Realisierung einer verteilten, Cloud-basierten Infrastruktur mit HTML5

Entwicklung einer Plattform zur Erforschung sozialer Netzwerke

Hans Katzenberger 2011

Diplomarbeit zur Erlangung des akad. Grades:
Diplom-Informatiker an der Universität Trier

Aufgabensteller:
Prof. Peter Sturm

Zweitkorrektor:
Ingo Scholtes

Die Diplomarbeit und sämtliche Quellcodes sind in digitaler Form auf CD-Rom hinten angefügt.

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbstständig und ohne unerlaubte Hilfe angefertigt und andere als die in der Diplomarbeit angegebenen Hilfsmittel nicht benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Hans Katzenberger, Trier, 2011

Abstract

This framework implements a technical core functionality as a basis for a distributed, cloud based infrastructure (e.g. a social game). It allows the user to access and interact with a global network (graph), which is stored in the cloud, and controls data consistency and propagation in real time.

It uses standards like HTML5 and JSON to create portable code and data that will run on a variety of platforms including desktop computers, tablets and mobile phones.

If you are a developer and you are looking for technical details in English, please skip forward to the Appendix (English):

7.1 A distributed, cloud based infrastructure with HTML5

7.2 Technical Documentary

Inhalt

1 Motivation 4

2 Anforderungen 6

2.1 Reichweite 6

2.2 Skalierbarkeit 7

2.3 Evaluation 7

2.4 Erweiterbarkeit 8

3 Grundlagen 9

3.1 Cloud Computing 9

3.2 HTML5 / JavaScript 14

3.3 Python 19

3.4 REST Architektur - Representational state transfer 20

3.5 Comet Architektur 20

3.6 SOA: Service Orientated Architecture 21

4 Design & Technik 22

4.1 Cloud Client Comet Design 22

4.2 Globaler Zustand in der Cloud 24

4.3 Smart Client: HTML5 / JavaScript 30

4.4 Evaluations Client 36

4.5 Spielimplementierungen 37

4.6 Sicherheitsaspekte 43

5 Evaluation & Auswertung 44

5.1 Interfaces 44

5.2 Reichweite 46

5.3 Theoretische Leistungsfähigkeit der Architektur 49

5.4 Praktische Leistungsfähigkeit: Stresstest 51

6 Fazit 54

7 Appendix (english) 56

7.1 A distributed, cloud based infrastructure with HTML5 56

7.2 Technical Documentary 57

8 Quellen 61

8.1 Weitere Quellen 65

1. Motivation

Die Beschleunigung der Kommunikation und die neue Art der Nutzung digitaler Medien, insbesondere aber der allgegenwärtige, hochfrequente Datenaustausch, sowohl zwischen Menschen als auch Maschinen, fordern einen Paradigmenwechsel in der Auswertung und Aufbereitung solcher Datenströme.

Während noch vor wenigen Jahren die Klassifizierung von Daten im Vordergrund stand, geht es heute vielmehr darum, die subjektiven Anforderungen des jeweiligen Nutzers zu erfassen und erfüllen. Diese Entwicklung ging zunächst von einer objektiven Kategorisierung von Daten in Verzeichnissen und Portalen wie AOL - America Online und dem ursprünglich rein Link-basierten Pagerank Algorithmus von Google aus. Schrittweise, durch stetiges Einflechten immer weiterer Indikatoren wie Herkunftsland oder verwendetem Browser, wurde die relevante Datenauswahl für den Nutzer immer mehr personalisiert und verfeinert[1] (z.B. Google[2] und Bing[3]). Zukünftig wird dieser Trend durch integrierte Daten aus den großen sozialen Netzwerken Facebook, Google+ und Twitter weiter zunehmen und letztendlich eine vollständig individuell zugeschnittene, subjektive Datenevaluation für jeden Nutzer herbeiführen.

Anforderungen an neue Algorithmen liegen also im Besonderen in der Bewertung der Rolle des Nutzers, sowie seiner Eigenschaften und Bedürfnisse. Diese Informationen sind oft unbewusst und deshalb nur indirekt erschließbar. Sie können demnach nur aus dem Verhalten des Nutzers geschlossen oder aber in seiner Umgebung gefunden werden, die sich digital als sein soziales Netz interpretieren lässt. Heute sind diese Daten bereits teilweise vorhanden, aber nur schwer für konkrete, allgemeine Vorhersagen nutzbar. In bestimmten Nischen[4] [5] gibt es mit proprietären (und nicht-öffentlichen) Algorithmen[6] bereits Erfolge, für weitergehende wissenschaftliche Arbeiten fehlt aber oft eine aussagekräftige empirische Datengrundlage[7].

Die hier vorgestellte Arbeit soll das Erzeugen und Evaluieren solcher Daten ermöglichen und besteht aus zwei Teilen: einem Framework, das mithilfe von Cloud- und HML5 Technologien eine verteilte Infrastruktur bereitstellt und zwei Mehrspieler Spielen (Multiplayer Games), die beispielhaft die konkrete Implementierung einer Anwendung unter Verwendung dieses Frameworks realisieren.

