud Computing konkret handelt. Es ermöglicht die Nutzung von IT- Ressourcen als Dienste über das Internet. Hierbei fallen unter Ressourcen die Infrastruktur, Plattformen und

Anwendungen. Diese Dienste werden von Providern angeboten, die entweder auf weiteren Cloud-Diensten aufbauen oder auf klassischen Servern zur Verfügung gestellt werden. Die in Rosenberg und Matoes (2011) genannten Eigenschaften des Cloud Computing werden im Folgenden erläutert:

- Die Metapher der Wolke suggeriert das Vorhandensein von unendlichen Ressourcen. Diese werden in großen Rechenzentren betrieben und Kunden zur Verfügung gestellt. Erst Rechenzentren im heutigen Umfang und an günstigen Standorten ermöglichen das Cloud Computing. Doch vermitteln tausende von physikalischen Servern noch nicht das Bild einer Wolke, da man ihren konkreten Standort und ihre Anzahl kennt.

- Erst durch die Virtualisierung dieser Server verschwimmt die Sicht auf die physikalische Infrastruktur, da diese auf mehrere virtuelle Server aufgeteilt werden. Die sogenannten Serverinstanzen sind die primären Grundeinheiten in der Cloud. Auf der Granularität dieser Einheiten kann jeder Cloud-Nutzer die, für seine Anwendung benötigte Infrastruktur individuell zusammenstellen und betreiben.

- Diese große Anzahl virtualisierter Server ermöglicht das Konzept der Elastizität. Dieses Konzept beschreibt die dynamische Skalierbarkeit einer Anwendung. Um dies zu ermöglichen werden bei hohem Bedarf an Ressourcen, diese automatisch hinzu geschaltet und bei fallender Nutzung der Anwendung wieder freigegeben. Diese wichtige Eigenschaft der Cloud macht es demnach überflüssig, jederzeit so viele Ressourcen vorzuhalten, wie diese für die maximale Auslastung benötigt würden.

- Um diese Anpassung an Laständerungen vorzunehmen, bieten die Provider virtueller Server spezielle APIs an, mit denen ermöglicht wird neue Instanzen anzulegen, zu starten und wenn sie nicht mehr benötigt werden, wieder abzuschalten. Desweiteren werden auch nur die benötigten Ressourcen in Form von Prozessor per Stunde oder Giga Byte per Tag berechnet.

- Dieses Vorgehen der Bereitstellung und Abrechnung folgt der Idee des Utility Computing und trägt so zu der wirtschaftlichen Bedeutung des Cloud Computing bei. Auf diese Weise fallen gegenüber dem klassischen Hosting Modell keine Anschaffungskosten für die Infrastruktur an und es müssen nur Kosten für den Betrieb gezahlt werden, wenn diese auch genutzt wird.

Es gibt nur wenige Autoren, die eine kurze, zusammenfassende Definition für Cloud Computing abgeben, daher soll an dieser Stelle eine von Baun u. a. (2009) genügen:

> „Unter Ausnutzung virtualisierter Rechen- und Speicherressourcen und moderner Web-Technologien stellt Cloud Computing skalierbare, netzwerk-zentrierte, abstrahierte IT- Infrastrukturen, Plattformen und Anwendungen als on-demand Dienste zur Verfügung. Die Abrechnung dieser Dienste erfolgt nutzungsabhängig."

2.1.3 Abgrenzung zum Grid Computing

Viele der beschriebenen Eigenschaften ähneln denen des Grid Computing, doch gibt es klare Unterschiede, die in Wang (2008) aufgezeigt werden. Hierzu werden verschiedene Aspekte beider Modelle verglichen: die Definition, Infrastruktur, Middleware und Anwendungen. Wobei an dieser Stelle der Middleware-Aspekt ausgelassen wird, da sich seit der Veröffentlichung der Stand der Forschung stark verändert hat.

Grid Computing definiert sich durch hochperformantes verteiltes Rechnen und stellt hierfür verteilte Ressourcen zur entfernten Ausführung von Programmen und für hoch skalierbare Problemlösungen zur Verfügung. Im Gegensatz dazu stellt Cloud Computing benutzerzentrierte Funktionalität bereit, um eine individuelle IT- Umgebung einzurichten und zu betreiben. Die Infrastruktur beim Grid Computing ist außerdem dezentral organisiert, wobei geographische Grenzen überschritten werden können. Aufgrund der dezentralen Organisation liegt die Kontrolle nicht in einer Hand. Desweiteren enthält eine Grid Infrastruktur heterogene Ressourcen, die sich bezüglich ihrer Konfiguration, Schnittstellen und Administration unterscheiden können. Die Cloud Computing Infrastruktur wird hingegen in Rechenzentren der jeweiligen Anbieter betrieben, wodurch homogene Ressourcen zur Verfügung stehen, die einer zentralen Kontrolle unterliegen.

Anders als beim Cloud Computing, muss die jeweilige Anwendung an die Eigenschaften des Grid angepasst werden müssen. Auch ist es schwierig eine klare Garantie für den verfügbaren Service abzugeben. Cloud Computing stellt hingegen klare Quality of Service Garantien aus und bietet einfachen Zugang zu skalierbaren und benutzerdefinierten Rechenressourcen.

Zusammenfassend lässt sich sagen, das Cloud Computing eine abstraktere und benutzerfreundliche Nutzung von hochperformanten Rechenressourcen bietet. Zusammen mit dem nutzungsbezogenen Abrechnungsmodell ist Cloud Computing vor allem für Unternehmen interessant. Grid Computing hingegen bietet die Möglichkeit hoch skalierbare Probleme über ein Netzwerk von heterogene und stark verteilte Ressourcen zu lösen, welche in speziellen Unternehmen aber auch häufig in der Wissenschaft anzutreffen sind.

2.1.4 Kategorisierung

Die Kategorisierung der Cloud kann von verschiedenen Standpunkten aus vorgenommen werden. Da in dieser Arbeit nur technische IT-Services behandelt werden, wird die von Baun u. a. (2009) angeführt Schicht Human as a Service (HuaaS) an dieser Stelle nicht genauer erläutert.

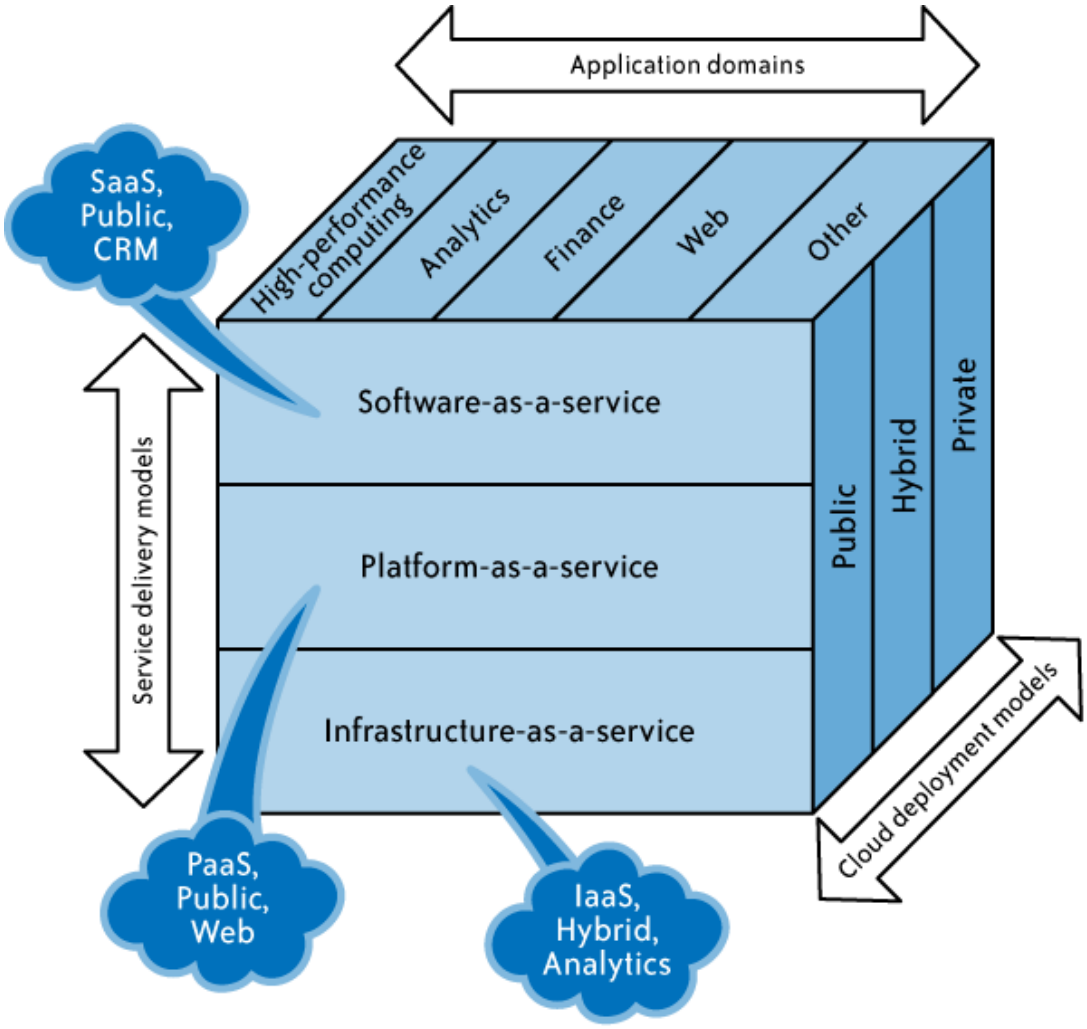

Abbildung 2.2: Gesamtbild der Cloud Mather u. a. (2009)

Organisatorische Kategorisierung

Wie Abbildung 2.2 zeigt, kann eine Kategorisierung der Cloud nach zwei unterschiedlichen Gesichtspunkten vorgenommen werden:

- organisatorisch oder
- technisch.

Im Folgenden soll zunächst die organisatorische Kategorisierung vorgenommen und anschließend die Aspekte der technischen Trennung beleuchtet werden.

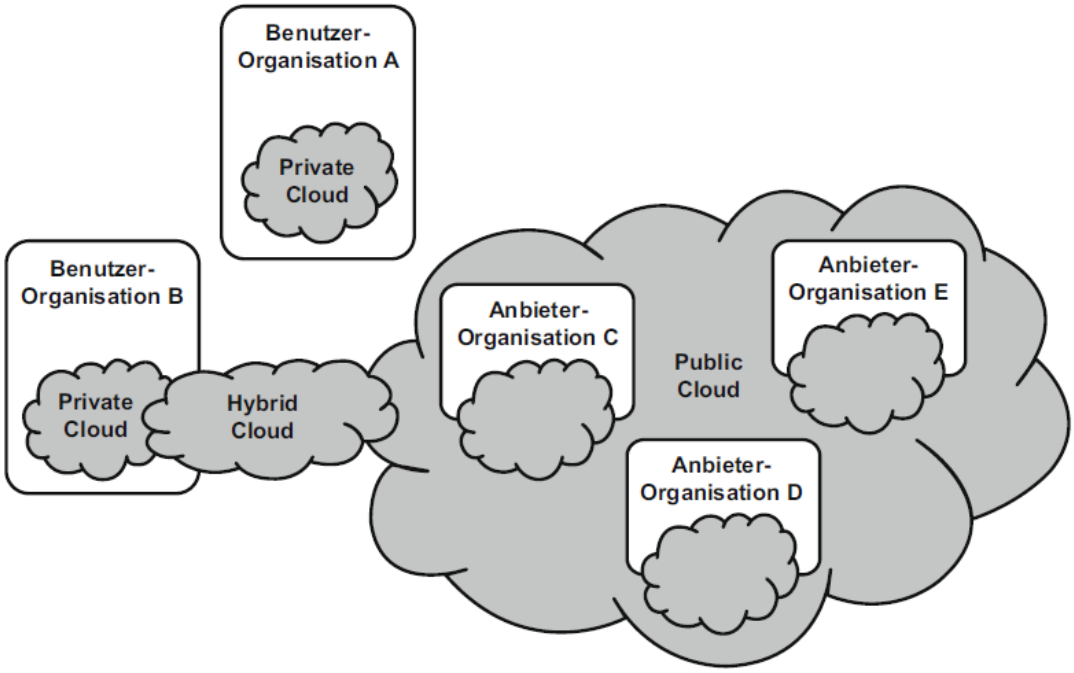

Abbildung 2.3: Public, Private and Hybird Cloud Baun u. a. (2009)

Public Cloud Aus organisatorischer Sicht lässt sich die Wolke nach ihrer Verfügbarkeit charakterisieren. Man spricht von der Public Cloud, wenn ein Anbieter seine Cloud Ressourcen veröffentlicht. Zur Verwaltung dieser Ressourcen wird dem Benutzer meistens ein Web-Portal gestellt, mit dessen Hilfe er seine individuelle Konfiguration verwalten kann. Diese Art der Cloud entspricht der allgemeinen Definition.

Private Cloud Es ist aber auch möglich, dass Unternehmen ihre Rechenzentren zu einer sogenannten Private Cloud umgestalten. So bleibt die Kontrolle über den Zugriff auf eventuelle schützenswerte Daten in der Hand des Unternehmens. Allerdings können so nur die im Unternehmen bereitstehenden Ressourcen genutzt werden und die Illusion der unendlichen Ressourcen ist nicht mehr vorhanden.

Hybrid Cloud Um diesem Effekt entgegen zu wirken, können im normalen Betrieb alle Aufgaben in der privaten Cloud des Unternehmens durchgeführt werden. Bei steigender Last ist es aber möglich, einige Aufgaben in die öffentliche Wolke zu verschieben. Ein solches Vorgehen wird als Hybrid Cloud bezeichnet. Die Zusammenhänge werden in Abbildung 2.3 noch einmal visualisiert.

Technische Kategorisierung

Aus technischer Sicht lässt sich die Cloud nach Abstraktionsebene der verschiedenen Angebote aufteilen. Es wird zwischen virtuellen Servern, Plattformen für verschiedene Programmiersprachen und Frameworks sowie dem Angebot von Anwendungen unterschieden. Die Abstraktionsebenen sind in Schichten organisiert, wobei ein Angebot einer oberen Schicht nicht zwangsläufig auf Angeboten einer der Unteren aufbauen muss. Dieser Sichtweise folgen Baun u. a. (2009) und Rosenberg und Matoes (2011). Doch nimmt Armbrust u. a. (2010) diese Unterteilung nicht vor, da eine genaue Trennung der in dem Modell unten angesiedelten Schichten nicht einfach möglich sei.
Die Benennung folgt hierbei dem Everything as a Service Paradigma (XaaS). Im Folgenden werden die drei Hauptvertreter dieses Paradigma im Cloud Computing genauer erläutert.

Infrastructure as a Service Auf der untersten Ebene der Cloud Architektur befindet sich IaaS. Diese Services ermöglichen dem Benutzer eine abstrahierte Sicht auf die angebotene Infrastruktur. Zur Verwaltung dieser abstrakten Hardware stehen Benutzerschnittstellen zur Verfügung, die es ermöglichen, benötigte Ressourcen zu allokieren und virtuelle Server zu starten. Hierzu kann der Benutzer aus einer Menge von bereits vorkonfigurierten Betriebssystemabbildern, zum Beispiel für den Betrieb einen Webservers, auswählen oder ein eigenes Abbild erstellen. Nun können von diesen Abbildern, den sogenannten Images, Instanzen erstellt und betrieben werden. Des weiteren ist es über die Benutzerschnittstelle möglich, die allokierten Ressourcen höher oder niedriger zu skalieren oder eine individuelle Netzwerktopologie zu definieren. Die angebotenen Ressourcen werden hierbei von Baun u. a. (2009) in zwei unterschiedliche Kategorien eingeteilt:

Virtual Resource Set Hierbei wird von der physikalischen Hardware komplett abstrahiert und es werden vollkommen virtualisierte Ressourcen zur Verfügung gestellt. Dies entspricht der Definition von Cloud Computing und ist somit am häufigsten anzutreffen.

Physical Resource Set Bei dieser Art der Bereitstellung wird zu Gunsten von Stabilität, Performanz oder spezieller Hardware-Anforderung auf die Virtualisierungsebene verzichtet. Es steht aber eine komfortable Schnittstelle bereit.

Zusätzlich zu Ressourcen können der IaaS Schicht noch sogenannte Infrastructure Services zugeordnet werden. Bei diesen Diensten ist der Anwendungsfokus jedoch beschränkt, so gibt es Angebote für Massenspeicher, Netzwerke oder verteilte Berechnungsaufgaben.
Laut Rosenberg und Matoes (2011) bietet IaaS die größte Flexibilität und Kontrolle über die verwendeten Ressourcen, doch geht mit dieser Freiheit auch mehr Aufwand für den Entwickler zur Konfiguration und Wartung einher.

Plattform as a Service In der nächst höheren Schicht der Cloud Architektur werden Entwicklern Plattformen bereitgestellt auf denen es möglich ist, Applikationen sehr schnell und einfach im Internet bereitzustellen. Hierbei stellt der Anbieter eine Laufzeitumgebung für eine konkrete Programmiersprache bereit, so ist weniger Administration und somit Interaktion mit der Plattform nötig. Diese festen Vorgaben der Anbieter schmälern die Flexibilität hinsichtlich der Konfiguration der Plattform und erschweren somit einen möglichen Anbieterwechsel.

Software as a Service Auf oberster Ebene der Cloud Architektur befindet sich Software as a Service. Angebote dieser Schicht richten sich an Endkunden und stellen entweder komplexe Anwendungen oder einzelne Anwendungsdienste bereit. Hierbei ist entscheidend, dass auf Seite des Benutzers keine Installation notwendig ist oder Ressourcen verfügbar sein müssen. Diese Applikation stehen über eine Browserschnittstelle zur Verfügung oder können via Web Service konsumiert werden.
Rosenberg und Matoes (2011) nennt als Unterkategorie von SaaS Framework as a Service, welches es ermöglichen soll, bestehende SaaS Angebote zu erweitern und so kundenspezifische Anpassungen vornehmen zu können.

2.1.5 Bewertung

Nachdem auf die Grundlagen des Cloud Computing eingegangen wurde, soll eine wirtschaftliche und sicherheitsbezogene Betrachtung dargestellt werden und der Komplex in einem Fazit zusammengefasst werden.

Wirtschaftliche Betrachtung

Eines der naheliegenden Einsatzgebiete des Cloud Computing sind Web-Anwendungen. Als ein prominentes Beispiel hierfür nennt Baun u. a. (2009) den Videodienst Animoto (2011), bei dem es möglich ist, aus hochgeladenen Musik- und Bilddateien ein Video mit einer Bilderschau zu generieren. Als sich dieser Dienst mit dem sozialen Netzwerk Facebook (2011c) vernetzte, stieg die Anzahl der registrierten Nutzer um stündlich bis zu 25.000 auf insgesamt bis zu 250.000. Nun war das 100-fache der vorher genutzten IT-Infrastruktur nötig um der Nachfrage stand zuhalten. Dies machte die Nutzung eines öffentlichen Cloud-Dienstes notwendig, so dass auf ein späteres Nachlassen der Nachfrage ebenfalls ohne Probleme reagiert werden konnte.
In dem Beispiel Animoto (2011) wird die wirtschaftliche Attraktivität des Cloud Computing für Start-Ups deutlich, da diese kein Startkapital benötigen und bei Erfolg schnell auf die steigende Benutzeranzahl reagieren können.
Durch die dynamische Anpassung der Ressourcen und der nutzungsabhängigen Abrechnung können die Fixkosten für IT reduziert und in operative Kosten umgewandelt werden. Dies bietet einen klaren Kostenvorteil gegenüber der Anschaffung der IT oder dem Leasing-Modell, wobei die Ressourcen entweder gekauft oder auf feste Zeit gemietet werden. Bei diesen Modell findet die Nutzung keine Berücksichtigung bei der Abrechnung.
Des weiteren ist es für Unternehmen nicht nötig Ressourcen für eine seltene Spitzenlast bereit zu halten, dem so genannten Overprovisioning. Es wird aber auch kein durch nicht ausreichende Ressourcen hervorgerufener Ausfall des Diensten riskiert (underutilization). Armbrust u. a. (2010) nennt die „cost associativity“ als weiteren Vorteil des Cloud Computing für Unternehmen mit großen, batch-orientierten Aufgaben. Hiermit ist gemeint, dass es keinen preislichen Unterschied macht, ob ein Server eintausend Stunden betrieben wird oder eintausend Server eine Stunde lang. Unter der Voraussetzung, dass diese Batchprozesse skalierbar sind, können so Unternehmen ihre Ergebnisse in einem Bruchteil der bisher benötigten Zeit erhalten.
Außerdem bietet das Cloud Computing auch Vorteile für Betreiber von Rechenzentren. Da laut Armbrust u. a. (2010) die durchschnittliche Auslastung eines Rechenzentrums 5 bis 20 Prozent beträgt, können durch Anbieten der nicht genutzten Ressourcen als IaaS Angebot auch nicht genutzt Ressourcen Gewinn erzielen.