---

[1] Kai Riemer, Fabian Brüggemann, 2009: Personalisierung der Internetsuche – Lösungstechniken und Marktüberblick, bit.ly/tVCw6N

[2] 2009 aktivierte Google auch für nicht eingeloggte Nutzer die personalisierte Suche. „Personalized Search for everyone“, bit.ly/71RcmJ

[3] Personalisierte News für Bing / Windows Phone 7: „News is now yours”, binged.it/dTDqA0

[4] Empfehlungsdienst Hunch: „Hunch's ambitious mission is to build a 'Taste Graph' of the entire web, connecting every person on the web with their affinity for anything, (…) Hunch is at the forefront of combining algorithmic machine learning with user-curated content, (…)”, http://www.hunch.com

[5] Musik Empfehlungsdienst LastFM: „Based on your taste, Last.fm recommends you more music and concerts!“, http://www.last.fm/

[6] Algorithmus zur sozialen Filmbewertung von Moviepilot und der Humboldt-Universität Berlin: „Through semantic analysis we connect content with fans of particular projects taking their movie preferences and social likes of actors, (…), etc. into consideration.”, http://moviepilot.com, http://de.wikipedia.org/wiki/Moviepilot

[7] Castellano, Fortunato, Loreto, 2009, Statistical physics of social dynamics

Dieses auf weithin akzeptierten Standards basierende Framework soll eine leicht erweiterbare technische Basis zur empirischen Erforschung von (sozialen) Netzwerken bereitstellen und das Beobachten, Replizieren und Evaluieren der Entstehung von Netzwerken und der Ausbreitung von Informationen in denselben ermöglichen.

Idealerweise wird dabei eine technisch elegante, theoretisch unendlich skalierbare und reichweitenstarke Architektur entworfen, implementiert und evaluiert. Dazu eignen sich in besonderem Maße die hier verwendeten, im Entstehen begriffenen Technologien wie Cloud-Computing und asymmetrische Lastverteilung hin zum Client in Echtzeit.

2. Anforderungen

Um robuste Aussagen oder gar Vorhersagen über die Dynamik von Netzwerken treffen zu können, ist eine qualitativ und quantitativ hochwertige, empirische Datenbasis unverzichtbar. Eine Plattform, die diese Daten liefern kann, muss über eine hohe Reichweite (Datenqualität) verfügen und auch mit großen Nutzerzahlen skalieren (Datenquantität).

Außerdem müssen sämtliche Daten in einem geeigneten Format (standarisiert, portabel) repräsentiert werden, um eine effiziente Evaluation auf verschiedenen Plattformen zu ermöglichen.

Im Hinblick auf zukünftige Einsatzzwecke des hier vorgestellten Frameworks wurde ein modularer und erweiterbarer Aufbau (gegenüber einem monolithischen Ansatz) gewählt. Ein Parallelbetrieb oder die gleichzeitige Nutzung von anderen Bibliotheken oder Frameworks ist ohne weiteres möglich.

2.1 Reichweite

Um eine möglichst große, heterogene und aussagekräftige Spielergruppe zu erschließen, ist eine maximale Reichweite bei hoher Zugänglichkeit nötig. Dazu muss die Funktionalität auf möglichst vielen

Browsern (z.B. Internet Explorer[8], Firefox[9], Chrome[10], Safari[11]) bzw.

Endgeräten (z.B. Desktop-Computern, Tablets, Handys) und

Betriebssystemen (z.B. Windows[12], Linux[13], iOS[14], Android[15])

gewährleistet sein. Eine Installation von Drittsoftware, Plugins oder sonstigen zusätzlichen Programmen stellt ebenfalls eine Hürde für viele Anwender dar.

Eine derartige Plattformunabhängigkeit lässt sich heute (ohne zusätzliche Software) nur Web- (bzw. Browser-) basiert, unter Verwendung von verbreiteten Standards, erreichen. Die steigende Bedeutung von Webtechnologien und ihre Ausweitung hin zu anderen Geräteklassen (z.B. Handys, Tablets, Fernseher, Spielekonsolen, embedded systems) sichert außerdem die Zukunftsfähigkeit und wird die Reichweite sogar noch vergrößern.

Mögliche Kandidaten sind also auch Webtechnologien wie Adobe Flash oder Microsoft Silverlight. Aus den oben genannten Gründen (hauptsächlich wegen der nicht nativen Unterstützung im Browser), nutzt das Framework den nicht-proprietären Standard HTML5.

---

[8] Microsoft Internet Explorer, bit.ly/dMOR0S

[9] Mozilla, http://www.mozilla.org/en-US/firefox/new/

[10] Google Chrome, http://www.google.com/chrome

[11] Apple Safari, http://www.apple.com/safari/

[12] Microsoft Windows, bit.ly/1Wl1ZM

[13] Linux, https://www.linux.com/

[14] Apple iOS, http://www.apple.com/ios/

[15] Android OS, http://www.android.com/

2.2 Skalierbarkeit

Viele Applikationen skalieren nur begrenzt, unter bestimmten Voraussetzungen oder auf bestimmten Architekturen. Meist treten dabei Schranken auf, deren Überwinden ein Anpassen der Software oder Aufrüsten der zugrundeliegenden Hardware notwendig macht.

Monolithisch designte Architekturen können, durch entsprechende Investitionen in Hardware und Modifikation der Software, für große Nutzerzahlen angepasst werden. Durch die steigende Gesamtkomplexität wird eine Pflege des Systems allerdings zunehmend erschwert und irgendwann unmöglich. Der Wechsel zu einer serviceorientierten Architektur[16] ist dann unausweichlich. Ein prominentes Beispiel für diese Entwicklung sind die VZ-Netzwerke der Holtzbrinck-Gruppe[17].

Die Verwendung von serviceorientierten Ansätzen ist ein erster möglicher Schritt zur Schaffung von besonders skalierbaren Architekturen. Optimaler Weise müssen aber auch die einzelnen Services selbst gut mit dem Gesamtsystem skalieren, da die schwächste Komponente die Grenze der gesamten Leistungsfähigkeit festlegt.

Hier biete sich die die Verwendung von virtualisierten Cloud-Plattformen als Grundlage für solche Dienste an. Bei fachgerechter Implementierung haben entsprechende, auf Clouds laufende Services ein extremes Skalierungs-Potential. Außerdem ist es möglich, mit vergleichsweise geringem Aufwand und kleiner Budgetgröße, verteilte Anwendungen für hohe Nutzerzahlen zu realisieren.

2.3 Evaluation

Die Spieler sollen auf der geschaffenen Plattform miteinander interagieren und kollaborieren, um vorgegebene Aufgaben zu lösen. Alle Vorgänge des Netzwerks sollten auch im Nachhinein nachvollziehbar und einfach zugänglich sein. Dazu ist ein offenes und portables Datenformat erstrebenswert, das auch auf fremder Software untersucht werden kann.

Sämtliche Spielabläufe und Handlungen der Nutzer, also Veränderungen im zugrundeliegenden Netzwerk, werden protokolliert. Alle Daten liegen im JSON[18] Format vor und sind plattformübergreifend portabel, um eine Auswertung auch mit anderer Software zu ermöglichen. Sie können mit einer beliebigen webfähigen Technologie direkt an der Cloud-Schnittstelle ausgelesen werden.

Ein HTML/JS-Client ist bereits im Framework enthalten. Er stellt die Entwicklung der gesamten Entstehung des Netzwerks, bzw. die Änderungen darin, grafisch im Zeitraffer dar.

Die zweite Möglichkeit zur Evaluation stellt die zugrundeliegende Cloud bereit, die (zu Abrechnungszwecken) sämtliche Transfers und Änderungen protokolliert. Über ein komfortables Dashboard lassen sich alle technischen Daten in Echtzeit einsehen.

---

[16] SOA: Service Orientated Architecture, siehe 3.6 SOA

[17] Restrukturierung der VZ Netzwerke: „Letzte Chance für die VZ-Netzwerke“, http://www.golem.de/1109/86675.html

[18] JSON: JavaScript Object Notation, siehe JSON

2.4 Erweiterbarkeit

Besonderer Wert wurde auf leichte Erweiterbarkeit und hohe Zugänglichkeit gelegt, sodass sich Modifikationen oder alternative Module schnell und effizient implementieren lassen. Das eigentliche System stellt nur eine feste Kernfunktionalität zur Verfügung.

Darüber hinausgehende Funktionalität kann durch Erweiterungen, wie zum Beispiel Infovis[19], implementiert werden, die abhängig vom konkreten Spiel sind. Durch die Art der Kapselung des Frameworks ist es möglich, mehrere beliebige externe Bibliotheken (jQuery, Infovis, etc.) parallel zu verwenden, ohne dass dabei Konflikte entstehen.

Eine Besonderheit von JavaScript basierten Webanwendungen ist die Möglichkeit, zu einem beliebigen Zeitpunkt Daten und Code in die Applikation einzubetten bzw. nachzuladen. Moderne Browser parsen sämtliche Daten, seien es JavaScript, Stylesheets oder Text, und interpretieren diese sobald sie in der Webseite eingefügt werden. Zu Demonstrationszwecken sind bereits verschiedene Infovis Visualisierungen (Sunburst, Hypertree, Spacetree) im Spiel enthalten und dynamisch ladbar.

[19] Infovis, http://thejit.org

3. Grundlagen

Im folgenden Kapitel werden einige technische Grundlagen erläutert.

3.1 Cloud Computing

Clouds („[in der] Wolke“) bezeichnen die Bereitstellung von Rechen- und Speicherkapazität über das Internet durch einen Hosting-Anbieter. Architekturbedingt sind dies meist Firmen mit großen, teils redundanten Rechenzentren. Die zugrundeliegende Architektur ist dabei unsichtbar („wolkig“) und für den Nutzer uninteressant. Ihm wird stets nur die benötigte Leistung bereit- und in Rechnung gestellt. Die zentrale Eigenschaft von Cloud-Diensten ist die hohe Abstraktionsebene, die für alle bereitgestellten Ressourcen wie Rechenkapazität, Speicherplatz oder komplette Software gilt. Durch diese Abstraktion lassen sich oft die Grenzen konventioneller Software aufheben.

Der Begriff „Cloud“ ist nicht eindeutig abgrenzbar, allerdings können verschiedene Modelle unterschieden werden:

IaaS: Infrastructure as a Service
Stark skalierbare Umgebung für virtuelle Instanzen von Servern, Speicherdiensten, etc.

PaaS: Platform as a Service
Virtuelle Umgebung für Anwendungen, die direkt in der Cloud lauffähig sind (ähnlich einem Betriebssystem) und von dieser APIs etc. bereitgestellt bekommt (persistenter Speicher, Kommunikation).
Die hier verwendete Google App Engine entspricht einer PaaS Cloud.