Sicherheit

Als Hauptgrund IT-Ressourcen nicht in die Cloud zu verlagern, nennen laut einer von Rosenberg und Matoes (2011) erwähnten IDC Studie von 2009 72,6 Prozent der befragten IT-Vorstände Bedenken bei der Sicherheit. Diesem Trend stimmen sowohl Sumter (2010) als auch

Mather u. a. (2009) zu.
Doch sind die für Cloud Computing relevanten Sicherheitsthemen identisch mit denen großer Rechenzentren und somit gibt es auch Lösungsansätze zur Bewältigung dieser Aufgaben. Armbrust u. a. (2010) veranschaulicht die Grenze der Verantwortung zwischen Cloud Benutzer und Anbieter. Bei IaaS Anbietern liegt die Anwendungssicherheit voll und ganz beim Benutzer und der Anbieter ist für die physikalische Sicherheit der Daten verantwortlich. Diese Verantwortlichkeit verschiebt sich jedoch bei höheren Schichten zugunsten des Cloud Benutzers, legt aber auch immer mehr Verantwortung in die Hände des Cloud Betreibers.
Das Augenmerk des Anbieters liegt nicht nur auf dem Schutz vor Hackerangriffen von außen, er muss auch die Benutzer untereinander schützen, vor allem deshalb, da Bugs in der Virtualisierungsoftware schwere Sicherheitslücken zur Folge haben können. Deshalb sollte die Standardabwehrtechnik in diesem Fall die vom Benutzer durchgeführte Verschlüsselung sein. Rosenberg und Matoes (2011) argumentiert allerdings, dass Cloud Computing sicherer ist, als das Betreiben der Ressourcen im eigenen Haus, da die Betreiber von großen Rechenzentren Erfahrung und eine gute technische Ausstattung haben, um die Daten vor Angreifern zu schützen. Dies bezieht sich nicht nur auf die physikalische Datensicherheit, durch RAID-Systeme und Wachpersonal, es werden sondern auch Public-Key Verfahren zur Authentifizierung und Verschlüsselung eingesetzt. Aus diesen Argumenten ziehen Rosenberg und Matoes (2011) und Baun u. a. (2009) den Schluss, das verschlüsselte Daten in der Cloud sicherer sind, als unverschlüsselte auf lokalen Servern. Einen juristischen Standpunkt hinsichtlich der Sicherheit und Datenschutz beim Cloud Computing stellt Winter (2010) da. Das größte Problem sei, dass der Ort der Speicherung und Verarbeitung von Daten nicht bekannt ist. Dies führe zu besonderen Herausforderungen hinsichtlich der Einhaltung von Gesetzen und branchenspezifischen Richtlinien. So erlaube das Datenschutzrecht keine Übermittlung von personenbezogenen Daten in nicht EU Staaten nur in sehr eingeschränkten Fällen und legt dem Betreiber bei der Übermittlung innerhalb der europäischen Union besondere Pflichten bei der Verarbeitung auf. Auch sei die Verschlüsslung der Daten keine optimale Lösung, da diese sehr rechenaufwendig sei und die Daten zur Verarbeitung entschlüsselt werden müssen. Einen weiteren von Winter (2010) Punkt betrifft die Gerichtsverwärtbarkeit von Daten, die in der Cloud gespeichert werden. Es ist beim Cloud Computing nicht möglich, physikalische Datenträger zu beschlagnahmen und Abbilder von Cloud Datenspeichern hätten nur einen verminderten Beweiswert. Auch seien bei Verträgen die Interessen der Cloud Nutzer gegenüber denen der Cloud Anbieter nicht genügend vertreten. So herrsche „eine Diskrepanz zur technischen Durchsetzung (z.B. technische Unmöglichkeit der Datenlösung bei Vertragsende oder besonderen Ereignissen wie Insolvenz)“ Winter (2010).

Fazit

Cloud Computing bricht mit der zur Zeit vorherrschenden Art IT-Ressourcen und Software zur Verfügung zu stellen und bietet damit viele Vorteile für IT-Unternehmen. So können nicht nur Kosten im Betrieb, sondern auch schon bei der Entwicklung von Software eingespart werden, da Ressourcen für einen Prototyp auf kurze Zeit allokiert werden können. Auch kann mithilfe des Cloud Computing praktisch jeder an den hochtechnisierten und sicheren Standards in großen Rechenzentren teilhaben und so auch mit wenig Budget sichere und verfügbare Dienste anbieten. Doch gibt es noch Risiken, die gelöst werden müssen. So nennt Armbrust u. a. (2010) 10 Risiken und damit verbundene Chancen, die von Baun u. a. (2009) ins Deutsche übersetzt wurden(Abbildung 2.4). Trotz dieser Risiken hat sich Cloud Computing fest im Markt integriert,

	Risiken	Chancen
1	Verfügbarkeit der Dienste	Nutzung mehrerer Cloud-Anbieter zur Sicherstellung der Dienste-Kontinuität
2	Lock-In der Daten	Standardisierung der Schnittstellen (API)
3	Vertraulichkeit und Nachvollziehbarkeit der Daten	Einsatz von Datenverschlüsselung, Virtuellen Netzwerken (VLAN) und Firewalls. Einhaltung nationaler Gesetze durch geographische Datenhaltung
4	Engpässe bzgl. Datentransfer	Versand von Festplatten durch Dritte (z. B. FedEx, UPS)[a]
5	Schlechte Vorhersagbarkeit der Leistungsfähigkeit (Performance)	Bessere Unterstützung virtueller Maschinen. Einsatz von Flash-Speicher
6	Skalierbarer persistenter Speicherplatz	Entwicklung skalierbarer Speicherplatz-Technologien
7	Fehler in großen, verteilten Systemen	Entwicklung von Debuggern für verteilte Maschinen
8	Schnelles Skalieren	Entwicklung automatischer Skalierungswerkzeuge, basierend auf Machine Learning. Ressourcen- und kostenbewusstes Nutzer- und Anbieterverhalten
9	Reputation und Haftpflicht	Einsatz von Dienstleistungen Dritter, wie z. B. für vertrauenswürdige Emails
10	Software Lizenzen	Nutzungsbezogene Lizenzen (Pay-for-use). Verkauf von Software und Diensten im Paket

Abbildung 2.4: Chancen und Risiken des Cloud Computing in Baun u. a. (2009)

was sich auch im Top Thema der CeBIT in diesem Jahr wiederspiegelt: „Work and Life with the Cloud“Cebit (2011). Doch ob das Cloud Computing, wie in Carr (2009) beschrieben, dieselben Folgen wie die Elektrifizierung haben wird, bleibt abzuwarten.

2.2 Representational State Transfer

Der Representational State Transfer Architekturstil eignet sich durch seine Eigenschaften sehr gut zur Realisierung von Cloud-Anwendungen. Auch bieten soziale Netzwerke meist ihre Daten

durch eine REST API an. Daher soll REST im folgenden auf Grundlage der Dissertation von Roy Thomas Fielding(Fielding (2000)) und entsprechenden Interpretationen erörtert werden.

2.2.1 Definition

REST dient als Architekturstil für verteile hypermedia Systeme. Die genaue Implementation von Komponenten, sowie die Protokolle zur Kommunikation sind hierbei nicht zu beachten. Hingegen wird Augenmerk auf die Rolle von Komponenten, ihrer Datenverarbeitung und die Bedingungen in der Kommunikation zwischen ihnen gelegt. Um einen ersten Eindruck von REST zu erhalten, werden im folgenden die Bedingungen der Kommunikation genauer erläutert:

client/server Diese Bedingung bedient das Prinzip Seperation of Concerns. Bei der Client/Server Architektur trifft dies zu, da hier die Benutzerschnittstelle vom Server gelöst und nur auf den Clients ausgeführt wird. Auf diese Weise wird die Portierbarkeit der Plattform gefördert und die Skalierung der Serverkomponenten vereinfacht.

stateless Zustandlosigkeit bedeutet, dass der Server in der Lage ist, jeden Request anhand der enthaltenen Informationen abzuarbeiten. Da so der Zustand der Anwendung im Client gehalten werden muss, hat der Server weniger Kontrolle über das Verhalten. Durch das mehr an Informationen, das zur Bearbeitung des Request benötigt wird, sinkt auch die Netzwerkperformanz. Allerdings weist die Bedingung der Zustandlosigkeit starke Vorteile auf: Das Loggen des Verkehrs wird einfacher, da keine Informationen über den Zustand zwischen Requests gehalten werden müssen. Des weiteren wird die Sicherheit gegenüber Ausfällen bei kleineren Fehlern gestärkt und das Skalieren wird vereinfacht, da kein Zustand zwischen Requests gespeichert werden muss und so Ressourcen schnell wieder freigegeben werden können.

cache Durch Caching ist es möglich, die Netzwerkeffizienz zu stärken, da der Client ein früheres Ergebnis eines Request wiederverwenden kann. Ein Einsatz ist nur unter der Bedingung möglich, dass jeder Response implizit oder explizit als cacheable oder non-cacheable markiert wird. Mit diesem Verfahren sinkt zwar die Ausfallsicherheit, andererseits können so einige Anfragen eingespart werden, was zu einer Verbesserung der Effizienz, Skalierbarkeit sowie einer geringen Latenz bei einer Vielzahl von Zugriffen führt.

uniform interface Eine systemweite uniforme Schnittstelle vereinfacht die Architektur und erhört die Sichtbarkeit von Interaktionen zwischen den Komponenten. Desweiteren werden

konkrete Implementierungen von den angebotenen Services entkoppelt. Ein Nachteil dieser Schnittstellen besteht in der Umwandlung von internen Repräsentationen zu standardisierten.

layered system Mit der Aufteilung von Systemen in mehrere Schichten kann die Komplexität verringert werden, da jede Schicht nur auf die darüber liegende zugreifen kann. Dies schafft zusätzlich Trägerunabhängigkeit und ermöglicht die Kapselung von legacy Diensten oder schützt Server vor legacy Clients. Ein weiterer Vorteil ist, dass durch das Ermöglichen von Load Balancing die Skalierbarkeit verbessert wird. Allerdings wird durch das Durchreichen der Schichten ein Overhead erzeugt, der zu größer Latenz führt.

code on demand Diese Bedingung ist optional, was bedeutet, dass zum Beispiel innerhalb eines Firmennetzes das Nachladen von Code erlaubt wird, aber das Laden von außen durch eine Firewall verhindert wird. Ein Vorteil des code on demand ist, dass Clients weniger Funktionalität implementieren müssen, da sie diese vom Server nachladen können. Beispiele für aktuelle Technologien für diese Bedingung sind Java Applets und JavaScript.

2.2.2 Architektur Elemente

Representational State Transfer identifiziert drei verschiedene Arten von Elementen innerhalb der Architektur: Komponenten, Konnektoren, welche die Kommunikation zwischen den Komponenten ermöglichen, sowie Daten Elemente, die ausgetauscht werden. Bei diesen Elementen werden Details der Protokollsyntax und Implementierung von Komponenten ignoriert.

Daten Elemente Daten Elemente sind das Schüsselelement innerhalb von REST. Sie beschreiben die Beschaffenheit und den Zustand einer Ressource. REST Komponenten kommunizieren durch den Austausch von Repräsentationen von Ressourcen, wobei das Format der Representation dynamisch im Request gewählt werden kann. Dies hat den Vorteil, dass die Beschaffenheit der eigentliche Ressource verborgen bleibt. Als Ressource innerhalb von REST kann alles bezeichnet werden, das benannt werden kann, so definiert Fielding (2000) eine Ressource als:

> „In other words, any concept that might be the target of an author's hypertext reference must fit within the definition of a resource. A resource is a conceptual mapping to a set of entities, not the entity that corresponds to the mapping at any particular point in time."

Um Ressourcen innerhalb der Kommunikation von Komponenten zu identifizieren ist der so genannte Ressource Identifier notwendig. Dieser wird von keiner Autorität innerhalb eines System gewählt, sonder wird von jeden Autor selbst bestimmt. Zum Austausch von Zuständen der Ressourcen werden in REST Repräsentationen verwendet. Sie halten alle Informationen zu einer Ressource in einem gewählten standardisierten Datenformat.

Konnektoren Konnektoren dienen als abstrakte Schnittstelle, die es ermöglicht auf Repräsentationen von Ressourcen zuzugreifen und sie zu manipulieren. Ihr Einsatz führt zur Erhöhung der Seperation of Concerns und verbirgt die Implementation einer Ressource. Durch die Bedingung stateless also der Zustandlosigkeit, muss ein Konnektor keine Zustände zwischen Request halten, da alle nötigen Informationen im Request enthalten sind. Ein weiterer Vorteil besteht in der einfacheren Parallelisierbarkeit der Anwendung, da eine verarbeitende Komponente nicht die Semantik der Interaktion kennen muss. Des weiteren ermöglicht das Vorhandensein aller wichtigen Informationen im Request ein effizientes Caching. Als verschiedene Ausprägungen von Konnektoren definiert Fielding einen *Client* der Anfragen stellt und einen *Server*, der sie beantwortet. Außerdem beschreibt er den Typ des *Cache*, der es ermöglicht, generierte Responses wiederzuverwenden. So wird durch den Einsatz auf Clientseite der Netzwerkverkehr verringert und auf Seite des Servers Rechenleistung eingespart. Die Aufgabe Teile oder ganze Ressource Identifier zu übersetzen übernimmt in REST der *Resolver* Konnektor. Als Beispiel kann der DNS Dienst angesehen werden. Als weiteren Konnektortyp definiert Fielding den *Tunnel*, der eine Kommunikation über Verbindungsgrenzen, wie Firewalls, ermöglicht. Dieser Typ ist Bestandteil von REST, da aktive Komponenten zu Tunneln werden können. Als Beispiel hierfür wird ein HTTP Proxy angeführt.

Komponenten Als Komponenten werden innerhalb von REST Elemente benannt, die Repräsentationen durch Konnektoren übertragen oder sie verarbeiten. Die *User Agent* Komponente verwendet einen Client Konnektor um Request zu stellen und Responses zu erhalten. Als Beispiel hierfür kann ein Web Browser angesehen werden. Die vom User Agent gestellten Request werden von Server Konnektoren einer *Origin Server* Komponente angenommen. Die Komponenten stellen für eine Ressource die einzige Quelle für Repräsentationen dar und führen als einzige Änderungen an ihr durch. Hierfür stellt diese Komponente ein generisches Interface als Resourcenhierachie bereit. Ein *Proxy* innerhalb von REST wird von einem Client ausgewählt, um einen anderen Dienst zu verbergen, Daten zu übersetzen, die Performanz zu verbessern oder Schutz vor Angreifern zu bieten. Die *Gateway* Komponente stellt die selben Eigenschaften Origin Server Komponenten zur Verfügung.

2.2.3 Architektur Sichten

Nachdem die Elemente von Representional State Transfer beschrieben wurden, sollen nun die Architektur Sichten die Interaktion der Elemente verdeutlichen.

Prozess Sicht Die Prozess Sicht dient zur Darstellung der Interaktionsbeziehungen der Komponenten, durch Verfolgen des Datenflusses im System. Ein Beispiel für eine mögliche Visualisierungen bietet Abbildung 2.5. Hieraus zieht Fielding (2000) folgende Vorteile des Representational State Transfer:

- Durch die client/server Seperation of Concerns bietet REST eine einfache Implementierungen der Komponenten, außerdem wird die Komplexität der Konnektoren reduziert und es werden gute Möglichkeiten zur Verbesserung der Performanz, sowie Skalierbarkeit der Serverkomponenten geboten.
- Durch die layered system Bedingung können Proxys, Gateways oder Firewalls in das System eingebaut werden ohne die Interfaces anzupassen. Diese Eigenschaft von REST ähnelt laut Fielding (2000) dem Pipes and Filter Stil in Unix.

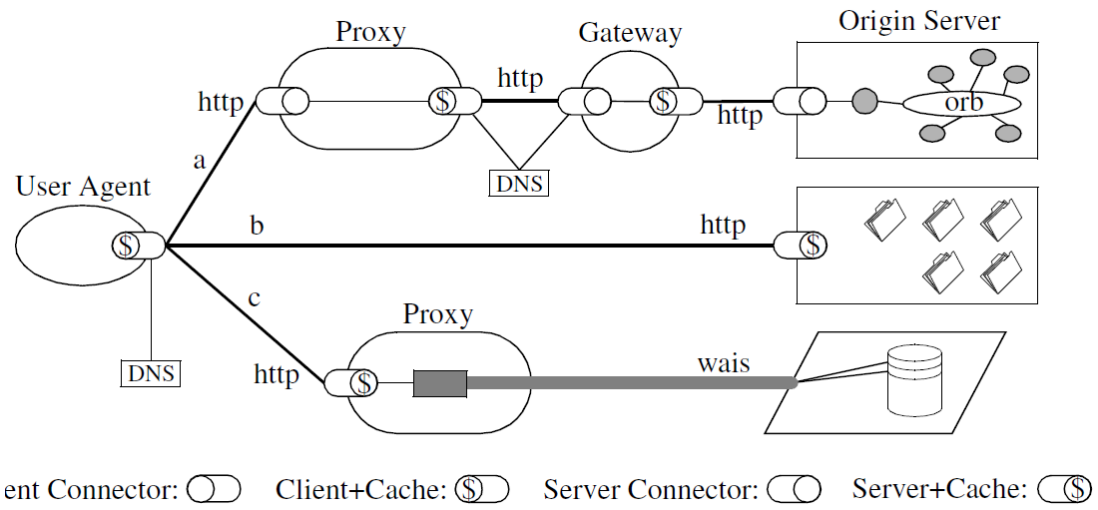

Abbildung 2.5: Beispiel für die Prozess Sicht eines Systems aus Fielding (2000)

Konnektor Sicht Das Hauptaugenmerk der Konnektor Sicht liegt auf der Veranschaulichung der Kommunikation zwischen den Konnektoren. In REST wird kein Protokoll zur Implementierung festgelegt, allerdings wird die Wahl durch die Definition beschränkt. Es ist

client Konnektoren möglich, je nach Request einen anderen Server für die Abarbeiten auszuwählen, so kann auch durch Proxys zwischen zwei REST-konformen Protokollen gewechselt werden.

Daten Sicht Die Daten Sicht in REST stellt den Zustand der Anwendung als Informationsfluss durch die Komponenten dar. Da Representational State Transfer für verteilte Informationssysteme ausgelegt ist, definiert es Anwendungen als eine zusammenhängende Struktur von Informationen mit Kontrollalternativen, wodurch es einem Benutzer möglich ist, eine Aufgabe zu erledigen. In dem Kommunikationsfluss zwischen Komponenten können verschieden große Nachrichten versendet werden, so dienen kleine meist für Kontrolldaten, wohingegen in großen Nachrichten Repräsentationen von Ressourcen versendet werden. Bei Letzteren bietet sich allerdings die Möglichkeit, sie effizient zu cachen. Ein Zustand einer Anwendung wird in REST durch folgende Faktoren beeinflusst:

- unerledigte Anfragen
- Topologie der verbundenen Komponenten
- aktive Anfragen an den Konnektoren
- Datenfluss der Repräsentation als Antwort auf eine Anfrage und
- die Verarbeitung einer Repräsentation am User Agent

Als ein spezieller Zustand wird der steady-state definiert. Eine Anwendung ist in diesem Zustand, wenn keine Anfragen mehr beantwortet werden müssen. Der Anwendungszustand wird vom Client verwaltet und gespeichert. Er kann aus mehreren Repräsentationen von Server zusammengesetzt sein. Dies hat den Vorteil, dass die Implementation des Servers vereinfacht wird und kein Speicher benötigt wird. Allerdings hat der Client die volle Kontrolle über den Zustand.