SaaS: Software as a Service
Webapplikationen, die vollständig im Browser und über das Internet bereitgestellt werden.

„Cloud Computing“ ist in den vergangenen Jahren einer der am häufigsten verwendeten Marketing Begriffe („Buzzwords“) in der IT[20]. Die professionelle öffentliche Meinung tendiert dazu, dass für viele Anwendungszwecke die Vorteile von Cloud Computing die Nachteile überwiegen.

[20] Häufig genutzte Buzzwords 2008: „Most Confusing High Tech Buzzwords: Cloud Computing, Green Washing & Buzzword Compliant“, bit.ly/vG3NfM

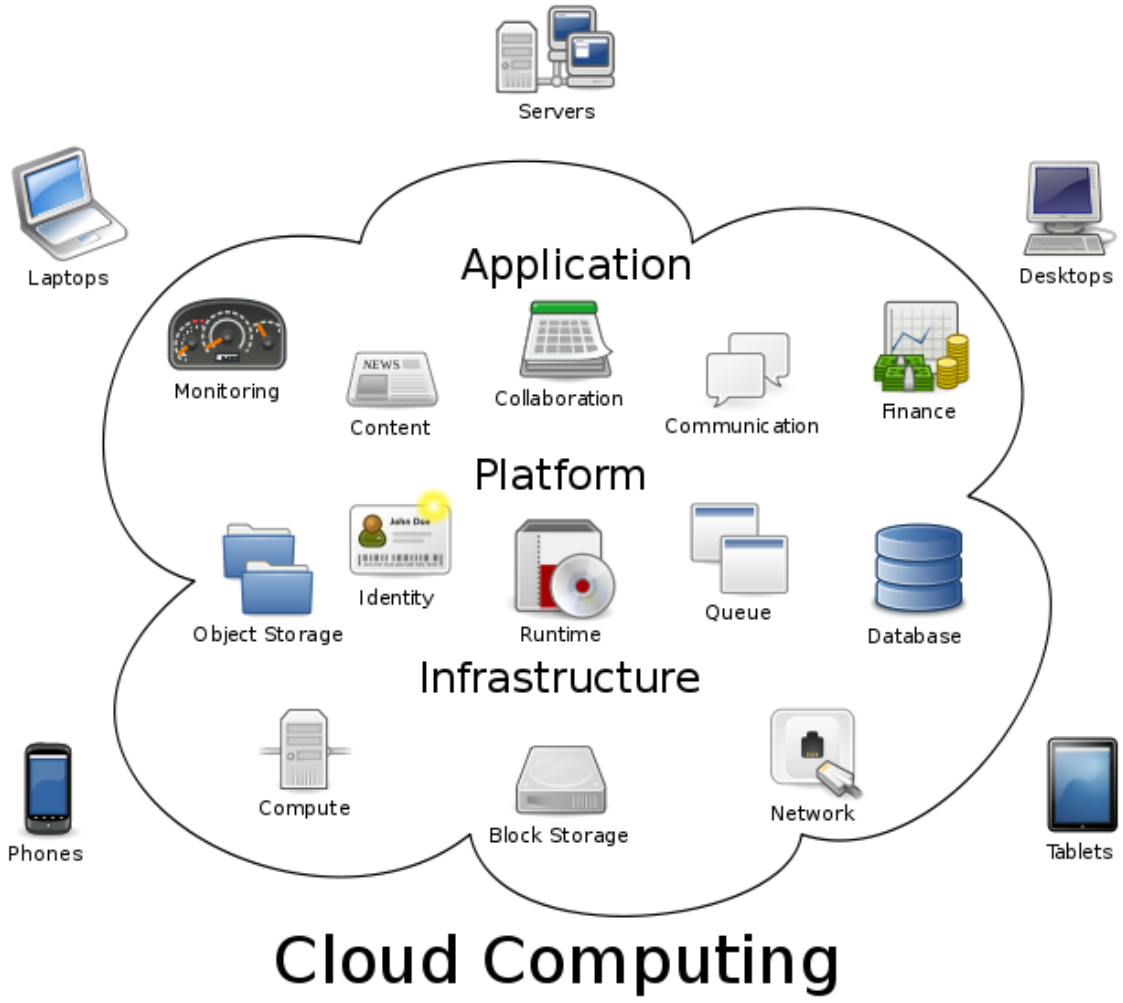


Bild Quelle: Wikipedia Deutschland
http://de.wikipedia.org/w/index.php?title=Datei:Cloud_computing.svg

Vorteile von Clouds für Nutzer

Die Vorteile einer Cloud liegen in der starken Abstraktion der zur Verfügung gestellten Ressourcen wie Rechenzeit und Speicherplatz. Diese sind theoretisch unbeschränkt und können den Anforderungen zeitnah angepasst werden (beispielsweise um Lastspitzen aufzufangen).

Der Kunde zahlt nur für die tatsächlich in Anspruch genommene Leistung (im Gegensatz zu traditionellen, fest abgerechneten Hosting-Lösungen wie Webspace, virtuelle oder dedizierte Server).

Durch mehrfache Streuung der Daten auf (geografisch) verschiedene Rechenzentren werden sowohl erhöhte Ausfallsicherheit (durch Redundanz) als auch kurze Antwortzeiten (Response Time) durch lokale Nähe erreicht.

Vorteile von Clouds für Anbieter

Auf Anbieterseite lassen sich brach liegende Ressourcen gewinnbringend nutzen, außerdem treten positive Pooling- und Skaleneffekte auf. Die Ursprünge der Cloud-Entwicklung liegen in extrem schnell wachsenden Internetfirmen mit stark heterogener Serverauslastung (z.B. Amazons Weihnachtsgeschäft).

Nachteile von Clouds

Durch das Outsourcen von Informationen in die Cloud geht die Zugriffskontrolle über sie verloren und wird in fremde Hände übergeben. Dies spielt sowohl bei Ausfällen oder Datenverlust (z.B. Google Mail, Hotmail) als auch rechtlich eine große Rolle.

Die Richtlinie 95/46/EG[21] (Datenschutzrichtlinie) fordert den Schutz von personenbezogenen Daten innerhalb der Europäischen Union. Es ist ohne eine ausdrückliche Erlaubnis der entsprechenden Person nicht erlaubt, sensible Daten ins Ausland abzuführen. Da Clouds Daten teils sogar gezielt in geografisch verteilten Rechenzentren speichern (Redundanz zur Erhöhung der Ausfallsicherheit), ist dieser Aspekt kritisch zu betrachten.

Einige Anbieter nutzen diese Umstände und bieten spezielle Clouds für deutsche Mittelstandsunternehmen an und bewerben diese explizit mit Standorten in Deutschland (Pironet[22], Nionex[23]).

Mit einer homomorphen Verschlüsselung ist es möglich, auf verschlüsselten Daten zu rechnen und diese erst beim Endnutzer zu entschlüsseln. Allerdings existieren zum jetzigen Zeitpunkt noch keine marktfähigen Implementierungen dieser Technologie. Unter Laborbedingungen wurde die grundsätzliche Machbarkeit gezeigt[24], für echte Anwendungen ist sie aber (noch) nicht robust genug. Einige Cloudhoster bieten integrierte, symmetrische Verschlüsselung von Dateien an[25], was aber bei großen Daten problematisch werden kann, da selbst bei kleinen Änderungen die komplette Datei übertragen und ausgetauscht werden muss.

Nach aktuellem Stand der Forschung scheint die homomorphe Verschlüsselung aus Nutzersicht die einzige umfassende und robuste Lösung zu sein.

Google App Engine

Die Google App Engine (GAE) ist ein PaaS-Dienst („Platform as a Service“ = Bereitstellung einer Computer Plattform für Webanwendungen und deren Hosting in der Cloud) von Google.

---

[21] Richtlinie 95/46/EG: „Datenschutzrichtlinie“, bit.ly/6a7sY0

[22] Pironet NDH, http://www.pironet-ndh.com

[23] Nionex, http://cloud.nionex.de/

[24] Nigel P. Smart und Frederik Vercauteren: *„Fully Homomorphic SIMD Operation“,* erscheint 2011, http://bit.ly/uT5cLz

[25] BoxCryptor, http://www.boxcryptor.com/

Durch Virtualisierung wird eine automatisch skalierende Hosting-Umgebung für internetbasierte Anwendungen erreicht. Diese ist in der Nutzung bis zur Überschreitung bestimmter Ressourcen-Grenzen kostenlos, vorausgesetzt wird allerdings ein Google-Account. Die Infrastruktur wird als bewährte Plattform beworben, auf der auch Googles eigene Dienste laufen, und die sich neben der hohen Leistungsfähigkeit und Skalierbarkeit vor allem durch einfache Administration (GAE Dashbord) auszeichnet.

Beschränkungen der kostenlos nutzbaren Version (Stand 11/2011):

Quota	Limit
Emails per day	2000
Bandwidth in per day	1 GB
Bandwidth out per day	1 GB
CPU time per day (to be removed)	6.5 hours per day
Instance-hours (IH)	24 hours per day
Data stored	1 GB
URLFetch API calls per day	657,084

Quelle: http://en.wikipedia.org/wiki/Google_App_Engine

Diese Limitierungen werden im Kapitel Stresstest noch einmal aufgegriffen.

Darunter liegende Verwaltungsaufgaben, wie die Datenreplikation über mehrere Server, werden durch die Cloud-Schnittstelle versteckt und sind komfortabel nutzbar (Paxos Algorithmus[26]).

Die GAE stellt folgende Programmiersprachen zur Verfügung:

Java[27] (Java),
Python[28] (Python) und
Go[29] (Go)

Fortgeschrittene Implementierung der Channel-API in Python, sowie die besondere Nähe der Programmiersprache zur GAE Infrastruktur und der kompakte Quellcode sind die Hauptgründe für die Wahl von Python (gegenüber Java) für den Cloud-Speicher.

Go wird selbst von Google noch als experimentell bezeichnet und der Verbreitungsgrad ist äußerst gering.

Als Entwicklungsumgebung wird von Google ein SDK[30] bereitgestellt[31], mit dem sich App Engine Applikationen entwickeln und testen lassen. Hierzu emuliert ein lokaler Server die GAE Laufzeitumgebung und stellt entsprechende APIs etc. bereit.