2.2.4 Anwendung

Die Anwendung des Representational State Transfer Architekturstil wird in Dix (2010) und Wirdemann und Baustert (2008) als RESTful Design bezeichnet und wird bei der Realisierung von Web Anwendungen angewandt. Diesen Anwendungen liegt das HTTP Protokoll zu Grunde, wobei zum Abfragen und der Manipulation von Ressourcen die vier HTTP-Methoden GET, PUT, POST und DELETE verwendet werden. Zur Identifikation der Ressourcen werden URLs verwendet. Hierbei wird in der Literatur folgendes Aufbauschema

beschrieben: `http://domain/ressourcenname/resourcenprimärschlüssel`. Fehlermeldungen oder andere Meta Informationen zwischen Server und Client können in HTTP Header übermittelt werden.

Beispiel Zur Verdeutlichung des RESTful Design soll im Folgenden eine Ressource *user* mit den Attributen *id*, *first_name* und *last_name* abgefragt und manipuliert werden. Als Repräsentation soll das JSON Format genutzt werden.

- `GET http://localhost/users/1.json` Dieser Aufruf an den Server liefert den User mit der ID 1. Die JSON Repräsentation würde wie folgt aussehen:

```
{
  "id": 1,
  "first_name": "Steffen",
  "last_name": "Brauer"
}
```

- `POST http://localhost/users` Laut Dix (2010) verhält sich POST wie das append an eine Liste. Also wird mit diesem Aufruf, gefolgt von Parameter, welche die Attribute enthalten, eine Ressource der Liste der User angehängt. Wichtig ist hierbei, dass der Aufruf keinen Primärschlüssel enthält, da dieser vom Server vergeben wird. Ein POST Aufruf kann also mit einer create oder insert Methode assoziert werden.
- `PUT http://localhost/users/1` Ein PUT Aufruf legt eine Ressource an der spezifizierten URL ab. Dies wird laut Dix (2010) in Ruby on Rails mit einer update Methode gleichgesetzt, die auch teilweise Aktualisierungen zulässt.
- `DELETE http://localhost/users/1` Dieser Aufruf veranlasst den Server, die in der URLs adressierten Ressource zu löschen.

Vorteile Wirdemann und Baustert (2008) nennt vielfältige Vorteile für RESTful Design:

- Das Beschreiben der Ressourcen durch saubere URLs bietet einen konsistenten Zugriff und macht Ressourcen aktionsunabhängig.
- Durch die Wahlmöglichkeit der Repräsentationen kann eine Anwendung verschiedene Clients bedienen und wird so automatisch multiclientfähig.
- Ein weiterer Vorteil der verschiedenen Repräsentation manifestiert sich in der nicht benötigten Wiederholung von Code zur Generierung verschiedener Repräsentationen. Dies entspricht dem DRY Prinzip.

- Ein solches konzeptionelle sauber Design führt zu einem gut lesbaren und wartbaren Anwendungscode.

2.3 Funktionen Sozialer Netzwerke

Um einen ersten Eindruck von der Bedeutung und Funktion sozialer Netzwerke zu vermitteln, sollen im Folgenden die Funktionen beschrieben werden. Hierbei handelt es sich weniger um die technischen, als vielmehr um die sozialen Funktionen. In Richter und Koch (2008) wurden vier soziale Netzwerke(Facebook, StudiVZ, Xing und LinkedIn) untersucht und ihre grundlegenden Funktionen in verschiedenen Gruppen aufgeteilt. Hierbei handelt es sich nicht um eine scharfe Trennung, da die Nutzungsintention der Funktionen von jeweiligen Nutzer abhängt. In Abbildung 2.6 sollen die verschiedenen Funktionsgruppen als Abfolge von Nutzungsschritten dargestellt und im weiteren genauer beschrieben werden.

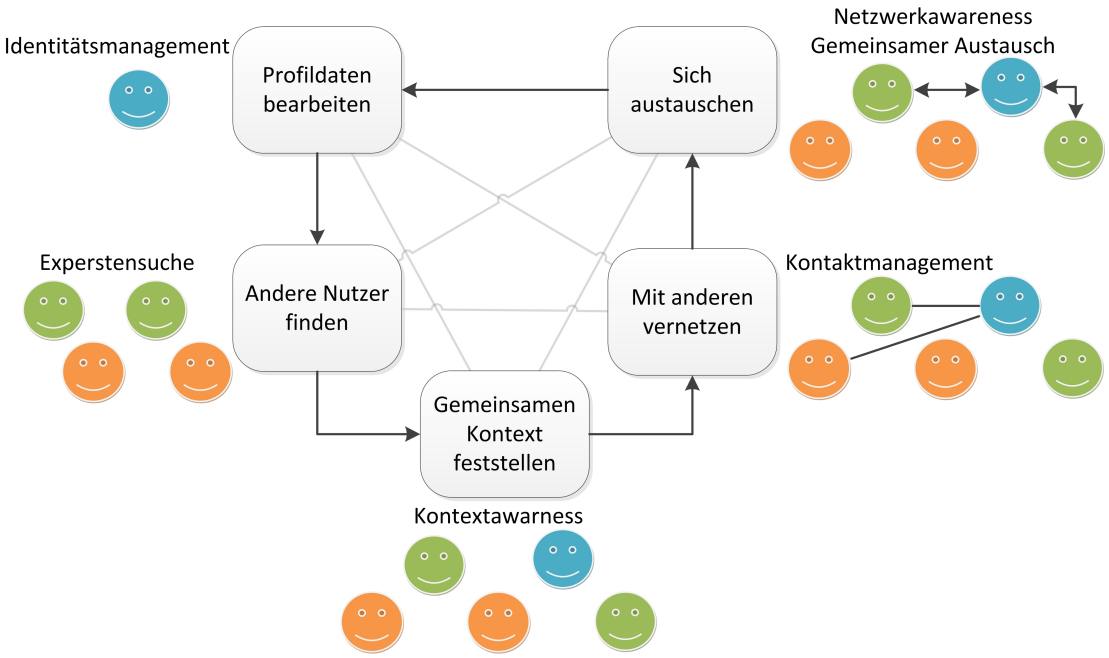

Abbildung 2.6: Funktionen eines sozialen Netzwerkes nach Richter und Koch (2008)

Gemeinsamer Austausch Der Austausch in sozialen Netzwerken kann über mehrere angeboten Funktionen geschehen: Es besteht die Möglichkeit Nachrichten zwischen einer kleinen Zahl von Nutzern zu versenden, in Foren zu diskutieren und kurze Texte über den aktuellen

Zustand oder Aktivitäten auf einer Pinnwand zu veröffentlichen, dem so genannten Microblogging. All diese Möglichkeiten der Kommunikation werden mit einem Login ermöglicht und es sind keine weiteren Daten zu Authentifizierung nötig. Viele Netzwerke bieten neben den schon erwähnten Kommunikationsmöglichkeiten eine Art Minimalkommunikation, dem sogenannten Stupsen oder Gruscheln. All diese Funktionen stellen auch einen erheblichen Beitrag zur schon erwähnten Netzwerkawareness.

Netzwerkawareness Der Begriff Netzwerkawareness beschreibt „das Gewahrsein über die Aktivitäten (bzw. den aktuellen Status und Änderungen des Status) der Kontakte im persönlichen Netzwerk“Richter und Koch (2008). Diese Funktion hat sich im Hinblick der Zeit, die auf der Plattform verbracht wird, als Erfolgsfaktor erwiesen. Hierbei wird zwischen Push- und Pull-Funktionen unterschieden. Bei der Push-Funktion werden nach dem Login Informationen über Aktivitäten des persönlichen Netzwerks, sowie zum Beispiel Geburtstag angezeigt. Die Pull-Funktionalität beschreibt das aktive Besuchen der Profile anderer Nutzer, um sich über ihre Aktivitäten zu informieren.

Identitätsmanagement Als Kern aller sozialen Netzwerken können die individuellen Profile der Nutzer angesehen werden. Mit diesen ist es dem Nutzer möglich sich bewusst und kontrolliert einer breiten Masse vorzustellen. Außerdem wird die Möglichkeit zahlreicher unterschiedlicher Gruppen beizutreten genutzt, um über den Gruppennamen einen Standpunkt einzunehmen. Dabei wird jedoch häufig die Kommunikation mit anderen Gruppenmitglieden nur wenig oder gar nicht genutzt. Ein weiteres Element des Identitätsmanagements in sozialen Netzwerken besteht darin, dass die sogenannte Pinnwand oder das Gästebuch genutzt wird, um öffentlich mit eng befreundeten Kontakten zu kommunizieren.

(Experten-) Suche Die Suche nach Kontakten ist ein zentraler Bestandteil der Netzwerke. Hinzu kommt die Möglichkeit in Business Netzwerken nach Experten für Fachgebiete zu suchen. Die Suche kann entweder aktiv nach verschiedenen Kriterien wie Name, Interessen oder Firma geschehen, es werden aber auch mögliche Kontakte durch das jeweilige Netzwerk vorgeschlagen. Um eine möglichst genaue und effiziente Suche anzubieten, ist es notwendig so viele Daten wie möglich des Benutzers zu speichern. Dies steht aber in Konflikt mit mehreren Nutzungsbarrieren, wie „Motivation, Ängste“ und „mangelnde Kenntnisse über effektives Identitätsmanagement“Richter und Koch (2008).

Vertrauensaufbau (Kontextawareness) Als eine Grundlage menschlicher Beziehungen kann gegenseitiges Vertrauen angesehen werden. Daher versuchen soziale Netzwerke dies

möglichst schnell herzustellen. Hierfür soll ein gemeinsamer Kontext konstruiert werden, um eine erfolgreiche Kommunikation zu ermöglichen. Dies geschieht durch das Anzeigen von gemeinsamen Bekannten, des persönlichen Netzwerks oder der Beziehungen zu anderen Personen. Des weiteren können Informationen im Profil oder Beiträge in gemeinsam genutzten Foren den gemeinsamen Kontext fördern oder zu bisher unbekannten Personen erst herstellen.

Kontaktmanagement Kontaktmanagement beschäftigt sich mit der Pflege des persönlichen Netzwerks des Benutzers. Die Verwaltung innerhalb des Netzwerks hat den Vorteil, dass die Kontaktinformation von jedem Nutzer selbst bearbeitet werden und so aktuell sind. Außerdem besteht die Möglichkeit die Kontakte zum Beispiel in Listen zu ordnen.

3 Konzeption der Anwendung

Zu Beginn der Kapitels soll die Vision des Systems entwickelt werden. Aus ihr wird im folgenden zuerst ein fachliches Konzept entwickelt und im Anschluss hieraus ein technisches Konzept abgeleitet.

3.1 Systemvision

In diesem Abschnitt soll zunächst eine Vision des Systems erstellt und im Anschluss die Anwendungsfälle, die in dieser Arbeit zu entwickeln sind, abgeleitet werden.

Das System soll in der Lage sein sämtliche Objekte aus den unterstützten Netzwerken anzeigen zu können, sowie die Objekte zu Netzwerken hinzufügen können. Gemeinsame Objekte sollen erkannt werden und es soll möglich sein, sie über Netzwerke hinweg zu aggregieren und ein Objekt in mehrere Netzwerke parallel zu veröffentlichen. Objekte, die dieselben realen Objekte repräsentieren sollen über Netzwerkgrenzen hinweg assoziiert werden können. Um der Definition der Cloud zu entsprechen, soll das System schnell auf Laständerungen reagieren können und die benötigten Ressourcen anpassen.

Aus dieser Vision leiten sich folgende in dieser Arbeit zu entwickelnde Anwendungsfälle ab:

Ein Benutzer meldet sich an

Um eine Anmeldung eines Benutzers bei der Anwendung durchzuführen, muss er auf der Startseite der Anwendung ein Netzwerk auswählen, über das er die Authentifikation durchführen möchte. Nach der Anmeldung bei dem Netzwerk, ist der Benutzer auch bei der Anwendung angemeldet.

Aggregierte Übersicht von Aktivitäten

Der Benutzer kann sich in der Anwendung eine Übersicht von aktuellen Aktivitäten aus allen verfügbaren Netzwerken in denen er angemeldet ist anzeigen lassen. Zur dieser Ansicht wird der Benutzer nach der Anmeldung weitergeleitet.

Veröffentlichen von Posts in mehreren Netzwerken

Es soll möglich sein über ein Formular einen Post gleichzeitig in mehreren Netzwerken zu Veröffentlichen. Hierzu sollen die Netzwerke einzeln auswählbar sein. Außerdem sollen eventuelle Einschränkungen auf Anwendungsebene berücksichtigt werden.

Freundschaftsbeziehungen anzeigen

Ein Benutzer kann sich sein Benutzerprofil anzeigen lassen und über dieses eine Liste seiner Freunde abrufen, von denen wiederrum die Profile angezeigt werden können.

3.2 Konzeption des fachlichen Kerns

Um das Konzept des fachlichen Kerns darzustellen, sollen im ersten Schritt verfügbare Schnittstellen von sozialen Netzwerken dargestellt werden. Auf ihrer Grundlage wird im Anschluss die benötigte Ebene der Abstraktion erarbeitet und eine fachliche Architektur erstellt.

3.2.1 Angebotene Schnittstellen

Um eine Kapselung sozialer Netzwerke zu ermöglichen, müssen die von ihnen bereitgestellten Schnittstellen genutzt und die gewonnenen Daten auf eine höhere Ebene abstrahiert werden. Hierfür werden im ersten Schritt die Schnittstellen einiger sozialen Netzwerke vorgestellt. Die Auswahl wurde aufgrund von Popularität und somit der Größe von verfügbare Daten getroffen.

Facebook Graph API

Facebook ist das zurzeit größte soziale Netzwerk. Laut eines Bericht auf Facebook (2011b) hat es über 500 Millionen aktive Nutzer, die 700 Milliarden Minuten pro Monat online sind. Zugriff auf die, von den Benutzer produzierten Daten, bietet die Facebook Graph API. Hier werden alle Objekte innerhalb des Netzwerks als Knoten mit verschiedenen Verbindungen im sozialen Graphen aufgefasst. Ein Beispiel hierfür bietet die Graph Facebook (2011d): Zugriff auf die Seite der Facebook Plattform mit der URL `https://graph.facebook.com/19292868552` gibt ein JSON Objekt zurück:

```
1 {
2    "id": "19292868552",
3    "name": "Facebook Platform",
4    "picture": "http://profile.ak.fbcdn.net/hprofile-ak-snc4/
```

```
    211160_19292868552_1855422_s.jpg",
    "link": "http://www.facebook.com/platform",
    "category": "Product/service",
    "likes": 1920294,
    "website": "http://developers.facebook.com",
    "username": "platform",
    "founded": "May 2007",
    "company_overview": "Facebook Platform enables anyone
    to build social applications on Facebook and the web.",
    "mission": "To make the web more open and social."
}
```

Hierbei ist `19292868552` die ID des Objekts. Auf diese Weise kann auf jedes Objekt im sozialen Graphen Facebooks zugegriffen werden. Verbindungen zwischen verschiedenen Objekten werden wie folgt abgefragt: Um alle Einträge auf der Pinnwand der Seite zu erhalten, wird die URL `https://graph.facebook.com/19292868552/feed` verwendet. Dieser Request erzeugt einen JSON- Array der Objekte des Typs Post enthält. Um Zugriff auf diese Daten zu erhalten, muss sich ein Benutzer über das oauth 2.0 Protokoll anmelden.

Twitter

Twitter ist ein populärer Dienst, um kurze Nachrichten von maximal 140 Zeichen auszutauschen. Laut einem Artikel auf Twitter (2011b) sind mehr als 200 Millionen Nutzer registriert, wobei jeden Tag 460.000 hinzukommen. Diese Nutzer produzieren 155 Millionen der sogenannten Tweets pro Tag. Zur Interaktion bietet Twitter eine REST API Twitter (2011a) an. Hier werden die Twitter Ressourcen in verschiedenen Formaten angeboten. Um innerhalb von Twitter nach Benutzern oder speziellen Tweets zu suchen, stellt Twitter eine spezielle Search API bereit. Der Zugriff auf einen Tweet funktioniert wie folgt: `http://api.twitter.com/version/statuses/show/:id.format`. Hierbei bestimmt das angegebene Format den Rückgabetyp. Üblich sind JSON oder XML. Ein Unterschied zur Facebook Graph API ist, dass die IDs der Objekte nur eindeutig innerhalb der selben Klasse sind. Eine Authentifizierung bei Twitter ist nur notwendig um benutzerbezogene Daten abzufragen. Alle Tweets sind öffentlich abrufbar. Um einen Nutzer zu authentifizieren wird oauth in Version 1.0a angeboten.

OpenSocial

OpenSocial stellt den Versuch dar, eine einheitliche Schnittstelle für soziale Anwendungen zu bieten, um die Interaktion über mehrere Webseiten einfacher zu gestalten. Hierbei umfasst OpenSocial die Spezifikation von Schnittstellen, die von verschiedenen Anbietern implementiert werden. Zur Authentifizierung von Nutzer und Anwendungen wird bei OpenSocial ebenfalls oauth 1.0a eingesetzt (Beri (2010)). Zum Konsumieren der Daten werden eine REST sowie eine RPC API spezifiziert, über die es möglich ist die Daten abzurufen. Als Kerndaten wurden von OpenSocial die Entitäten Activity, AppData, Group, Message und Person identifiziert, welche jeder Container, also Anbieter der Schnittstelle, bereitstellen muss.

3.2.2 Abstraktion

Um den benötigten Abstraktionsgrad zu bestimmen, sollen die Funktionalitäten der oben genannten Netzwerke, sowie Anderson (2010) herangezogen werden. In dieser Studie wurden neun sogenannte social network sites analysiert. Hierzu meldete sich ein Mitarbeiter bei den Netzwerken an und probierte alle Funktionen aus. Die Studie kam zu dem Ergebnis, dass alle untersuchten Netzwerke folgende fünf der vierzehn möglichen Funktionen beinhalten:

- Profile
- Fotomanagement
- E-Mail
- Netzwerkmanagement und
- Suche

Aus diesem Ergebnis lässt sich folgern, dass zwar viele soziale Netzwerke eine Grundfunktionalität bieten, aber jedes Netzwerk zusätzliche Funktionen anbietet. Ein genauer Überblick über das Ergebnis bietet Abbildung 3.1. Ihr liegt einer Auswertung von Anderson (2010) zugrunde, die um die hier verwendeten Netzwerken erweitert wurde. Die Betrachtung der Schnittstellen der oben genannten Netzwerke kommt zu dem selben Ergebnis: So gehören bei der Facebook Graph API sechs der 22 angebotenen Entitäten zu den fünf identifizierten Grundfunktionalitäten. Die anderen Entitäten beziehen sich auf weitere Funktionalitäten wie Statusnachrichten oder Eventmanagement. Auch die Twitter REST API stellt von 20 verfügbaren Entitäten fünf der Grundfunktionalität bereit, wobei Fotomanagement nicht unterstützt wird. Bei der open social Schnittstelle sind vier von zwölf für die Grundfunktionalität relevant.

	Profile	Comment Wall	Pic Mgmt	EMail	IM	Status Mgmt	Rec Sys	Event Plan	Net Mgmt	Group Mgmt	Blog	Video	Tags	Search
MySpace	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Facebook	X	X	X	X	X	X	X	X	X	X		X	X	X
LinkedIn	X		X	X		X	X		X	X	X		X	X
Yahoo!	X	X	X	X		X			X		X			X
Classmates	X	X	X	X			X	X	X	X			X	X
Youtube	X		X	X			X		X			X		X
BlackPlanet	X	X	X	X	X	X			X	X	X	X		X
Orkut	X	X	X	X	X	X	X	X	X	X		X		X
Hi5	X	X	X	X		X	X		X	X			X	X
Twitter	X	X		X			X		X					X
OpenSocial	X	X	X	X		X	X		X	X		X		X

Abbildung 3.1: Unterstützte Funktionen sozialer Netzwerke Anderson (2010)

Aus dieser Vielfalt an Entitäten und Funktionalitäten lässt sich das Ziel ableiten, einen möglichst hohen Grad an Abstraktion zu erreichen. Er sollte aber dadurch limitiert werden, dass auch Objekte aus verschiedenen Netzwerken mit derselben Schnittstelle benutzbar sein sollen, um zum Beispiel das Posten einer Nachricht in mehreren Netzwerke in einem Schritt zu ermöglichen.

Die größte zu erreichende Abstraktion ist, dass ein soziales Netzwerk aus Objekten besteht, die miteinander in einer Beziehung stehen. Somit kann ein soziales Netzwerk als ein ungerichteter Graph verstanden werden, wobei die Knoten des Graphen durch die Entitäten und die Verbindungen, wie die Freunde-Beziehung, als Kanten repräsentiert werden. Das in Abbildung 3.2 genannte Beispiel soll dies nochmal verdeutlichen:

Die Nutzer Hinz und Kunz werden innerhalb des sozialen Netzwerkes durch ihre Profile repräsentiert. Ihre Freundschaft wird durch die Kante mit dem Label befreundet dargestellt. Eine Nachricht mit dem Inhalt Hallo! die von Hinz an Kunz gesendet wurde, wird durch einen Knoten des Typs Nachricht repräsentiert, der eine Kante gesendet zu Hinz und eine Empfänger zu Kunz hat. Eine Typisierung innerhalb des Graphen ist notwendig, um die oben genannte gemeinsame Schnittstelle von zum Beispiel Nachrichten zu ermöglichen, wobei ein Typ durch die Menge der Attribute beschrieben wird, die ein Knoten besitzt.

3.2.3 Architekturübersicht

Das Ziel der Architektur ist es, einem beliebigen Frontend eine konsistente und abstrakte Schnittstelle für verschiedene soziale Netzwerke zu liefern. Hierzu wird von der SocialObjects Komponente das Interface SocialObject bereitgestellt, das ein Objekt in einem sozialen Netzwerk repräsentiert oder es ermöglicht, neue Objekte in ein soziales Netzwerk hinein zu schreiben. Innerhalb der SocialObjects Komponente befinden sich die Komponenten der

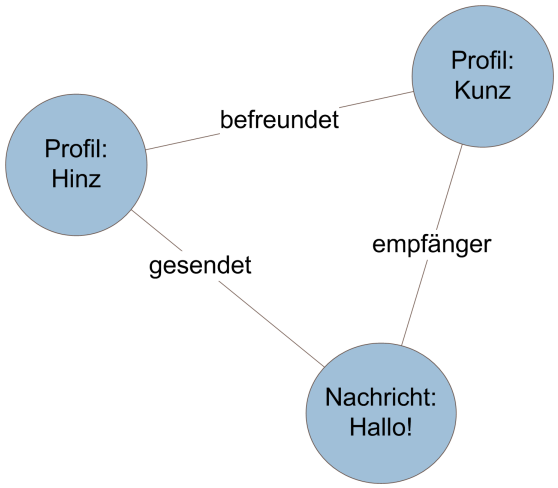

Abbildung 3.2: Beispiel für die Auffassung eines sozialen Netzwerkes als Graph

jeweiligen Netzwerke, in denen sich die Implementierungen der Objekte befinden. Um das Laden der Objekte via den Netzwerk spezifischen Schnittstellen zu ermöglichen, weist jede Netzwerkkomponente eine HTTP Schnittstelle auf. Des weiteren sind Schnittstellen zur persistenten Speicherung der benötigten Authentifizierungsmerkmale vorhanden. Im weiteren sollen die einzelnen Komponenten genauer erläutert werden, eine Übersicht der Architektur befindet sich in Abbildung 3.3 in UML Notation.

3.2.4 Socialobjects Komponente

Wie bereits erwähnt, kapselt die SocialObjects Komponente den Zugriff auf die verschiedenen sozialen Netzwerke in sich. Daher ist es ihre Aufgabe, Objektanfragen entgegen zu nehmen und sie zu dem jeweiligen Netzwerk weiterzuleiten. Hierzu muss die Komponente alle in ihr enthaltenen Netzwerke kennen. Dieses Wissen soll später möglichst dynamisch realisiert werden. Eine weitere Aufgabe der SocialObjects Komponente ist es, die Benutzer der Anwendungen zu speichern um so eine plattformübergreifende Zuordnung eines Nutzers in verschiedenen Netzwerken zu ermöglichen.

Kopplung

Die Kopplung zwischen der SocialObject Komponente zum Frontend und den Implementierungen der Netzwerke soll sich auf eine get und eine post Methode beschränken, die sowohl die SocialObject Komponente also auch die Netzwerkimplementationen ausweist. Hier durch wird die Aufgabe der Delegation von Anforderungen und Manipulationen an die einzelnen

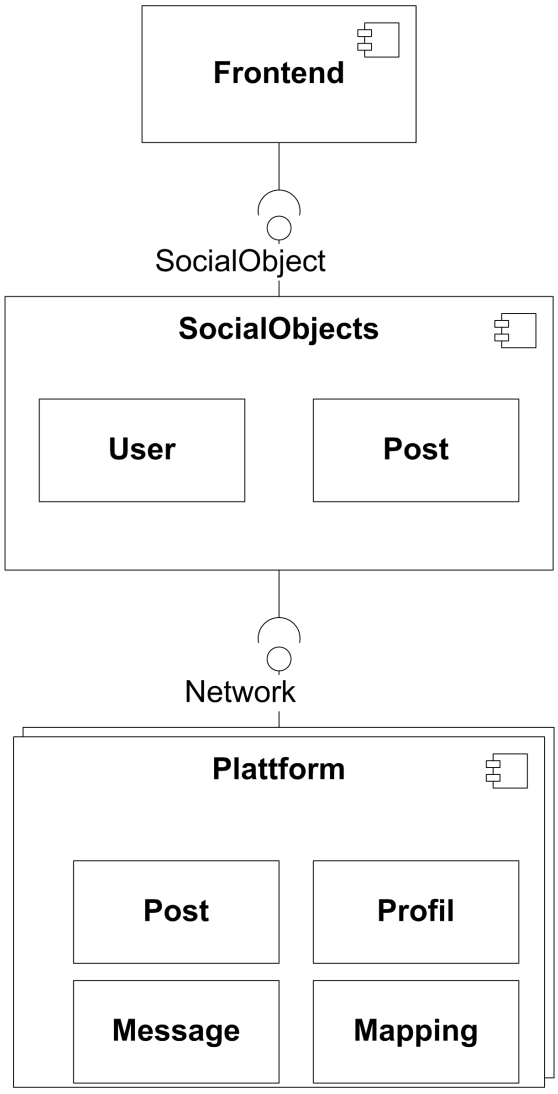

Abbildung 3.3: Fachliche Architektur des Kerns

Netzwerke deutlich. Die Aufrufe der Methoden vom Frontend erfolgen an der SocialObject Klasse. Um die Aufrufe an die Netzwerke zu delegieren, soll eine Abbildungen verwendet werden, welche den Namen des Netzwerkes auf die Klasse abbildet, die die get und post Methoden des Netzwerkes anbietet.

Abbildung von Beziehungen

Eine weitere Funktionalität der SocialObject Komponente umfasst das Darstellen von Beziehungen zwischen den Objekten. Wie in 3.3.1 festgelegt, werden Objekte als Knoten in einem Graph aufgefasst. Die Kanten innerhalb des Graphen bilden die Beziehung zwischen den Objekten. Um also die Freunde eines Nutzers abzurufen muss dem Benutzer der Verbindungsname übergeben werden. Dies ist in diesem Fall *friends*. Es soll ein Connection Objekt zurück gegeben werden, welches den Namen der Beziehung und eine Liste mit Objekten, die in dieser Beziehung zu dem Benutzer stehen, enthalten. Ein Nachteil der durch dieses sehr generische Konzept entsteht, ist die Erhöhung der möglichen Objekte eines Netzwerkes, da manche Netzwerke spezielle Repräsentationen von Objekten bei der Abfrage von Beziehungen verwenden.

Schreiben von Objekten in Netzwerke

Um beim Schreiben von Objekten in Netzwerke die Kopplung möglichst gering zu halten, gibt es zu jedem Typ von Objekt, der geschrieben werden soll eine Klasse. Sie hält alle benötigten Informationen. Im Falle einer Nachricht handelt es sich hier um den Text, den Empfänger, sowie den Sender. Die Objekte werden dann an die SocialObjects Komponente zusammen mit dem Zielnetzwerk übergeben. Hier erfolgt der selbe Delegationsmechanisums, wie bei dem Abfragen von Objekten.

3.2.5 Plattform Komponenten

Die Plattform Komponente soll die Implementation eines sozialen Netzwerks darstellen. Hierbei werden sich mehrere Instanzen dieser Komponente im System befinden, welches durch die sogenannte Multi-Komponenten Notation in Abbildung 3.3 angedeutet werden soll. Die Aufgabe der Komponente ist es, Objektanfragen von der SocialObjects Komponente entgegen zu nehmen und dann eine Repräsentation des angeforderten Objektes zurück zu liefern. Hierzu hat jede Plattform Komponente eine Schnittstelle zum sozialen Netzwerk und ist für die Abbildung des globalen Nutzers zu dem jeweiligen Netzwerkaccount zuständig.

3.2.6 Frontend

Die Aufgabe der Frontend Komponente innerhalb der Anwendung ist es, Anfragen des Benutzers an die SocialObjects Komponente weiterzuleiten und die erhaltenen Repräsentationen in eine vom Benutzer geforderte Repräsentation umzuwandeln. Als Frontend dieser konkreten Anwendung ist eine REST-Schnittstelle geplant, bei der das MVC Pattern angewendet werden soll.

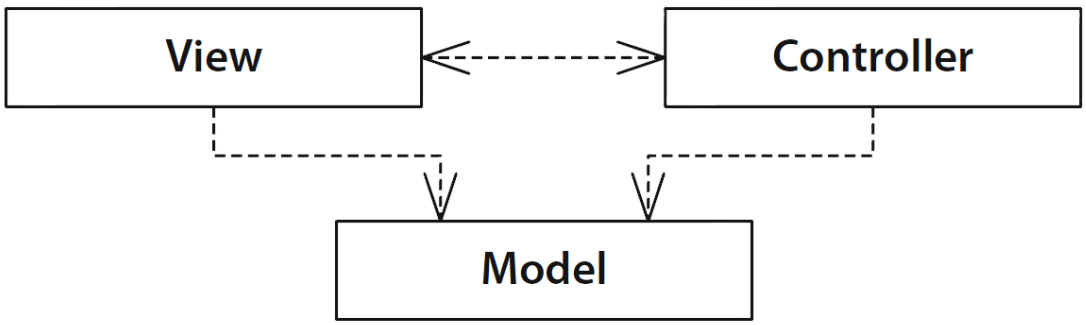

Abbildung 3.4: Abhängigkeiten des MVC Pattern

MVC Pattern Laut Eilebrecht und Starke (2010) hat das MVC Pattern als Ziel einen Austausch der Benutzerschnittstelle zu vereinfachen. Hierfür werden die Verantwortlichkeiten in *Model*, *View* und *Controller* aufgeteilt. Das Model kapselt die Daten, der View die Darstellung und der Controller die Manipulationen der Daten. Die Trennung führt zu einer besseren Wartbarkeit, da so zyklische Abhängigkeiten vermieden werden. Das MVC Pattern eignet sich besonders für die Implementation eines REST Service. Jeder Ressource wird ein Controller zugeordnet, der die verschiedenen Repräsentationen vorhält.

3.3 Konzeption des technischen Kerns

Nachdem nun das fachliche Konzept der Anwendung erarbeitet wurde, soll im nächsten Abschnitt ein technischen Konzept zur Umsetzung dargestellt werden. Im ersten Teil soll der Zugriff auf die Objekte diskutiert werden und im weiteren ein Protokoll zur Authentifizierung bei einem sozialen Netzwerk vorgestellt werden.

3.3.1 Adressierbarkeit der Objekte

Das in 3.2.2 beschriebene Konzept der Auffassung eines sozialen Netzwerkes als Graphen soll sich auch in der Adressierbarkeit der Objekte auf Ebene des Frontend, sowie der SocialObjects Komponente durchsetzen. Hierzu soll die konkrete Zugehörigkeit des Objektes zu einem Typ oder Netzwerk möglichst verborgen bleiben, um dem Prinzip des Information Hiding gerecht zu werden. Des weiteren soll nur der Knoten innerhalb des sozialen Graphen des Benutzer adressiert werden und nicht ein Objekt innerhalb eines Netzwerkes. Das Problem der Adressierbarkeit der Objekte besteht im Verwenden von REST Schnittstellen, sowie im Bereitstellen der selbigen. So muss jede Ressource ein Identifizierer zugewiesen werden.

Erzeugung künstlicher Identifizierer

Als Referenz für den ersten Ansatz ist die Facebook Graph API zu nennen (Facebook (2011d)). Hier werden alle Knoten innerhalb des Graphen mit einer über den Typ hinweg eindeutigen ID identifiziert und können über diese abgefragt werden. Diese Herangehensweise lässt sich nur mit erheblichen zusätzlichen Aufwand für die Anwendung umsetzen, da für jedes Objekt das je aufgerufen wird, eine künstliche ID generiert und gespeichert werden müsste, um mit dem Konzept von REST übereinzustimmen. Dies ist nötig, da sich ein Objekt innerhalb der Anwendung durch 3 Merkmale unterscheidet:

- Plattform
- Typ
- ID

Mit diesem Wissen wäre eine Tabelle erforderlich, in der neben den drei eben genannten Attributen zusätlich noch die künstliche ID gespeichert würde. Die Pflege der Tabelle wäre allerdings sehr aufwendig, da für jedes Objekt erst überprüft werden müsste, ob bereits eine ID ausgestellt wurden ist und wenn dies nicht der Fall wäre, eine erzeugt werden müsste. Auch würde die Größe der Tabelle bei einer geringen Anzahl an Nutzern sehr groß werden. So werden laut Facebook (2011b) bei Facebook jeden Monat 30 Milliarden Objekte von Benutzern erstellt.

Zusammengesetzte Identifizierer

Diese Art der Adressierung ist zwar konzeptionell am besten geeignet wäre, aber die Verwaltung der ID stellt eine schwer zu lösende Aufgabe dar. Diese Tabelle durch etwas Anderes

ersetzt werden. Es ist eine Übersetzung von ID auf die drei oben erwähnten Attribute erforderlich. Ein Ansatz dieses Problem zu lösen, ist die Attribute in die ID zu codieren. So würde eine ID aus einem Teil bestehen, das die Plattform angibt, gefolgt von einem Typteil und der eigentlichen Objekt ID. Bei diesem Ansatz stecken die Schwächen in der Codierung und Decodierung der IDs, sowie im Aufbau. Auch wenn in diesem Entwurf immer noch IDs verwendet werden, so verbirgt er nicht die einzelnen Attribute, sondern codiert sie nur.

Graph auf Objekt Ebene

Da auch der zweite Ansatz keinen Mehrwert zur einfachen Nennung der drei Attribute zeigt, soll das Konzept des Graphen nur auf Ebene der Objekte angewandt werden. Es werden Objekte über Plattformgrenzen hinweg durch Plattform, Typ und interne ID beschrieben, doch werden sie auf Objekt Ebene wie Knoten innerhalb des Graphen des jeweiligen Netzwerkes aufgefasst und auch so mit ihnen interagiert.

3.3.2 Authentifizierungsmöglichkeiten

Nachdem nun die Adressierung der Objekte konzipiert wurde, stellt sich im weiteren die Frage, wie der Zugriff auf die Objekte in den sozialen Netzwerken geschützt beziehungsweise geregelt wird. Zu diesem Zweck unterstützen alle API offene Authentifizierungsprotokolle. Als ein Beispiel für diese Protokoll soll im folgenden das oauth 2.0 Protokoll vorgestellt werden, wie es Facebook für die Graph API verwendet. Die Spezifikation des Protokolls ist unter Hammer-Lahav u. a. (2011) verfügbar. An dieser Stelle wird auf die unter Facebook (2011a) verfügbare Dokumentation der Umsetzung auf der Facebook Plattfrom eingegangen. Zur Durchführung der Authentifizierung bietet oauth 2.0 zwei verschiedene Protokollflüsse an:

authentication code flow Hier stellt ein Webserver Anfragen an die Graph API.

implicit flow Dieser Fluss wird benötigt, wenn ein Client direkt Anfragen stellt, zum Beispiel bei JavaScript auf Webseiten oder bei mobilen Apps.

In der hier konzipierten Anwendung werden nur Anfragen von einem Server gestellt, somit ist nur der erste Fluss relevant.

- Um den Dialog zu starten, stellt die Anwendung einen Request an den Facebook Server:

```
1 https://www.facebook.com/dialog/oauth?
2         client_id=YOUR_APP_ID&
3   redirect_uri=YOUR_URL
```

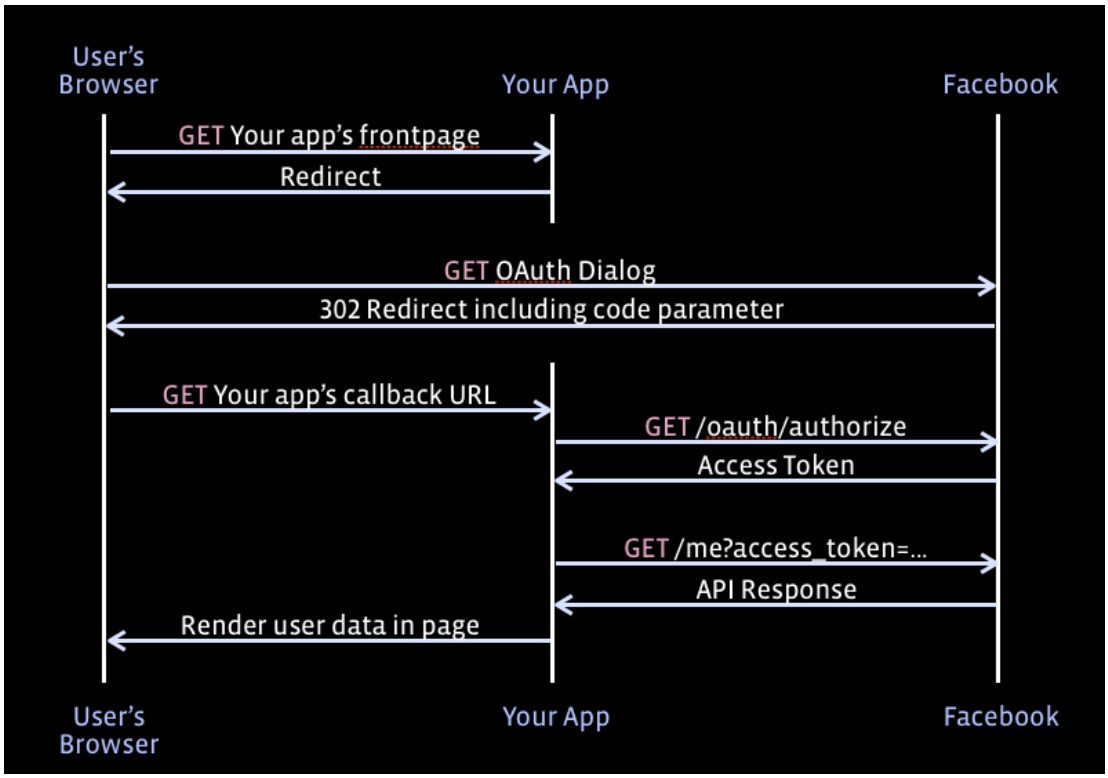

Abbildung 3.5: Server side flow Facebook (2011a)

Der Parameter `client_id` dient zur Authentifizierung der Anwendung anhand ihrer ID. In `redirect_uri` wird angegeben, auf welche Seite die Weitergeleitung erfolgen soll, wenn die Authentifizierung abgeschlossen ist.