---

[26] Lamport, 2001, Paxos made simple, SIGACT News 32(4):18-25, bit.ly/EZOqD

[27] Programmiersprache Java, http://www.oracle.com/technetwork/java/

[28] Programmiersprache Python, http://www.python.org/

[29] Programmiersprache Go, http://golang.org/

[30] SDK, Software Developement Kit, http://en.wikipedia.org/wiki/Software_development_kit

Außer den entsprechenden Python Dateien enthält eine Applikation zwei weitere Dateien:

app.yaml : Konfigurationsdatei, Zuordnung von Urls zu Dateihandlern etc.
index.yaml : Indexdatei für Datenbankzugriffe, automatisch erzeugt

Per SDK lassen sich Anwendungen komfortabel in der Cloud ablegen und aktualisieren. Außer dem SDK sind alle zum Ausführen des Frameworks benötigten Dateien enthalten.

GAE Channels

Die GAE erfüllt mit der direkten und asymmetrischen Kommunikationsmethode („Channels“[32]) die Anforderung, die von den Clients propagierten Statusänderungen bzw. Spielhandlungen direkt, also in Echtzeit, an benachbarte Clients zu pushen. Diese Verbindung ist persistent und erlaubt es Webanwendungen mit der Cloud Daten auszutauschen, ohne dass permanent kommuniziert werden muss. Ohne eine solche Technik müsste man auf Ressourcen-intensives Polling zurückgreifen, also ein ständiges Abfragen, ob neue Daten vorliegen.

Das relativ junge Alter dieser Echtzeittechnik spiegelt sich im noch stark begrenzten Funktionsumfang der Channels wider. Die Comet-artige Channels Technik soll in Zukunft höchstwahrscheinlich durch den HTML5 Standard Websockets[33] ersetzt werden[34].

Die Push-Technologie kann ebenfalls über Websockets realisiert werden, die aber zum Zeitpunkt der Erstellung der Diplomarbeit noch nicht in der GAE verfügbar sind. Es ist absehbar, dass diese Funktionalität noch bereitgestellt wird, allerdings nicht wann. Es existieren vergleichbare Cloud-Plattformen, die bereits Websockets unterstützen und die Funktionalität der GAE größtenteils implementieren können, wie z.B. die TyphoonAE[35]. Diese Implementierungen erfreuen sich allerdings nur eines kleineren Verbreitungsgrads und können die Funktionalität des „Original“ GAE aber nur annähernd simulieren.

Alternative Clouds

Die wohl bekannteste Cloud, Amazon S3[36], ist für den gegebenen Zweck nicht geeignet, da sie eine Authentifizierung des Clients in Form einer geheimen ID voraussetzt. Da der genutzte Spielclient aber rein auf Basis von HTML5 und JavaScript realisiert ist, ist ein solider Schutz des Authentifizierungs-Schlüssels nicht möglich, da JavaScript als Schriftsprache stets lesbar ist, also der Schlüssel extrahierbar bleibt. JavaScript Packer / Crypter[37] und Obfuscater[38] erlauben lediglich eine

---

[31] GAE SDK Python, bit.ly/Ng50E

[32] Google App Engine: The Channel API, bit.ly/sfjOoJ

[33] HTML5 Websockets, http://dev.w3.org/html5/websockets/

[34] App Engine gets Streaming API (…), bit.ly/h0bIYQ

[35] Alternative Laufzeitumgebung für Google App Engine Programme, Typhoon App Engine: http://code.google.com/p/typhoonae/

[36] Amazon Simple Storage Service, http://aws.amazon.com/s3/

[37] JavaScript Packer, bit.ly/493Cja

Verschleierung und keinen effektiven Schutz. Das Umgehen der Authentifizierung mithilfe eines Proxyservers, der die Anfragen der Clients an die AS3 mit dem geheimen Schlüssel versieht und von Server zu Server weitergibt, zerstört die gewünschte direkte Kommunikation der Clients mit dem Cloud-Speicher und führt einen unnötigen Engpass ein.

Grundsätzlich ließe sich ein derartiges Framework auch auf einer anderen technischen Plattform als der Google App Engine realisieren. Amazon als prominentester und einer der ersten Anbieter von Cloud-Diensten verfügt über das wahrscheinlich breiteste Portfolio, aber auch Microsoft Azure oder IBM SmartCloud bieten inzwischen eine komplette und umfassende Produktpalette (IaaS, SaaS, Paas).

Die ausschlaggebenden Gründe für die GAE Cloud sind der relativ einfache Zugang, die vorhandenen Features (z.B. Echtzeitkommunikation und ausgereiftes Dashboard) und die Möglichkeit der kostenlosen Nutzung.

3.2 HTML5 / JavaScript

Die HyperText Markup Language (HTML) ist die meistgenutzte Auszeichnungssprache im Internet. Durch Tags werden Textdokumente um Metainformationen angereichert und bilden so Webseiten. HTML ist eine Unterart der Standard Generalized Markup Language (SGML[39]).

Durch die Einbettung von „echten“ Programmiersprachen wie JavaScript[40] lässt sich ein Teil der Ablaufsteuerung vom Server zum Client transferieren und damit die Implementierung von responsiven Webapplikationen realisieren.

Traditionelles HTML

Die ursprüngliche Verwendung von HTML als reine Auszeichnungssprache war stark limitiert. Interaktivität beschränkte sich für die Nutzer auf das Folgen von Hyperlinks, die Dokumente an sich waren aber vollständig statisch konzipiert.