- Im Anschluss muss sich der Benutzer auf der Facebook Seite anmelden und den Befugsnisanforderungen der Anwendung zustimmen. Ist dies geschehen, leitet Facebook den Benutzer zu der in `redirect_uri` spezifizierten Seite weiter.

- Ist die Authentifizierung gelungen und der Benutzer hat den Befugnissen zugestimmt, enthält die URL einen Parameter `code` der den Authentifizierungscode beinhaltet. Stimmt der Nutzer den Befugnissen nicht zu, so enthält die URL eine Fehlermeldung:

```
http://YOUR_URL?error_reason=user_denied&
  error=access_denied&
  error_description=The+user+denied+your+request
```

- Mithilfe des Authentifizierungscodes kann nun der access token erzeugt werden. Hierzu wird folgender Request an den Facebook Server gestellt:

```
https://graph.facebook.com/oauth/access_token?
        client_id=YOUR_APP_ID&
        redirect_uri=YOUR_URL&
        client_secret=YOUR_APP_SECRET&
        code=THE_CODE_FROM_ABOVE
```

  Gelingt die Authentifizierung wird ein `access_token` zurück gegeben. Ist dies nicht der Fall tritt ein JSON Objekt mit einer Fehlerbeschreibung an Stelle der `access_token`. Mit dem so erhaltenen Token kann die Anwendung jetzt Anfragen an die Graph API stellen.

Der Ablauf dieses Protokolls wird in der Abbildung 3.5 nochmal grafisch verdeutlicht.

4 Portierung in die Cloud

> „Although there are many articles describing the cloud and giving advice on chossing cloud vendors, there is a dearth of material on how to build a good cloud application“.

Sessions (2011)

4.1 PaaS Anbieter Heroku

Der einfachste Weg die bisher konzipierte Anwendung in die Cloud zu portieren, besteht in dem Deployment bei einem PaaS-Anbieter. Da die Realisierung in Ruby erfolgen soll, wurde Heroku Heroku (2011a) ausgewählt. Auf der Heroku Plattform ist es möglich Ruby Web Anwendungen laufen zu lassen. Die Kosten errechnen sich hier durch die Anzahl der verwendeten Dynos, der verwendeten Datenbank, sowie möglicher Add-ons. Heroku verwendet git um eine Anwendung auf der Plattform zu deployen. Jeder Anwendung die erstellt wird, wird ein git Repository zugewiesen. Lädt ein Benutzer die Anwendung hoch, wird sie sofort deployt. Die Skalierung der Anwendung geschieht durch die Erhöhung der Anzahl an Dynos, die Anfragen entgegen nehmen. Um die Eigenschaften einer Cloud zu erreichen, verwendet Heroku die in Abbildung 4.1 dargestellte Architektur. Sie enthält folgende Elemente:

Dyno Dynos stellen den Mittelpunkt der Heroku Architektur dar. Hierbei handelt es sich um voll isolierte Prozesse, denen verschiedene Aufgaben zu gewiesen werden können. Die Anzahl der Dynos, die zu einer Anwendung gehören, kann dynamisch jederzeit erhöht oder erniedrigt werden, wodurch die benötigte Elastizität entsteht Heroku (2011c). Jeder Dyno läuft in einem subvirtualisiertem Container, der eine Isolation der Ressourcen, der Prozesstabelle, sowie dem Dateisystem ermöglicht Heroku (2011b).

Dyno Manifold Das Dyno Manifold bildet eine verteilte Ausführungsumgebung für die Dynos. Seine Aufgabe ist es, die Dynos zu überwachen und bei Ausfall, einen neuen zu Starten. Außerdem verteilt das Dyno Manifold die Dynos so, dass sobald 2 Dynos

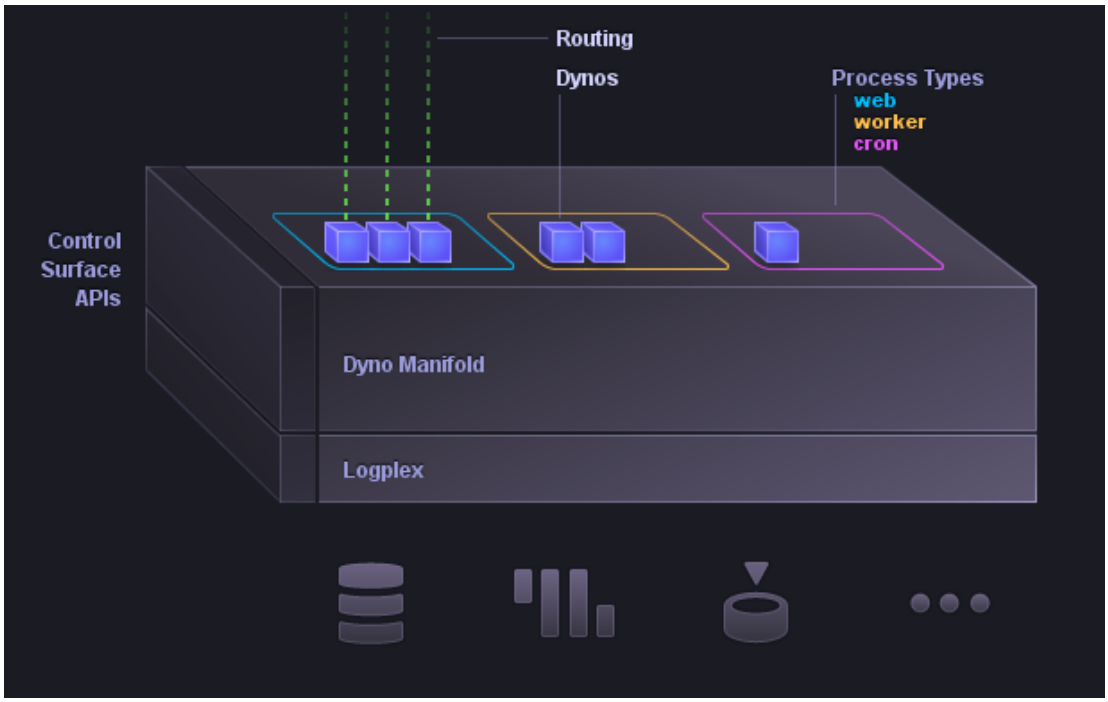

Abbildung 4.1: Heroku Architektur Heroku (2011d)

für eine Anwendung zuständig sind, beide keines falls auf dem gleichen physikalischen Server laufen. So wird die Ausfallsicherheit erhöht.

Prozesstypen Jedem Dyno kann ein bestimmter Prozesstyp zugewiesen werden. Der Web Prozess empfängt HTTP Requests und arbeitet sie ab. Ein Worker Prozess kann im Hintergrund Aufgaben abarbeiten, um zum Beispiel die Latenz zu verringern. Des weiteren kann jede Anwendung weitere Typen definieren.

Routing Das Routing auf Heroku erfolgt auf mehreren Stufen: Ein eingehender HTTP Request geht zuerst in eine Komponente, die zum Beispiel SSL unterstützt. GET Requests werden anschließend an einen Cache weitergeleitet. Sollte im Cache keine Kopie liegen, wird der GET Request, genau wie alle POST Requests, an das Routing Mesh weitergeleitet. Welches die Requests zu einem zuständigen, freien Dyno delegiert.

Logplex Logplex trägt alle Informationen von allen Dynos und anderen Komponenten der Anwendung zusammen und bietet so eine zentrale Schnittstelle zur Überwachung der Anwendung.

Control Surface API Die Control Surface API bietet Möglichkeiten zum Verwalten und Deployment der Anwendung via Komandozeile, Web-Console und REST API.

Add ons Außerdem ist es möglich, schnell und einfach weitere Komponenten zur Anwendung hinzuzufügen. Hierfür bietet Heroku eine Vielzahl von Angeboten, die zum großen Teil kostenlose Konfigurationen bereitstellen.

4.2 Optimierungsmöglichkeiten

Da ein einfaches Deployment in der Cloud keines Falls alle Möglichkeiten des Cloud Computing voll nutzt, soll im folgenden Abschnitt auf die Optimierungsmöglichkeiten bei Cloud-Anwendungen eingegangen werden.

4.2.1 Problem von nicht optimierten Anwendungen

Laut Sessions (2011) weißt die oben beschriebene Portierung in die Cloud einige Optimierungsmöglichkeiten auf. So könnten durch die Verringerung der benötigten Ressourcen für einen hinzukommenden Benutzer erheblich Kosten eingespart werden. Beläuft sich der Unterschied zwischen einer nicht optimierten Anwendung und einer optimierten pro Stunde nur auf 56 Cent, so kann dies auf Jahressicht einen Unterschied von zwei Millionen Dollar ausmachen.

Viele Anbieter von IaaS und PaaS Angeboten bieten eine automatisierte Möglichkeit zur Skalierung an, um die Elastizität der Anwendung zu gewährleisten. Diese Verfahren stoßen allerdings an ihre Grenzen, sobald viele Transaktionen mit einer einzigen Datenbank durchgeführt werden müssen, da sich hier ein Flaschenhals bildet. Als Optimierung kann die Datenbank nach verschiedenen Aspekten aufgeteilt werden.

Ein weiteres Problem einer einzigen Datenbank zeigt sich unter dem Gesichtspunkt der Sicherheit. Bei einer hohen Elastizität einer Anwendung ist es schwierig die Sicherheitskonfiguration der Datenbank stetig aktuell zu halten. Umso keine Sicherheitslücken entstehen zu lassen, kann die gleiche Optimierungsmöglichkeit genutzt werden, wie im oberen Punkt.

Bei der Nutzung einer Plattform für die Anwendung können schnell plattformspezifische Eigenschaften in die Software einbezogen werden, dies führt zu einem Vendor Lock-in. Um diesem Effekt entgegen zu wirken, sollte der Code wenig Abhängigkeiten aufweisen.

4.2.2 Aufteilen von Zuständigkeiten

Als wichtigste Möglichkeit der Optimierung von Cloud-Anwendungen beschreiben Sessions (2011), Dix (2010) und Rosenberg und Matoes (2011) das Aufteilen von Zuständigkeiten auf

weitere kleiner Dienste. Dix (2010) nennt verschiedene Vorgehensweisen bei der Aufteilung der Zuständigkeiten:

Aufteilung nach Änderungsrate Um die Stabilität der Hauptfunktionalität der Anwendung bei der Änderung oder dem Hinzufügen von anderen Funktionen nicht zu gefährden, kann eine Anwendung nach ihrer Änderungsrate in einzelne Dienste aufgeteilt werden. Auch lässt sich so die Entwicklung vom Kernsystem und neuen Funktionen genau trennen.

Aufteilung nach Fachlichkeit Das Aufteilen einer Anwendung nach ihren Fachlichkeiten, lässt Abhängigkeiten im System klar erkennen und diese können so minimiert werden. Auch ist es hier möglich, Optimierungen für jede Funktion individuell vorzunehmen. Nach diesen Änderungen müssen nur Tests für die jeweiligen Entitäten durchgeführt werden. Der Umfang jedes Dienstes umfasst hierbei die Datenhaltung, sowie die Logik.

Aufteilung nach Lese- Schreibfrequenz Entitäten in einem System können sich hinsichtlich ihrer Lese- und Schreibhäufigkeit sehr stark unterscheiden. Um die Implementation dieser Entitäten zu vereinfachen, können sie mit gleichen Eigenschaften zusammengefasst werden. So kann für jede Gruppe die optimale Strategie gefunden und umgesetzt werden.

Aufteilung nach Verbundhäufigkeit Bei einem Verbund zweier Entitäten müssen, wenn sie in getrennten Diensten betrieben werden, mehrere Request abgesetzt werden. Um diese Zahl zu verringern, können Entitäten, die oft verbunden werden in einem Dienst zusammengefasst werden. Wird eine Entität mit zwei verschiedenen oft verbunden, besteht die Möglichkeiten Replikate in beiden Diensten vorzuhalten. Hierbei ist jedoch die Änderungsrate der zu verbindenden Entität zu beachten.

4.2.3 Sharding

Die Aufteilung von Zuständigkeiten auf Datenbankebene wird durch das in Rosenberg und Matoes (2011) beschriebene Sharding ermöglicht. Der Begriff wurde von Google verbreitet, das Konzept dahinter existiert unter dem Begriff shared-nothing Datenbank schon längere Zeit. Es gibt Implementierungen von unter anderem eBay, Amazon und Wikipedia. Rosenberg und Matoes (2011) definiert Sharding wie folgt:

> „A decomposition of a database into multiple smaller units (called shards) that can handle requests individually. It's related to a scalability conecpt called shared-

> nothing that removes dependencies between portions of the application such that they can run completely independently and in parallel for much higher troughput."

Der klassische Weg Datenbanken zu skalieren, ist es größere und schnellere Server bereitzustellen. Eine solche Möglichkeit besteht jedoch nicht bei Anforderungen, wie sie Google oder Facebook aufweisen. Mit Sharding ist es möglich, bei steigender Last linear zu skalieren ohne teure neue Hardware zu kaufen. Ein weiterer Vorteil ist die hohe Verfügbarkeit. Fällt ein Shard aus, können Anfragen von anderen Shards weiter bedient werden. Dieser Effekt kann noch erhöht werden, wenn von jedem Shard Replikate existieren. Außerdem können Anfragen schneller bearbeitet werden, da sie auf einem geringeren Datenbestand ausgeführt werden müssen. Ein Flaschenhals bei vielen Anwendungen ist das Schreiben von Daten. Sind allerdings mehrere Shards vorhanden, kann parallel geschrieben werden. Vergleicht man Sharding mit traditionellen Datenbanken sind folgende Unterschiede zu beachten:

- Daten in einem Shard sind nicht normalisiert. Es werden Daten zusammengespeichert, die zusammen gelesen werden, um die Anzahl an Anfragen zu verringern.
- Die Daten sind über mehrere physische Server verteilt. So ist ein horizontales Skalieren ohne Limit möglich.
- Die Menge an verschiedenen Entitäten auf einem Server wird klein gehalten, wodurch ein effizientes Caching möglich wird und der Aufwand für Back-Ups und Wiederherstellung sinkt.

Partitionierungsschemata

Im folgenden sollen die populärsten Schemata zum Aufteilen der Daten beschrieben werden. Hierbei ist zu beachten, dass das Schema je nach Charakter der Anwendung und Nutzungsmuster ausgewählt werden sollte.

vertikale Partitionierung Bei der vertikalen Partitionierung wird jeder Entität der Anwendung ein eigener Shard zugewiesen. Diese Art der Aufteilung ist einfach zu Implementieren und hat wenig Auswirkungen auf das gesamte System.

intervallbasierte Partitionierung Bei der intervallbasierten Partitionierung wird von jeder Entität eine bestimmte Eigenschaft ausgewählt. Ihr Wert bestimmt auf welchem Shard der Datensatz gespeichert wird. Hierbei ist die Eigenschaft so zu wählen, dass die Shards untereinander möglichst balanciert sind.

key- oder hashbasierte Partitionierung Bei dieser Art der Partitionierung wird aus dem Wert einer Eigenschaft einer Entität mittels einer Hash-Funktion die zugehörige Datenbank errechnet. Ein Problem ist hierbei, dass die Funktion auch eine variable Anzahl an Shards berücksichtigen sollte.

verzeichnisbasierte Partitionierung Die verzeichnisbasierte Partitionierung weißt die loseste Kopplung auf, da hier ein Dienst die Abbildung von Entität auf den Shard verwaltet. So ist es möglich zur Laufzeit den Ort eines speziellen Datensatzes oder die Anzahl an Shards zu ändern. Dieser Vorteil geht allerdings mit einer sehr hohen Komplexität einher.

Herausforderungen

Neben den herausragenden Vorteilen von Sharding, gibt es auch Herausforderungen bei der Implementierung. Soll zum Beispiel der Verbund von zwei Entitäten, die in verschiedenen Shards gespeichert sind, gebildet werden, müssen beide Shards angefragt werden. Auch müssen Fremdschlüsselbeziehungen zwischen Entitäten auf verschiedenen Shards vom Anwendungscode verwaltet werden, was zu einem erhöhten Entwicklungsaufwand führt. Das größte Problem bei Sharding ist das Ausbalancieren von Shards. Ist ein Shard voll, muss er in zwei aufgeteilt werden. Dies muss von Anfang an bedacht werden und Möglichkeiten für die Aufteilung geschaffen werden. Eine solche Funktion ist am einfachsten mit dem verzeichnisbasierten Partitionierung möglich, wobei sich hier auch ein neuer Single Point of Failure bildet. All diese Aufgaben fallen dem Anwendungsprogrammieren zur Last, da es für Sharding noch keine Standardlösung gibt, da die Implementierungen von großen Firmen bisher geheim gehalten werden.

4.2.4 Snowmann Architektur

Sessions (2011) stellt eine mögliche Architektur für cloudoptimierte Anwendungen vor. Mit ihr sollen die Kosten im Vergleich zu einer Portierung, wie in 4.1 beschrieben enorm verringert werden. Hierbei wendet er die beiden Prinzipien *shared instances* und *tight projection* an.

Shared Instances Bei dem Prinzip shared instances, ist jede Anwendungsinstanz für mehrere Benutzer gleichzeitig zuständig. So entstehen gegenüber von nichtgeteilten Instanzen bei gleicher Anzahl zwar die gleichen Kosten, es können aber wesentlich mehr Benutzer bedient werden, da eine Instanz im Normalfall in der Lage ist, die Requests von mehreren Benutzern zu verarbeiten. Als Begründung dafür, dass dieses Modell sich noch nicht durchgesetzt hat, nennt Sessions (2011) den erhöhten Entwicklungsaufwand.

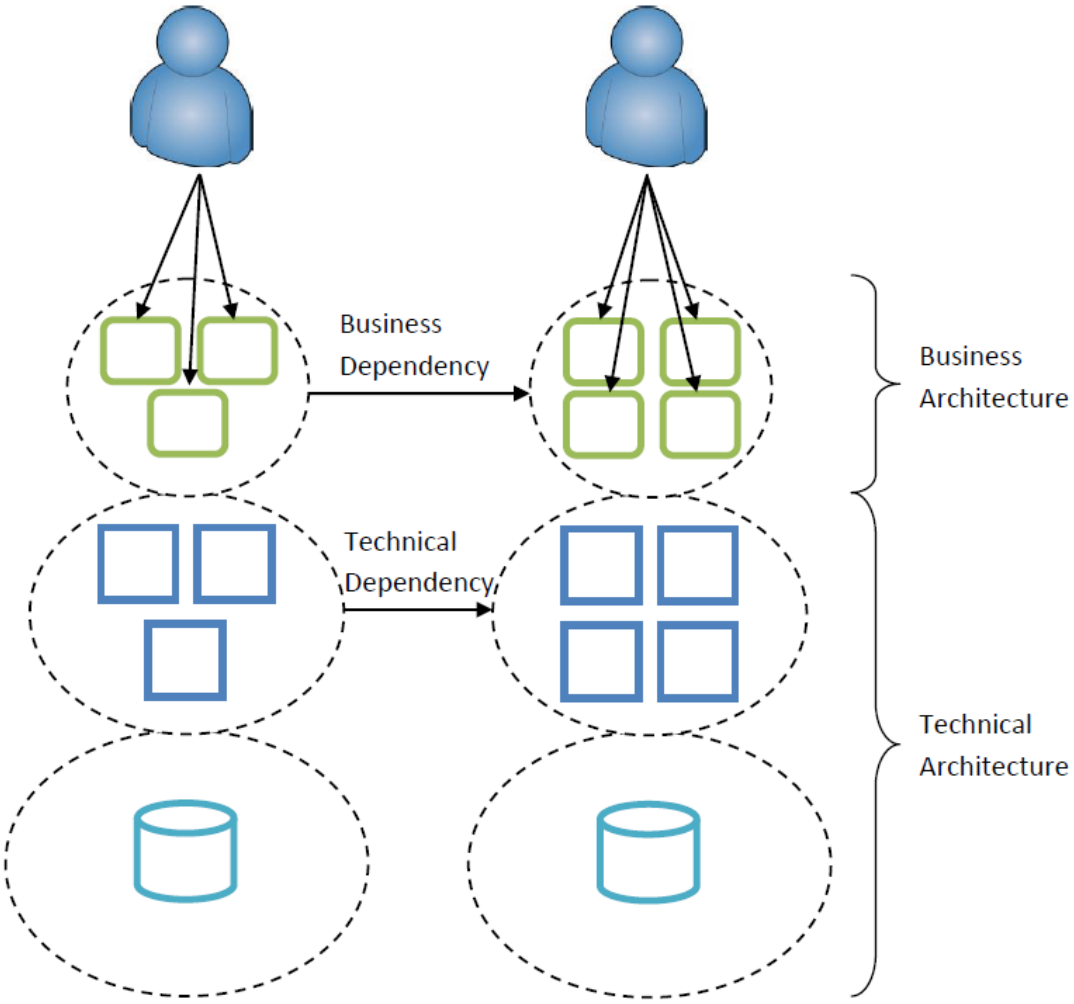

Abbildung 4.2: Snowman Architektur nach Sessions (2011)

Als Bedingung um geteilte Instanzen zu ermöglichen, darf der Zustand nicht in den Nachrichten zwischen Client und Server enthalten sein. Auch müssen die Nachrichten zu Business Transaktionen zusammengefasst werden, um den Verkehr zur Datenbank nicht gering zu halten.

Tight Projection Tight projection beschreibt ein Prinzip, um Ressourcen effizient zu allokieren. Dies hat eine hohe Priorität, da im Cloudumfeld die Ressourcen nach Nutzung abgerechnet werden. Ein Nutzer eines Systems nutzt oft Funktionen hintereinander, die in einem fachlichen Zusammenhang stehen. Dies sollte bei der Aufteilung von Funktionen auf verschiedene Dienste berücksichtigt werden, um eine bessere Ausnutzung der Ressourcen zu ermöglichen. Voraussetzung für optimale Ergebnisse ist, dass das Prinzip auf allen Ebenen der Anwendung durchgesetzt wird.

Aus der Kombination dieser beiden Prinzipien leitet Sessions (2011) die sogenannte Snowman Architektur ab. Getrieben wird diese Architektur von den, in Clustern zusammengefassten, Business Funktionen, die den Kopf des Schneemannes bilden. Die Abhängigkeiten zwischen Funktionen in zwei verschiedenen Clustern werden durch die Business Dependencies dargestellt. Den Oberkörper des Schneemanns bilden die Implementationen der Funktionen. Die Business Dependencies werden eins zu eins auf die technischen Abhängigkeiten abgebildetet. Um dem Prinzip der shared instances gerecht zur werden, erfolgt die Implementation der Funktionen zustandslos. Im unteren Teil des Schneemanns ist die Datenbank angesiedelt. Hierbei hat jedes Cluster seine eigene Datenbank. Ein solches Vorgehen widerspricht dem Ziel alle Daten in einem Unternehmen in einer zentralen Datenbank zu speichern, um den Überblick zu bewahren. Dieses Ziel kann jedoch durch die Dokumentation der Aufteilung in Cluster ebenfalls erzielt werden.

Eine Umsetzung der Architektur bietet gegenüber einer nichtoptimierten Anwendung folgende Vorteile: Die bessere Ausnutzung der Ressourcen bietet eine sehr hohe Kosteneffizienz. Aufgrund einer kleinen Anzahl von Nutzern wird die Überwachung pro Cluster einfacher und erhöht so insgesamt die Sicherheit der Anwendung.

4.2.5 Load Balancing

Load Balancing beschreibt eine Strategie zur Verteilung von Request von einer IP-Adresse auf mehrere Backend-Server. Dies Verteilung bietet die Vorteile, dass auf den Ausfall einzelner Server reagiert werden kann, mehr Request zur Zeit bearbeitet werden können und die Latenz durch eine geringere Auslastung der einzelnen Server erzielt werden kann. Mit Load Balancing ist es möglich eine Anwendung horizontal zu skalieren, das bedeutet, dass mehr Server zur

Verarbeitung von Request zur Verfügung gestellt werden. Beim vertikalen Skalieren werden die Ressourcen der einzelnen Server erhöht. Um zu entscheiden, zu welchem Server ein Request weitergeleitet werden soll, stehen dem Load Balancer verschiedene Algorithmen zur Verfügung. Drei die in Dix (2010) beschrieben werden, sollen auch an dieser Stelle vorgestellt werden:

Round-Robin Im Zentrum des Round-Robin Algorithmus steht eine Liste der aktuell verfügbaren Servern. Sie wird zyklisch durchlaufen und immer der aktuelle Server zur Abarbeitung des Requests gewählt. Ein solches Verfahren eignet sich gut für Anfragen mit konstanten Rechenaufwand. Dies ist aber bei Services selten der Fall, da hier unterschiedlich komplexe Daten angefordert werden können.

Least-Connection Die unterschiedlich benötigte Zeit zur Abarbeitung von Request wird bei dem Least-Connections Algorithmus berücksichtigt. Hier wird zu jedem Server die Anzahl der zurzeit offenen Verbindungen gehalten. Der Request wird dann an den Server weitergeleitet, der die geringste Anzahl hat.

URI-Based Der URI-Based Algorithmus grenzt sich zu den anderen Algorithmen ab, da er zur Entscheidung nicht die Server sondern den Request heranzieht. Hierzu wird aus der URI mittels einer Hash-Funktion eine Abbildung auf einen Server vorgenommen. Dieser Algorithmus entwickelt seinen Vorteil erst, wenn er zusammen mit einer Caching-Lösung eingesetzt wird, da nun nicht jeder Server alle Objekte cachen muss, sondern nur die, für die er zuständig ist.

4.2.6 Erreichen von Elastizität

Mit Hilfe des Load Balancing können zwar Requests auf verschiedene Dienste oder Cluster von Funktionen aufgeteilt werden, doch kann die Anzahl der Server nicht erhöht werden. Dies ist aber bei der Cloud-Anwendung erforderlich, um die Eigenschaft der Elastizität der Anwendung zu erreichen. Um den Zeitpunkt der Skalierung festlegen zu können, müssen zunächst Indikatoren bestimmt werden. Chieu u. a. (2009) nennt als mögliche Indikatoren für Web Anwendungen folgende:

- Anzahl der parallelen Nutzer
- Anzahl der aktiven Verbindungen
- Anzahl der Requests pro Sekunde
- Durchschnittliche Antwortzeit

Ist einer der Indikatoren ausgewählt, sollte er in der bereits laufenden Anwendung beobachtet werden, um einen tatsächlichen Wert für den Indikator zu finden. Es sollte jedoch nicht nur ein oberer Schwellenwert für das Hinzufügen von neuen Servern gefunden werden, sondern ebenso ein unteren, um auf einen möglichen Rückgang der Last reagieren zu können. Zur Realisierung der Elastizität setzte Chieu u. a. (2009) eine Service Monitor Komponente ein, die von jeder Instanz der Anwendung in einem gewissen zeitlichen Abstand den Indikator abfragte und auf dieser Grundlage die Entscheidung trifft, inwieweit an der Anzahl der Instanzen eine Änderungen vorgenommen werden muss. Eine Voraussetzung für diese Implementation ist es, dass der Load Balancer ein dynamisches Hinzufügen oder Entfernen von Instanzen unterstützt. Eine Möglichkeit der Umsetzung ist das Aktualisieren des Config-Datei. Des weiteren muss eine Schnittstelle zum Anbieter der Instanzen bestehen, um die Verwaltung zu realisieren. Zur Vereinfachung des Vorgang des Hinzufügens von Instanzen, sollten diese einfach zu Konfigurieren und schnell zu starten sein. Im nachfolgenden Algorithmus nach Chieu u. a. (2009) wird die Anzahl aktiver Sitzungen in einer Instanz als Indikator verwendet:

Sei:
A_i : Anzahl der aktiven Sitzungen in Instanz i
S_{max} : maximale Anzahl an Sitzungen pro Instanz (zum Beispiel 40.000)
T_{upper} : oberer Sitzungsschwellenwert (zum Beispiel 80%)
N_{lower} : unterer Sitzungsschwellenwert (zum Beispiel 60%)
$N_{instance}$: Anzahl existierender Instanzen
N_{exceed} : Anzahl Instanzen über dem oberen Sitzungsschwellenwert
N_{below} : Anzahl Instanzen unter dem unteren Sitzungschwellenwert
for $i = 0 \rightarrow N_{instance}$ **do**
  if $A_i/S_{max} \geq T_{upper}$ **then**
    $N_{exceed}++$
  end if
  if $A_i/S_{max} < T_{lower}$ **then**
    $N_{below}++$
  end if
  Sortiere alle A_i in J aufsteigend nach $\frac{A_i}{S_{max}}$ ein
end for
if $N_{exceed} == N_{instance}$ **then**
  Erzeugen und Starten einer neuen Instanz
  $N_{instance}++$
  Neue Instanz Load Balancer mitteilen

end if
if $N_{below} \geq 2$ **then**
$m =$ erster Index in J
if $A_m == 0$ **then**
Instanz m vom Load Balancer entfernen
Instanz m ausschalten
$N_{instance} - -$
$N_{below} - -$
entferne Index m von J
if $N_{below} \geq 2$ **then**
$n =$ erster Index in J
entferne Instanz n temporär
$N_{instance} - -$
$L_n = 0$ normalisierter Load Faktor
end if
end if
end if
for $i = 0 \rightarrow N_{instance}$ **do**
Berechnung der normalisierten Load Faktoren:
$L_i = (1 - \frac{A_i}{S_{max}}) / \sum_{k=1}^{N_{instance}} 1 - \frac{A_k}{S_{max}}$
L_i dem Load Balancer mitteilen
end for

Zu Beginn des Algorithmus wird über die im Betrieb befindlichen Instanzen iteriert und für jede berechnet, ob sie über dem oberen Schwellenwert oder unter dem unteren liegt. Trifft eines hiervon zu, wird der jeweilige Zähler erhöht. Zusätzlich werden alle Werte aufsteigend in eine Liste einsortiert.

Ist die Anzahl der Instanzen über dem oberen Schwellenwert gleich dem der existierenden Instanzen, wird eine neue Instanz gestartet, der Zähler der Instanzen erhöht und die neue Instanz dem Load Balancer mitgeteilt.

Hat der Zähler der Instanzen unterhalb des unterer Schwellenwert einen Wert kleiner oder gleich Zwei, wird das erste Element auf der Liste der Instanzen in m gespeichert, da diese Instanz die geringste Auslastung aufweist. Ist die Anzahl der aktiven Sitzungen gleich Null, wird diese Instanz vom Load Balancer entfernt und anschließend herunter gefahren. Zusätzlich wird die Anzahl der aktiven Instanzen dekrementiert und m aus der Liste entfernt. Ist der Zähler für die Instanzen unter dem unteren Schwellenwert immer noch kleiner oder gleich 2,

wird die nächste Instanz in der Liste temporär entfernt und der Load Faktor für diese Instanz gleich Null gesetzt.

Am Ende des Algorithmus wird für jede Instanz ein neuer Load Faktor berechnet und dem Load Balancer mitgeteilt, um ein optimiertes Load Balancing zu gewährleisten.

4.3 Technische Architektur einer optimierten Cloud-Anwendung

Fasst man die bisher beschriebenen Optimierungsmöglichkeiten zusammen, lassen sich zwei unterschiedliche technische Architekturen für die Anwendung erarbeiten:

- Mit dem Einsatz von Sharding und
- der Anwendung der Snowman-Architektur.

Beide Möglichkeiten sollen im folgenden für diese Anwendung entwickelt werden.

4.3.1 Einsatz von Sharding

Neben den fachlichen Komponenten, muss die Architektur für die optimierte Anwendung mit dem Einsatz von Sharding um zwei weitere Komponenten erweitert werden:

- Einem Load Balancer und
- einer Monitor Komponente,

die den Status der Serverinstanzen überwacht und auf Laständerungen reagiert und diese dem Load Balancer mitteilt. Die fachlichen Komponenten befinden sich auf den Serverinstanzen, welche zusätzlich eine Möglichkeit unterstützten muss, den jeweiligen Indikator zur Skalierung dem Monitor mitzuteilen. Der Einsatz von Sharding fällt bei dieser Anwendung sehr leicht, da die Trennung der zu speichernden Daten schon auf Anwendungsebene geregelt ist. So soll die globale Userentität und Zuordnung in den einzelnen Netzwerken in getrennten Datenbanken gespeichert werden. Ein Nachteil dieser Architektur liegt in dem Single-Point-of-Failure bei dem Load Balancer, sowie bei der Monitor Komponente.

4.3.2 Anwendung der Snowman-Architektur

Auch die Anwendung der Snowman-Architektur setzt einen weiteren Load Balancer, sowie eine Monitor Komponente voraus, wobei diese sich hier komplizierter gestalten. Neben der

Aufteilung der Last auf verschiedene Server, muss hier auch zusätzlich die Auswahl auf Grundlage der jeweiligen Zuständigkeiten getroffen werden. Das betrifft nicht nur den Load Balancer, sonder auch den Monitor, da dieser auf Änderungen der jeweiligen Zuständigkeit gesondert reagieren muss. Vorstellbar ist hier, je Zuständigkeit einen gesonderten Monitor zu betreiben. Die Aufteilung der Zuständigkeiten soll bei dieser Anwendung nach Lese- und Schreibzugriff erfolgen, da so die optimale Strategie für die jeweilige Zugriffsform eingesetzt werden kann. Eine Aufteilung nach Netzwerken wurde nicht gewählt, da bei dieser Anwendung besonders Wert auf die Aggregation von Informationen aus verschiedenen Netzwerken gelegt wird und dies durch die Aufteilung der Netzwerke auf verschiedene Instanzen zusätzlich erschwert wird.

5 Realisierung der Anwendung

„Ruby - A Programmer's Best Friend“

Ruby (2011)

In diesem Kapitel wird die Realisierung der bisher konzipierten Anwendung beschrieben. Zuerst soll die Umsetzung der Komponenten beschrieben werden,um anschließend das Deployment auf der Heroku Plattform, sowie die Umsetzung mit Sharding und der Snowman-Architektur zu beschrieben. Am Ende des Kapitels werden die eingesetzten Testverfahren dargestellt.

5.1 Komponenten

Als Grundlage der verschiedenen Portierungsarten dienen jeweils die fachlichen Komponenten. Die Realisierung dieser soll im Folgenden beschrieben werden.

5.1.1 Frontend

Um das Frontend der Anwendung zu realisieren, wurde das Sinatra Framework Sinatra (2011) zusammen mit der Markuplanguage Haml Haml (2011) verwendet. Die implementierten Controller verwenden hierbei Sinatra, um HTTP Request entgegen zu nehmen. Haml dient der Erzeugung des Response in Form von HTML. Die Zuordnung von Objekten aus der SocialObjects Komponente zu ihren entsprechenden HAML Views wurde über den Klassennamen realisiert. Neben der Möglichkeit alle Objekte aus den Netzwerken anzuzeigen, wurde zusätzlich ein View implementiert, in dem die aktuellen Posts aus den Netzwerken spaltenweise nebeneinander dargestellt werden. Zusätzlich gibt es dort auch die Möglichkeit Posts in Netzwerke zu schreiben. Um den Code hierfür übersichtlich zu gestalten, wird in diesen Spalten jeweils das Template für das jeweilige Objekt verwendet.

5.1.2 SocialObjects

Das Hauptaugenmerk bei der Realisierung der SocialObjects Komponente lag auf einer möglichst hohen Abstraktion und Erweiterbarkeit. Um zu ermöglichen, dass neue Netzwer-

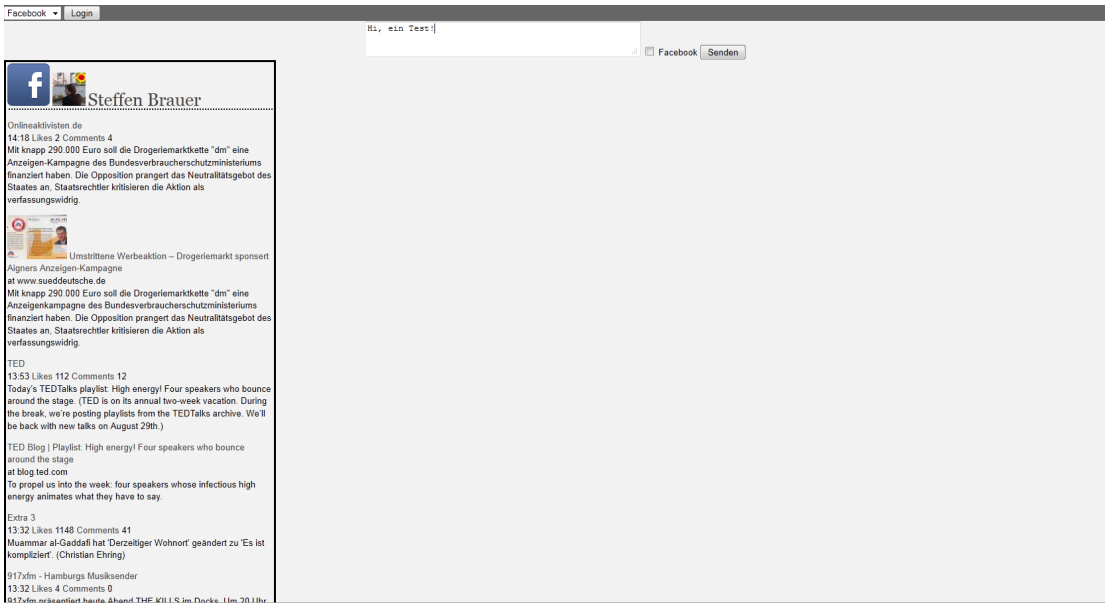

Abbildung 5.1: Screenshot der Anwendung

ke zur Laufzeit hinzugefügt werden können, enthält die Klasse eine Map um die Namen des jeweiligen Netzwerks auf die Klasse abzubilden, die dieses implementiert. Mithilfe der `add_network name, classname` Methode ist es so möglich ein Netzwerk hinzuzufügen.

Neben der Verwaltung der Netzwerke enthält die SocialObjects Komponente die Connection Klasse. Ihr Aufgabe ist es, Verbindungen zwischen Objekten Abzubilden. Hierzu enthält die Klasse den Namen der Verbindungen, sowie einen Array, der die Objekte enthält. Um den Umgang mit der Klasse zu vereinfachen implementiert sie die `each` Methode. Zusätzlich wird das Modul Enumerable Doc (2011) inkludiert, womit alle Methoden hieraus nutzbar sind. Dies vereinfacht die spätere Verarbeitung der Verbindungen, da dieses Modul viele sehr praktische Methoden enthält.

Um die Abbildung von Nutzer auf die Accounts in den einzelnen Netzwerken vorzunehmen, enthält die SocialObjects Komponente die User Klasse. Sie enthält nur eine Id. Um diese Ids persistent zu speichern, wurde DataMapper DataMapper (2011) verwendet.

5.1.3 Implementierung einer Plattform

Die Implementierung einer Plattform innerhalb der SocialObjects Komponente umfasst die folgenden Aufgaben:

- Zuordnung der globalen Nutzer zu den Netzwerk spezifischen Accounts

- Authentifizierung des Accounts am Netzwerk
- Abrufen von Objekten aus dem Netzwerk
- Schreiben von Objekten in das Netzwerk

Um wiederrum die Kopplung möglichst gering zu halten und so die Möglichkeit zur Erweiterbarkeit zu erhöhen, müssen all diese Aufgaben durch eine Schnittstelle implementiert werden. Um trotzdem dem Prinzip Seperation of Concerns nachzukommen, soll diese Klasse nur Container weiterer Module dienen, welche die Implementation der einzelnen Aufgaben beinhalten. Als Beispiel für die Implementation der Plattform soll Facebook herangezogen werden.