Trotzdem führte die Möglichkeit einer grafischen und intuitiveren Navigation (gegenüber text-basierten Shell-Systemen) schnell zu einer hohen Verbreitung und trug letztendlich stark zur rapiden Verbreitung des Internets bei. Die erste konzeptuelle Erwähnung findet sich 1989 bei Tim Berners-Lee als „Internet-basiertes HyperText System“[41].

---

[38] JavaScript Obfuscator, http://www.javascriptobfuscator.com/

[39] SGML, Standard Generalized Markup Language, http://www.w3.org/MarkUp/SGML/

[40] JavaScript / ECMAScript, bit.ly/FpAue

[41] Tim Berners-Lee, "Information Management: A Proposal." CERN (March 1989, May 1990). http://www.w3.org/History/1989/proposal.html

HTML5

HTML5 stellt die fünfte und neuste Version der Hypertext Markup Language dar. Dieser vom W3C Konsortium spezifizierte Entwurf sieht vor allem neue Möglichkeiten zur Erschaffung von Web-Applikationen vor und beerbt HTML4 und XHTML. Die neuen Features tragen dabei den sich rapide entwickelnden Browsern und ihren Rendering- und JavaScript Engines Rechnung. Während das ursprüngliche HTML als reine Auszeichnungssprache entworfen wurde, eignet sich HTML5 für vollständige Anwendungen. HTML5 ist nicht als abgeschlossen anzusehen, sondern stetiger Weiterentwicklung unterworfen und noch permanent im Fluss. Der Fortschritt der Implementierung in den verschiedenen Browserengines ist daher auch nicht gleichmäßig und kann von Browser zu Browser variieren.

Auch die HTML zugrundeliegende Hardware hat sich von reinen Desktop Computern hin zu mehreren und verschiedenen Plattformen wie Handys, Tablets und Fernsehern gewandelt. Die Zukunft von Webapplikationen wird zum größten Teil auf HTML5 aufbauen. Kein anderer Standard bietet eine vergleichbare Reichweite und Funktionalität. Die Summierung vieler dieser Vorteile haben zu einer hohen und aktuell immer noch steigenden Popularität von HTML5 geführt.

Unter dem Oberbegriff HTML5 versteht man im weitesten Sinne auch eng verknüpfte Technologien wie JavaScript und Css3 (W3C, CSS3 Cascading Style Sheets). Im Folgenden werden einige Schlüsselaspekte hervorgehoben, die HTML5 von seinen Vorgängerversionen unterscheiden:

Semantik

Spezielle Tags und RDFa[42] ermöglichen es, Daten eine semantische Bedeutung zuzuordnen und diese maschinell lesbar und für weitere Verarbeitung zugänglich zu machen. Diese Technologien entspringen aus XML (eXtensible Markup Language) und sind über XHTML in HTML5 eingeflossen.

Lokaler Speicher / Offline Storage

Ein großer Nachteil von Webanwendungen ist ihre Abhängigkeit von einer (permanenten) Internetanbindung. Durch die Möglichkeit auch größere Datenmengen konsistent und sicher lokal zu speichern und den Applikationen zugänglich zu machen, entfällt dieser Nachteil jedoch. So können nicht nur kurze „Aussetzer“ der Verbindung abgefangen und überbrückt werden, sondern ein komplett autarkes Arbeiten im Offlinemodus wird möglich. Die Fähigkeiten gehen dabei über die bloße Datenablage hinaus und implementieren sogar grundlegende Datenbankfunktionen. Mit dem Chrome-Browser ist es beispielsweise möglich, Google Mail im Offline Modus zu verwenden.

Hardware Schnittstellen

Zugriff auf Hardwareschnittstellen wie Geolocation, Kamerabild oder Lagesensor (vor allem auf mobilen Geräten wie Handys und Tablets) war bisher nativen Applikationen vorbehalten. Da Hardwarezugriffe tiefgehende Zugriffsrechte im System erfordern spielen hier

[42] Erweiterung von XML zur Einbettung von Metadaten in Dokumente, Resource Description Framework, http://en.wikipedia.org/wiki/RDFa

Sicherheitsbestimmungen eine besondere Rolle. Viele moderne JavaScript Engines laufen deshalb in einer Sandbox.

Erweiterte Kommunikationskanäle

Wo HTML ursprünglich auf statische Client / Server Strukturen begrenzt war, sieht HTML5 neue asynchrone Echtzeitkommunikationskanäle in Form von Websockets und Event-basiertem Datenaustausch vor.

3D Hardwarebeschleunigung

Um die bereits extrem angewachsene Leistungsfähigkeit der Browserengines weiter zu steigern, gibt es erste Möglichkeiten, die hochgradig parallele und performante Grafikhardware für JavaScript Berechnungen zugänglich zu machen. Erste Standards wie z.B. WebGL[43] demonstrieren in Proof-of-Concept Applikationen schon vollwertige 3D Engines im Browser[44].

JavaScript / Ajax

JavaScript ist die am häufigsten verwendete (clientseitige) Scriptsprache zur Erzeugung von dynamischen Webseiten. Sie ist dynamisch, schwach typisiert und unterstützt mehrere Programmierparadigmen, wie objektorientierte, imperative oder funktionale Programmierung. Durch einen kompakten Sprachkern kann der Interpreter auch auf relativ schwacher Hardware effizient implementiert werden (mobile Geräte wie Handys).