Zuordnung der Nutzer zu den Accounts

Diese Aufgaben werden den einzelnen Netzwerkkomponente zugeschrieben, da so nicht bei jeder Änderung der Implementierung oder beim Hinzufügen eines Netzwerks die Attribute des globalen Nutzers geändert werden müssen. Um diese Aufgabe zu erfüllen, ist es nötig eine Entität mit folgenden Attributen zu erstellen:

user_id Primärschlüssel des globalen Nutzers

facebook_id In diesem Fall der Primärschlüssel des Accounts im Facebook Netzwerk

access_token Dieser Token wird zur Authentifizierung des Accounts bei Facebook benötigt.

Um die Konsistenz der Anwendung nicht zu gefährden, sollte zwar jede Plattform ihre eigenen Entitäten zur Zuordnung vorhalten, aber es sollte die selbe Datenbank zur Speicherung genutzt werden. Die Datenbankanbindung in dieser Anwendung basiert ebenfalls auf DataMapper.

Authentifizierung

Die Facebook Graph API verwendet das in 3.3.2 erwähnte oauth 2.0 Protokoll. Um die Komplexität der Implementierung dieser Aufgabe zu verringern wurde zur Realisierung eine Bibliothek verwendet. Diese kapselt das Protokoll in einer Klasse und ermöglicht so die Konzentration auf den fachlichen Code. Zu finden ist die Bibliothek unter Github (2011b). Wie in 5.1.3 erläutert, soll die Schnittstelle zwischen SocialObjects Komponente und Plattform nur eine Klasse umfassen. Da aber auch Seperation of Concerns berücksichtigt werden soll, wurde die Bibliothek in einem Ruby Module gekapselt, welches in die Schnittstellen Klasse `Facebook` integriert wird. Neben der Kapselung der oauth 2.0 Bilibothek ist eine weitere Aufgabe des Modules die Verwendung der Entität zur Zuordnung der Nutzer. Dies ist hier einfach möglich, da hier der Zugriff auf alle nötigen Informationen gegeben ist.

Abrufen von Objekten

Um die Delegation von Plattformschnittstelle zum konkreten Objekt zu leisten, wird das selbe Verfahren angewendet wie in 3.2.4 beschrieben, nur das nicht auf Plattformen, sondern auf die Objekte abgebildet wird. Der Zugriff auf Graph API erfolgt dann im Konstruktor des Objektes. Hier werden die Daten abgerufen und den jeweiligen Instanzvariablen zu gewiesen. Die Objekte haben neben dem Halten der Informationen noch zusätzlich die Aufgabe, die Verbindungen zwischen den Objekten zu erstellen. Hierfür werden die angeforderten JSON Listen durchlaufen und pro Objekt in der Liste ein neues Objekt im System erstellt. Der Code im folgenden Beispiel soll zeigen, wie die Freunde eines Nutzers in Facebook abgerufen werden:

```
1 json = JSON.parse(@access_token.get("/#{id}/friends"))
2 conn = json["data"].inject(Array.new)
3         {|arr,obj| arr << FbUser.new(obj)}
4 return Connection.new(:friends,conn)
```

Schreiben von Objekten

Das Schreiben von Objekten folgt dem Verfahren, das in 3.2.4 beschrieben wurde. Auch hier wird die Abbildung von Typ auf Klasse genutzt, um für das übergebene Objekt die passende Plattformimplementation zu finden. Dieses Objekt wird dann an die entsprechende Klasse übergeben, welche dieses dann über die API in das Netzwerk schreibt.

5.2 Varianten

Nach dem nun die fachlichen Komponenten realisiert wurden, sollen sie im nächsten Schritt zu einer Cloud Anwendung portiert werden. Dies geschieht in den in 4 vorgestellten Varianten.

5.2.1 Heroku

Um die Anwendung auf Heroku zu deployen, muss zunächst das Heroku Gem Github (2011a) installiert werden. Dieses bietet via Konsole Zugriff auf alle benötigten Funktionen der Heroku Plattform. Um den Quellcode auf der Plattform zu verwalten, wird jeder Heroku Anwendung ein Git-Repository zugewiesen. Um den Code von dem Entwicklungsrechner zur Plattform hochzuladen, muss zunächst lokal ein Git-Repository initialisiert werden.

```
1 git init
2 git add .
```

```
3 git commit -m "initial␣commit"
```

Mithilfe des Heroku Gems kann nun eine neue Anwendung auf Heroku erzeugt werden und mit dem Git-Befehl für das Hochladen zu einem globalen Repository zu Heroku geschickt werden.

```
1 heroku create "socialobjects"
2 git push heroku
```

Hier werden alle Abhängigkeiten installiert und die Anwendung gestartet. Über das Heroku Gem ist es nun möglich die aktuellen Logs der Anwendung auszulesen, die Anzahl der Dynos zu erhöhen oder andere Entwickler zur Anwendung hinzuzufügen.

Ein Hindernis formierte sich in der Nutzung verschiedener Datenbanken in der Entwicklung und Produktion. Für die Entwicklung wurde eine Sqlite Datenbank benutzt, bei Heroku ist nur die Nutzung von PostgreSQL kostenlos möglich. Dies stellte sich insofern als Problem dar, da die Länge der von Facebook verwendeten IDs zwar in den Integer Datentyp der Sqlite Datenbank passte, es aber bei der PostgreSQL Datenbank zu Fehlern kam.

Als weitere Herausforderung stellte sich der Zugriff auf die Graph API von Facebook dar. Hier mussten für Entwicklung und Produktion zwei verschiedene Anwendungen erstellt werden, um die verschiedenen URLs für den Callback des oauth 2.0 Protokolls zu verwalten. Auch mussten für das von Facebook verwendete HTTPS Protokoll die Zertifikate erst nachgeladen werden.

5.2.2 Sharding

Da die Möglichkeit der Optimierung der Anwendung auf der Heroku Plattform sehr gering ist, wurden die optimierten Anwendungsvarianten auf lokalen Rechner getestet, um vollen Zugriff auf die verwendeten Systeme zu haben.

Load Balancer Als Load Balancer Komponente wurde in dieser Variante HAProxy eingesetzt. Hierbei handelt es sich um einen Open Source Projekt, das bei vielen populären Unternehmen im Einsatz ist HAProxy (2011). Weitere Gründe für die Wahl waren die einfache Konfiguration, die Vielzahl an unterstützen Algorithmen, sowie die Möglichkeit, die Konfiguration ohne Neustart des Programms zu ändern, da nur so neue Instanzen hinzugefügt und entfernt werden können.

Monitor Die Monitor Komponente ruft den Indikatorwert von den zurzeit laufenden Instanzen ab und trifft die Entscheidung, ob eine Instanz, mithilfe einer abgeänderten Version

des in 4.2.6 beschriebenen Algorithmus, hinzugefügt oder entfernt werden soll. In dieser Implementation werden den Instanzen keine Lastfaktoren mitgeteilt.

Die Kommunikation zwischen dem Monitor und dem Load Balancer erfolgt über Kommandozeilenaufrufe aus dem Monitor. Um die Konfigurationsdatei zu ändern, liest sie der Monitor ein und führt eventuelle Änderungen daran durch. Nur durch die Konfigurationsdatei kann der Monitor erfahren, von welchen Instanzen er die Indikatorwerte abrufen soll. Um dies durchzuführen, wird ein HTTP Aufruf verwendet, der ein JSON Objekt mit dem Wert zurückliefert. Das Starten von neuen Instanzen erfolgt im Monitor durch den Kommandozeilenaufruf zum Start eines neuen Web-Servers. Um eine Instanz herunterzufahren, wird zurzeit ein HTTP Aufruf verwendet, der im Web-Server ein Herunterfahren zur Folge hat. Bei der Verwendung des Amazon IaaS Angebots EC2, würde das Verwalten der Instanzen über eine Schnittstelle mit Amazon geschehen.

Aufteilung mit DataMapper Als Sharding Strategie wurde die vertikale Partitionierung gewählt. Es gibt je eine Datenbank für die globalen Nutzer und für jedes Netzwerk eine weitere. Dies hat neben dem Performanzgewinn noch eine weitere Unabhängigkeit der Netzwerke vom Rest der Anwendung zur Folge. In der jetzigen Implementierung gibt es also einen Standard-Shard, den DataMapper vorgibt, einen Shard für die globalen Benutzer und einen Shard für das Mapping von globalen Nutzen auf die Facebook Accounts.

Die Realisierung dieser Aufteilung stellte sich als sehr einfach heraus, da DataMapper eine vertikale Partitionierung unterstützt. So muss nur beim Erzeugen der Verbindung zur Datenbank ein Name übergeben werden und in der jeweiligen Klasse die Methode Repository überschrieben werden. Zur besseren Verwaltung, welche Entität in welchem Shard abgelegt werden soll, wurde eine Map eingeführt, die eine Abbildung von Klasse auf den Namen der Verbindung enthält.

5.2.3 Aufteilung nach Zuständigkeit

Als Verfahren zur Aufteilung der Zuständigkeiten, wurde bei dieser Variante der Portierung in die Cloud zwischen Lese- und Schreibzugriff aufgeteilt. Dies entspricht bei dem eingesetzten HTTP Protokoll der Unterscheidung zwischen GET und POST Methoden. Neben dieser Trennung der Zuständigkeiten wurde der Zugriff auf statische Dateien, wie Bilder oder CSS Style Sheets von den Anwendungsservern getrennt.

Anpassung von Load Balancer und Monitor HAProxy ermöglicht eine Abfrage nach der Methode des eingehenden Request. So werden nun zwei Listen von Servern in der Konfi-

gurationsdatei gehalten. Eine Liste mit Instanzen, die für die Verarbeitung von GET zuständig ist und eine für das Bearbeiten von POST Requests. Um die Trennung der Anwendungsserver von den Dateiservern vorzunehmen, wurde ein URL-basierter Algorithmus bei den, für GET Methoden zuständigen Instanzen eingesetzt. Eine Anpassung des Monitors war insofern nötig, da nun zwei unterschiedliche Listen verwaltet werden müssen. Um hier Aufwand zu sparen und die Wartbarkeit zu erhöhen könnten zwei Monitore eingesetzt werden.

Aufteilung der fachlichen Komponenten Um das Aufteilen der Zuständigkeiten auch auf Komponentenebene durchzuführen, wurden die fachlichen Komponenten je nach GET und POST Methoden aufgeteilt. Die Aufteilung betrifft hierbei nur die Objekte, die Verfahren zum Auffinden oder Schreiben von Objekten in die jeweilige Plattform bleiben die gleichen.

5.3 Tests

In diesem Abschnitt soll auf die eingesetzten Testframeworks eingegangen werden. Um die Unit Tests durchzuführen wurde RSpec, für die Anwendungsfälle Cucumber und für Lasttests JMeter verwendet.

5.3.1 Unit Tests

RSpec folgt dem Ansatz des Behavior Driven Development. Dieser unterscheidet sich zum Test Driven Development nur insofern, dass die Funktionen nicht getestet, sondern im Vorhinein beschrieben werden. Hierzu versucht RSpec sich möglichst nah an der natürlichen englischen Sprache zu orientieren. Zur Veranschaulichung der Syntax soll die Beschreibung der Post Klasse dienen:

```
1  require './helper_spec.rb'
2  require 'post'
3
4  describe Post do
5
6    it "should␣has␣attributes" do
7      post = Post.new("me","you","hi")
8      post.from.should eql "me"
9      post.to.should eql "you"
10     post.message.should eql "hi"
11   end
12
```

```
13   it "should␣check␣contraints␣by␣initialization" do
14     Post.add_constraint lambda {|from,to,message|
15                           raise "Too␣short" if message.size < 1}
16     lambda{ Post.new("me","you","")}.should raise_error
17   end
18
19 end
```

Ein Problem beim Schreiben der Spezifikationen entstand durch den Umstand, dass viele Klassen der Facebook Plattform auf die Schnittstelle zugreifen. Um die Durchführbarkeit der Tests auch offline zu gewährleisten, wurde die Bibliothek, die den Zugriff auf Facebook realisiert in den Specs durch ein Mock Objekt ersetzt. Für solche Objekte können in RSpec Methoden und ihre Rückgabewerte zugeordnet werden. Zusätzlich kann die Anzahl der Aufrufe, die stattfinden sollen definiert werden. So war es möglich, mit vorher erstellen JSON Objekten das Parsen der Objekte zu überprüfen. Neben dem Abrufen von Objekten war es auf diese Weise auch möglich das Schreiben von Objekten zu testen, indem die übergebenen Parameter überprüft wurden. Insgesamt war der Einsatz von RSpec eine gute Wahl, da durch die fast natürlich sprachliche Syntax das Schreiben der Tests sehr vereinfacht wurde.

5.3.2 Anwendungsfälle

Zum Test der in 3.1 beschriebenen Anwendungsfälle wurde das BDD Framework Cucumber eingesetzt Cucumber (2011). Cucumber setzt auf RSpec auf und orientiert sich noch mehr an natürlicher Sprache. In der Praxis, wie auch hier, wird es zum Testen von Views eingesetzt. Hierzu wird ein Testframework für Webseiten innerhalb von Cucumber benutzt. Folgendes Listing zeigt den Test für das Anzeigen von Profilen und der Freundesliste.

```
1  Feature: Friendships
2    In order to explore some friendships
3    As a user
4    I want to view profils and friendlists
5
6    Scenario: I watch a profile
7      Given I am logged in
8      And I visit '/socialobject/'
9      When I follow 'Steffen␣Brauer'
10     Then I should see 'Profil'
11     And I should see 'Steffen␣Brauer'
12
```

```
13  Scenario: I watch my friendlist
14    Given I am logged in
15    And I visit
16      '/socialobject/facebook/profil/100001008073688'
17    When I follow 'Freunde'
18    Then I should see 'Nico La'
```

5.3.3 Lasttests

Lasttests können zur Festlegung von Indikatorwerten dienen, um zum einen zu entscheiden, wann neue Ressourcen benötigt werden und zum anderen, um die Elastizität der Anwendung zu testen. In dieser Arbeit wurde aufgrund von mangelnder Zeit nur die erste Variante durchgeführt. Zur Umsetzung der Tests wurde JMeter eingesetzt. JMeter ist eine Java Desktop Anwendung, die ermöglicht, einen in einer GUI erstellten Testplan auszuführen und die Ergebnisse im Nachhinein auf verschiedene Weise zu Visualisieren JMeter (2011). Neben Performanz Tests für HTTP Server ist es hiermit auch möglich Tests für Mail oder Datenbank Server durchzuführen. Bei der Durchführung von Lasttests zeigt sich ein Vorteil von der Durchführung von Tests in der Cloud. Um möglichst viel Last zu simulieren, müssen keine teuren Server angeschafft werden, sondern nur einige Instanz erstellt werden.

Testaufbau und Durchführung

Um die Last mehrerer Nutzer zu simulieren, wurde JMeter auf Amazon EC2 Instanzen eingesetzt. Ein Master Server war notwendig, da eine verteilte Nutzung von JMeter nur innerhalb eines Subnetzes möglich ist. Auf den Master Server wurde via SSH und SFTP zugegriffen. Die Kommunikation zwischen den verschiedenen JMeter Instanzen verlief über Java RMI. Als Testplan wurde der Zugriff auf die Startseite der Anwendung auf Heroku verwendet, da eine Simulation des Anmeldevorgangs innerhalb von JMeter nur unter sehr hohem Aufwand zu realisieren schien. Eine Übersicht über den Testaufbau liefert Abbildung 5.2. Die Durchführung des Tests umfasste die folgenden Schritte:

- Zunächst wurde auf dem Kontrollrechner mit JMeter ein Testplan erstellt und abgespeichert. In diesem wurde neben dem Aufruf der Startseite auch eine Option gewählt, die die Ergebnisse in eine extrene Datei schreibt, um die Auswertung auf dem Kontrollrechner zu ermöglichen.
- Im nächsten Schritt wurden mithilfe der AWS Management Console vier EC2 Instanzen mit einem Linux Betriebssystem erzeugt und gestartet. Zusätzlich war es nötig eine

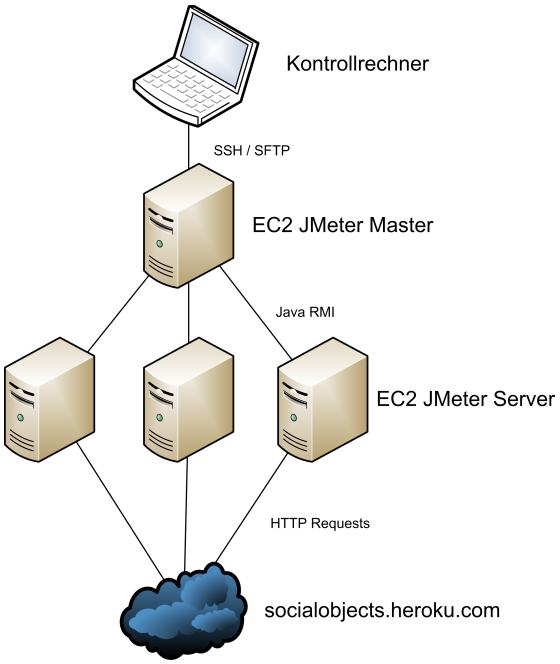

Abbildung 5.2: Aufbau der Testumgebung

sogenannte Security Group zu definieren. Hierbei handelt es sich um eine Art Firewall, die verschiedenen Instanzen zugeordnet werden kann. Aufgrund der Nutzung von RMI innerhalb von JMeter auf unterschiedlichen Ports musste neben den SSH und HTTP Ports auch eine große Anzahl an weiteren TCP Ports freigegeben werden.

- Nachdem eine SSH Verbindung zu allen Instanzen hergestellt wurde, konnte auf jeder Instanz JMeter heruntergeladen werden. Um den Austausch des Testplans und der Datei mit den Ergebnissen zu gewährleisten, wurde eine SFTP Sitzung zwischen dem Kontrollrechner und dem JMeter Master eröffnet.
- Jetzt wurden die drei JMeter Server gestartet und der Testplan zum JMeter Master übertragen. Um die Verbindung zwischen dem Master und den Servern herzustellen, mussten die IP Adressen der Server in der JMeter.properties Datei vermerkt werden. Der Start des Testlaufs erfolgte durch das Ausführen von JMeter auf dem Master Server.
- Nach Beendigung des Laufs wurde die Datei mit den Ergebnissen mittels SFTP zurück zum Kontrollrechner übertragen und alle EC2 Instanzen gestoppt.
- Die Ergebnisdatei konnte nun auf dem Kontrollrechner mithilfe der von JMeter mitgelieferten Möglichkeiten ausgewertet werden.

Testergebnisse

Mit dem oben vorgestellten Verfahren wurden insgesamt vier Testläufe mit verschiedenen Konfigurationen durchgeführt. Der erste Testlauf diente zur Erzeugung eines Idealwertes. Hier wurde mit einer EC2 Instanz mit zehn Benutzern je 40 mal auf die Startseite zugegriffen. Die durchschnittliche Latenz lag bei 121 Millisekunden. Von diesem Startwert aus wurde die Anzahl der Instanzen auf drei erhöht und die Anzahl der Benutzer gesteigert. Hierbei musste die Anzahl der Aufrufe zurückgenommen werden, um in JMeter alle getätigten Aufrufe übersichtlich darzustellen.

Als Ergebnis der Testläufe lässt sich zusammenfassen, dass zwar mit Erhöhung der Last neben der Latenz pro Aufruf auch die Abweichungen zwischen den Aufrufen stieg, es aber nicht gelang, mit dem hier verwendeten Aufbau, den Dyno zum Absturz zu bringen. Die allgemein geringe Latenz lässt sich mit dem wenig rechenintensiven Aufruf der Startseite erklären. Bei Verwendung der Schnittstellen der Netzwerke dürfte die Latenz schnell über eine Sekunde hinaus gehen.

6 Zusammenfassung und Ausblick

Um die Arbeit abzuschließen sollen nun noch ein Mal die Ergebnisse zusammengefasst und ein Ausblick auf mögliche weitere Entwicklungen der Anwendung erstellt werden.

6.1 Zusammenfassung

Cloud Computing beschreibt das zur Verfügung stellen von IT-Ressourcen über das Internet. Der Begriff der Cloud soll hierbei verdeutlichen, dass Ort und Beschaffenheit der Ressourcen nicht bekannt sind. Neben dieser Verschleierung der Ressourcen ist die Elastizität eine weitere wichtige Eigenschaft des Cloud Computing. Es soll also möglich sein, auf Laständerungen zu reagieren, indem neue Ressourcen hinzugefügt und später wieder entfernt werden können. Eine weitere Schlüsseleigenschaft des Cloud Computing ist die nutzungsabhängige Abrechnung der benutzten Ressourcen. Übliche Einheiten sind hierbei CPU pro Stunde und Giga Byte pro Monat. Eine Aufteilung der Cloud kann hierbei in organisatorische Einheiten vorgenommen werden: Public, Private und Hybird Cloud. Eine weitere mögliche Aufteilung stützt sich auf die technische Abstraktion, so gibt es Infrastucture as a Service - hier werden Serverinstanzen oder Speicher angeboten, Plattform as a Service - wobei ganze Anwendungsplattformen bereitgestellt werden und Software as a Service - welches ganze Softwareprodukte über das Internet zugänglich macht. Eine der größten Herausforderungen des Cloud Computing ist der Sicherheitsaspekt. Diesbezüglich gibt es verschiedene Standpunkte von einer Zunahme der Sicherheit durch hochmoderne Rechenzentren, bis hin zur Ausschließung von Cloud Computing für manche Branchen mit besonders schützenswerten Daten. Durch seine Eigenschaften bietet sich der REST Architekturstil zur Realisierung von Cloud Anwendungen, gerade durch die oft angetroffene Verknüpfung von REST und dem HTTP Protokoll an.

Soziale Netzwerke gewinnen immer mehr an Bedeutung. Um jedoch einen tieferen Einblick zu gewinnen, muss man sich die abstrakten Funktionen von sozialen Netzwerken ansehen. Als wichtigste Funktion ist der gemeinsame Austausch zu erwähnen. Eine weitere Funktion eines Netzwerks ist es, Nutzer über die aktuellen Aktivitäten von anderen Nutzern zu informieren, der so genannten Netzwerkawareness. Alle sozialen Netzwerke haben ein Profil eines Nutzers gemein, über das es möglich ist, Identitätsmanagement zu betreiben. Auf Grundlage der so

gewonnen Daten bieten zahlreiche Netzwerke eine Suche von Objekten in den Netzwerken an. Um Benutzer langfristig an ein Netzwerk zu binden, wird Vertrauen aufgebaut, da dies die Grundlage für menschliche Beziehungen darstellt. Als eine praktische Funktion sozialer Netzwerke kann das Kontaktmanagement angesehen werden, das heißt, dass jeder Nutzer selbst seine Kontaktdaten aktuell hält und so ein immer aktuelles Adressbuch bereitstellt.

Als Grundlage der Anwendung dienten die Schnittstellen von Facebook, Twitter und der Schnittstellenspezifikation OpenSocial. Alle diese Netzwerke stellen eine REST API bereit. Zur Authentifizierung der Anwendung, sowie der Nutzer setzten sie auf das oauth Protokoll.

Um möglichst flexibel bei der Verarbeitung von Objekten aus sozialen Netzwerken zu sein, wurde die Kopplung auf ein Minimum reduziert: Wie der Begriff des Netzwerks bereits versucht zu verdeutlichen, handelt es sich bei sozialen Netzwerken um Knoten, die mit einander verbunden sind.

Der einfachste Weg eine Anwendung zur Cloud-Anwendung zu portieren, ist es sie bei einem PaaS Anbieter zu hosten. Hier ist es nur nötig den Anwendungscode hochzuladen. Diese Einfachheit geht dem Nachteil einher, dass die Freiheit für Optimierungen und Anpassungen außerhalb der vorgegebenen Eigenschaften der Anwendung sehr eingeschränkt wird. Um diese Freiheiten jedoch zu bekommen, ist es nötig selbst eine Cloudumgebung zu entwickeln. Um die benötigte Skalierbarkeit der Anwendung zu gewährleisten, ist es notwendig sie auszuteilen. Dies kann auf Datenbankebene geschehen, dem so genannten Sharding oder auf Anwendungsebene. Um trotzdem den Eindruck einer zusammenhängenden Anwendung zu bekommen, wird der Einsatz eines Load Balancers nötig. Um der von der Cloud Definition geforderten Elastizität nachzukommen, muss ein Monitor eingesetzt werden, der bei Änderungen der Last Server beim Load Balancer hinzufügt oder entfernt.

Das Ergebnis dieser Bachelorarbeit bietet eine gute Grundlage zur Erweiterung um weitere Netzwerke. Zusätzlich bieten die verschieden Varianten des Hostings einen bereits für die Produktion geeignetes Ergebnis, sowie zwei unterschiedliche Varianten zur Optimierung, auf die bei Verzicht auf einen PaaS Anbieter zurückgegriffen werden könnte.

6.2 Methodische Abstraktion

Die während der Entwicklung der Anwendung gesammelten Erfahrungen lassen sich wie folgt hinsichtlich der Einbindung von sozialen Netzwerken, wie auch der Portierung in die Cloud zusammenfassen:

Einbindung sozialer Netzwerke Die Einbindung eines sozialen Netzwerkes bedeutet immer, einen Dienst zu nutzen, der von einem Dritten entwickelt und betrieben wird. Daraus können Probleme beim Debugging, sowie bei der Verfügbarkeit resultieren. Hinzu kommt eine höhere Latenz durch die geographisch unterschiedlich verteilten Server der Anwendung und des sozialen Netzwerkes. Neben diesen allgemeinen Herausforderungen sollte auch der Wahl der richtigen Bibliothek zur Kapselung des Netzwerken ein besonderes Augenmerk geschenkt werden. Ihr Funktionsumfang angebotener Bibliotheken reicht von der Kapselung des Anmeldevorgangs und des HTTP Verkehrs bis hin zum Vorhandensein aller im Netzwerk befindlichen Objekte. Hierbei gilt zu beachten, dass bei größerem Funktionsumfang auch die Möglichkeit der eigenen Anpassungen und Optimierungen geringer wird.

Cloud-Anwendungen Bei der Entwicklung einer Anwendung für die Cloud sollte von Beginn an die Skalierbarkeit mit berücksichtigt werden. Um eine einfache und effiziente Skalierung zu ermöglichen, sollten entsprechende Konzepte auf allen Ebenen des Systems zur Anwendung kommen. Eine Orientierung der REST Architekurstils erwies sich bei der Entwicklung dieser Anwendung als sehr hilfreich, da er eine einfache Orchestrierung von verschiedenen Komponenten wie Load Balancer und Anwendungsserver ermöglicht. Bei der Wahl von Standardkomponenten sollte Wert auf die mögliche Unterstützung von Skalierungskonzepten gelegt werden, so erleichtert die Möglichkeit von DataMapper verschiedene Repositories zu verwalten die Umsetzung des Sharding erheblich. Auf den Zugriff auf eine Datenbank oder Dateien sollte bei der Entwicklung von Cloud-Anwendung besonders geachtet werden, da hierdurch die physikalische Speicherung von Daten häufig Flaschenhälse auftreten können.

Um einen schnellen Release der Anwendung zu ermöglichen wurde bei dieser Anwendung zuerst ein PaaS Anbieter zum Deployment gewählt. So war es möglich, die Anwendung schnell dem Kunden verfügbar zu machen, ohne viele Konfigurationen vorzunehmen. Sollen diese Konfigurationen und weitere Änderungen vorgenommen werden, die der Anbieter nicht unterstützt, ist es nötig ihn zu wechseln. Hierzu ist es erforderlich bei der Entwicklung möglichst keine anbieterspezifischen Abhängigkeiten aufzubauen. Doch bietet das Cloud-Computing nicht nur Vorteile beim Deployment von Anwendung. Auch Testumgebungen können innerhalb der Cloud definiert und genutzt werden, ohne Hardware anzuschaffen, die fast nie vollständig genutzt wird. Zusätzlich ist es möglich mithilfe von einer Vielzahl an virtuellen Server sehr hohe Last für den Test der Anwendung zu erzeugen und zu steuern.

Zur Reduzierung der Zeit für Konfiguration und Überwachung der Cloud sollten möglichst viele Arbeitsschritte, unter anderem beim Start von Testumgebungen und dem Hinzufügen von

neuen Ressourcen, automatisiert werden. Um dies vorzunehmen bieten viele Cloud-Anbieter Schnittstellen zur Konfiguration der Umgebung an.

6.3 Ausblick

Cloud Computing ist in der Informatik angekommen und auch in den Massenmedien präsent. Wie am Anfang jeder Entwicklung sind Herausforderungen zu bewältigen um alle Vorteile nutzen zu können. Neben dem Cloud Computing spielen soziale Netzwerke eine immer wichtigere Rolle im Leben vieler Menschen. Ihr Potenzial wird heute erst von vielen Personen und auch Unternehmen entdeckt und wird sich weiter steigern. Ob das Kräfteverhältnis zwischen den jetzigen Anbietern bestehen bleibt, bleibt abzuwarten, da zum Ende dieser Arbeit Google gerade sein soziales Netzwerk Google+ releaste. Eine große Herausforderung wird es sein, die aktuelle Rechtslage an die neuen Bedingungen des Cloud Computing sowie sozialer Netzwerke anzupassen. Hierzu ist es aber nötig, ein tiefgreifendes Verständnis auch bei der gesetzgebenden Gewalt zu schaffen.

Eine Weiterentwicklung der Anwendung könnte den Aufbau automatisierter Deploymentlösungen und Tests einschließen oder ein JavaScript-basiertes Frontend einführen. Zunächst sollten jedoch weitere Implementationen von Netzwerken der Anwendung hinzugefügt werden. Als weiteres Ziel ist ein Ausbau der Monitoring Komponente anzusehen. Diese könnte neben Reaktionen auf Laständerungen genaue Überwachung aller Instanzen des Load Balancers sein. Neben den lastspezifischen Daten könnten auch Werte hinsichtlich Ausfall der Server oder Fehler in der Anwendung erhoben werden.

Glossar

API Ein **A**bstract **P**rogramming **I**nterface beschreibt eine Programmierschnittstelle auf Quelltextebene. In dieser Arbeit handelt es sich speziell um Schnittstellen zu Web Services. 5

DRY **D**on't **R**epeat **Y**ourself - Dieses Prinzip besagt, dass weder Daten noch Funktionalität redundant im System vorkommen sollen, da andersfalls der Wartungsaufwand erhöht wird. 19

Pipes and Filter Das Pipes and Filter Architekturmuster beschreibt ein System in dem verschiedene verarbeitende Komponenten (Filter) mithilfe von Datenströmen (Pipes) verbunden sind. 17

Ruby on Rails Bei Ruby on Rails handelt es sich um ein populäres Web- und Persistenz-Framework für die Programmiersprache Ruby. 19

Seperation of Concerns Seperation of Concerns beschreibt die Trennung von Zuständigkeiten innerhalb eines Systems. Es soll genau eine Aufgabe genau einem Element zugeordnet werden. So können Aufgaben besser verteilt werden und Änderungen sind einfacher durchzuführen. 14

Single Point of Failure Ein Single Point of Failure ist ein Element eines Systems, dessen Ausfall einen Ausfall des gesamten Systems zur Folge hat. 41

Skalierbarkeit Skalierbarkeit beschreibt das Verhalten eines Systems hinsichtlich des Ressourcenbedarfs bei Laständerungen in Bezug auf Performanz und Komplexität. 5

Literaturverzeichnis

[Anderson 2010] Anderson, Rebecca L.: Analysis of social network sites and supporting functions. In: *Proceedings of the 73rd ASIS&T Annual Meeting on Navigating Streams in an Information Ecosystem - Volume 47*. Silver Springs, MD, USA : American Society for Information Science, 2010 (ASIS&T '10), S. 143:1–143:2. – URL http://portal.acm.org/citation.cfm?id=1920331.1920515

[Animoto 2011] Animoto: *Animoto - Video Slideshow Makter with Music*. http://animoto.com/ (gesehen am: 31.05.2011 17:10). 2011

[Armbrust u. a. 2010] Armbrust, Michael ; Fox, Armando ; Griffith, Rean ; Joseph, Anthony D. ; Katz, Randy ; Konwinski, Andy ; Lee, Gunho ; Patterson, David ; Rabkin, Ariel ; Stoica, Ion ; Zaharia, Matei: A View of Cloud Computing. In: *Communications of the ACM* (2010), April

[Baun u. a. 2009] Baun, Christian ; Kunze, Marcel ; Nimis, Jens ; Tai, Stefan: *Cloud Computing: Web-basierte dynamische IT-Services*. Springer, 2009. – ISBN 978-3642015939

[Beri 2010] Beri, Jonathan: *OpenSocial REST Developer's Guide (v0.9)*. http://docs.opensocial.org/display/OSD/OpenSocial+REST+Developer(gesehen am: 16.05.2011 11:59). 2010

[Carr 2009] Carr, Nicholas: *The Big Switch: Der große Wandel. Cloud Computing und die Vernetzung der Welt von Edison bis Google*. mitp, 2009. – ISBN 978-3826655081

[Cebit 2011] Cebit: *Top-Thema der CeBIT 2011*. http://www.cebit.de/de/ueber-die-messe/daten-und-fakten/die-cebit-2011/cloud-computing-top-thema (gesehen am: 15.03.2011 16:21). 2011

[Chieu u. a. 2009] Chieu, T.C. ; Mohindra, A. ; Karve, A.A. ; Segal, A.: Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment. In: *e-Business Engineering, 2009. ICEBE '09. IEEE International Conference on*, oct. 2009, S. 281 –286

[Cucumber 2011] Cucumber: *Cucumber - Making BDD fun.* http://cukes.info/ (gesehen am: 15.08.2011 15:52). 2011

[DataMapper 2011] DataMapper: *DataMapper Ruby Object Relational Mapper.* http://datamapper.org/ (gesehen am: 21.06.2011 11:25). 2011

[Dix 2010] Dix, Paul: *Service-Oriented Design with Ruby and Rails.* Addison Wesley, 2010. – ISBN 978-0321659361

[Doc 2011] Doc: *Module Enumerable.* http://ruby-doc.org/core/classes/Enumerable.html (gesehen am: 20.06.2011 13:06). 2011

[Eilebrecht und Starke 2010] Eilebrecht, Karl ; Starke, Gernot: *Patterns kompakt Entwurfsmuster für effektive Software-Entwicklung.* Spektrum Akademischer Verlag, 2010. – ISBN 978-3-8274-2525-6

[Facebook 2011a] Facebook: *Authentication.* http://developers.facebook.com/docs/authentication/ (gesehen am: 18.04.2011 14:11). 2011

[Facebook 2011b] Facebook: *Über Facebook.* http://www.facebook.com/press.php (gesehen am: 18.04.2011 12:21). 2011

[Facebook 2011c] Facebook: *Facebook.* http://www.facebook.com (gesehen am: 31.05.2011 17:09). 2011

[Facebook 2011d] Facebook: *Graph API.* http://developers.facebook.com/docs/reference/api/ (gesehen am: 18.04.2011 13:21). 2011

[Fielding 2000] Fielding, Roy T.: *Architectural Styles and the Design of Network-based Software Architectures*, University of California, Dissertation, 2000

[Github 2011a] Github: *heroku / heroku.* https://github.com/heroku/heroku (gesehen am: 02.08.2011 17:42). 2011

[Github 2011b] Github: *intridea / oauth2.* https://github.com/intridea/oauth2 (gesehen am: 21.06.2011 11:34). 2011

[Haml 2011] Haml: *Haml.* http://haml-lang.com/ (gesehen am: 18.07.2011 16:19). 2011

[Hammer-Lahav u. a. 2011] Hammer-Lahav, Eran ; Recordon, David ; Hardt, Dick: *The OAuth 2.0 Authorization Protocol draft-ietf-oauth-v2-15.* Verfügbar unter: http://tools.ietf.org/pdf/draft-ietf-oauth-v2-15.pdf. April 2011

[HAProxy 2011] HAProxy: *HAProxy The Reliable, High Performance TCP/HTTP Load Balancer*. http://haproxy.1wt.eu/ (gesehen am: 02.08.2011 18:12). 2011

[Hargittai und li Patrick Hsieh 2010] Hargittai, Eszter ; Patrick Hsieh, Yu li: From dabblers to omnivores. A typology of social network site usage. In: *A networked self. Identity, community, and culture on social network sites*. Zizi Papacharissi, 2010

[Heroku 2011a] Heroku: *Heroku cloud application platform*. http://heroku.com (gesehen am: 28.06.2011 15:40). 2011

[Heroku 2011b] Heroku: *Heroku Dyno Isolation*. http://devcenter.heroku.com/articles/dyno-isolation (gesehen am: 28.06.2011 15:49). 2011

[Heroku 2011c] Heroku: *Heroku Dynos*. http://devcenter.heroku.com/articles/dynos (gesehen am: 28.06.2011 14:12). 2011

[Heroku 2011d] Heroku: *Heroku Scale*. http://www.heroku.com/how/scale (gesehen am: 28.06.2011 13:50). 2011

[JMeter 2011] JMeter: *Apache JMeter*. http://jakarta.apache.org/jmeter/ (gesehen am: 15.08.2011 16:02). 2011

[Mather u. a. 2009] Mather, Tim ; Kumaraswamy, Subra ; Latif, Shahed: *Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance*. O'Reilly Media, 2009. – ISBN 978-0596802769

[Richter und Koch 2008] Richter, Alexander ; Koch, Michael: Funktionen von Social-Networking-Diensten. In: *Proceding Multikonferenz Wirtschaftsinformatik 2008*, 2008, S. 1239–1250

[Rosenberg und Matoes 2011] Rosenberg, Jothy ; Matoes, Arthur: *The Cloud at Your Service*. Manning, 2011. – ISBN 1-935182-52-8

[Ruby 2011] Ruby: *Ruby - A Programmer's Best Friend*. http://www.ruby-lang.org/de/about/ (gesehen am: 18.07.2011 15:56). 2011

[Sessions 2011] Sessions, Roger: *Cloud Optimized Architectures For The Public Sector*. http://www.objectwatch.com/whitepapers/CUEC-PP-002-002.pdf. 2011

[Sinatra 2011] Sinatra: *Einführung*. http://www.sinatrarb.com/intro-de.html (gesehen am: 31.05.2011 18:06). 2011

[Sumter 2010] Sumter, La'Quata: Cloud Computing: Security Risk / Florida A&M University. 2010. – Forschungsbericht

[Twitter 2011a] Twitter: *API Documentation.* http://dev.twitter.com/doc (gesehen am: 16.05.2011 09:17). 2011

[Twitter 2011b] Twitter: *What is Twitter?* http://business.twitter.com/basics/what-is-twitter (gesehen am: 16.05.2011 09:11). 2011

[Wang 2008] Wang, Lizhe: *Virtual Enviroments for Grid Computing.* Universität Karlsruhe. 2008

[Winter 2010] Winter, Cornelia: *GI stellt zehn Thesen zu Sicherheit und Datanschutz in Cloud Computing vor.* Pressemittelung der Gesellschaft für Informatik http://www.gi.de/presse/pressearchiv/pressemitteilungen-2010/pressemitteilung-vom-1-dezember-2010.html ,gesehen am 06.06.2011. 2010

[Wirdemann und Baustert 2008] Wirdemann, Ralf ; Baustert, Thomas: *Rapid Web Development mit Ruby on Rails.* Hanser Fachbuchverlag, 2008. – ISBN 978-3446403949

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 25. August 2011 Steffen Brauer