Einer der häufigsten Einsatzzwecke von JavaScript auf Internetseiten ist es, rein statische HTML-Dokumente zu verändern und Daten nachzuladen (Ajax). Während anfängliche Einsätze von JavaScript auf Webseiten oft als störend bzw. als Spielerei abgetan wurden[45], sind moderne Web basierte Applikationen ohne JavaScript undenkbar (abgesehen von Drittsoftware wie Adobe Flash). Aufgrund der hohen Verbreitung und weiterer Eigenschaften hat sich JavaScript weitere Einsatzgebiete erschlossen, wie beispielsweise die Webserver Implementierung Node.js[46]. Die Bedeutung von JavaScript ist über die Jahre stetig angewachsen und wird in Zukunft weiterhin zunehmen.

JavaScript ist des ähnlichen Namens wegen nicht mit Java zu verwechseln oder programmiertechnisch in Verbindung zu bringen. Trotz der beiden Sprachen zugrundeliegenden Objektorientierung ist das Konzept jeweils unterschiedlich ausgeprägt (Prototyp-basierte Objektorientierung in JavaScript im Gegensatz zu Klassen-basierter Objektorientierung in Java). JavaScript basiert auf der ECMA-262 Spezifikation[47].

---

[43] WebGL Spezifikation, bit.ly/eFqoAU

[44] Google Chrome Experiments, http://www.chromeexperiments.com/

[45] Popups, verschleiern von Links, etc., http://de.wikipedia.org/wiki/JavaScript#Missbrauch

[46] Modularer, Ereignis (Event) gesteuerter Webserver auf Basis von JavaScript, Node.js, http://nodejs.org

[47] ECMA Script, ECMA-262 Spezifikation, http://www.ecmascript.org/

JSON Datenaustausch

Die „JavaScript simple Object Notation“ ist sehr kompakt und erzeugt wenig Overhead oder redundante Daten und damit eine effizientere und schnellere Kommunikation. Als Bestandteil von JavaScript ist diese Wahl naheliegend. Die Verwendung von XML wäre technisch genauso möglich, würde die Daten aber unnötig aufblähen und zusätzliche Bibliotheken erfordern (auf Client- und Serverseite).

JavaScript Objekte bestehen im Kern aus „Schlüssel“ : „Wert“ – Paaren (ähnlich wie assoziative Arrays), die ihrerseits Objekte darstellen und in sich geschachtelt sein können. JSON stellt quasi eine Art serialisierte Notation dieser Objekte dar.

Sämtliche Kommunikation zwischen Client und Cloud, abgesehen vom initialen Download der statischen Seite, geschieht in Form von JSON[48]-codierten Daten. Die Beschränkung des Datenaustauschs auf diesen Standard gewährleistet eine hohe Zugänglichkeit für andere (Web-) Applikationen.

AJAX – Asynchronous JavaScript and XML

AJAX bezeichnet das dynamische Nachladen (im Hintergrund) von Daten oder Code mit JavaScript. Die ausführende HTML-Seite kann so aktualisiert oder verändert werden, ohne dass sie vollständig neu geladen werden muss oder auf Nutzerinteraktion angewiesen ist. So lässt sich ein einer Desktopapplikation ähnliches Verhalten bei Webseiten erzeugen. Eine typische Anwendung ist z.B. ein Liveticker. Die großflächige Verbreitung von modernen Webapplikationen wurde erst durch den massiven Einsatz von AJAX-Techniken möglich.

Es ist zu erwähnen, dass AJAX-Calls (aus Sicherheitsgründen) stets auf die eigene Domain beschränkt sind (Same Origin Policy) und nicht beliebig von fremden Seiten Daten (z.B. sensible Nutzerdaten) anfordern dürfen. Ein Aushebeln dieses Sicherheitskonzeptes (XSS - Cross Site Scripting) wird häufig beim Ausspähen von Daten angewandt.

JavaScript Frameworks und Bibliotheken

Sämtliche hier vorgestellten Bibliotheken sind optimal und für das Framework an sich nicht erforderlich. IE7 und ExCanvas werden per Standardeinstellung geladen, um die Kompatibilität zu älteren Internet Explorern zu erhöhen. jQuery und Infovis werden von den Spielimplementierungen dynamisch nachgeladen.

Die Verwendung eines Frameworks bietet mehrere Vorteile:

- wesentlich schnellere Entwicklung,
- getestete crossbrowser Kompatibilität,
- leichtere Zugänglichkeit des Quellcodes durch kompaktere Notation,
- höhere Portabilität.

[48] JSON, JavaScript Object Notation, http://www.json.org/

Der Hauptnachteil, geringere Flexibilität durch die Vorgaben der entsprechenden Bibliotheken, entfällt in diesem Fall, da die Infovis Sunburst Visualisierung ziemlich genau den Anforderungen des Spiels entspricht und keiner größeren Anpassung bedarf.

IE7-JS & Excanvas

Um ein komfortableres Programmieren zu ermöglichen, lädt das Framework vor der eigentlichen Ausführung zwei JavaScript Bibliotheken:

IE7-JS	(HTML, CSS und PNG Fixes)
ExCanvas	(HTML5 Canvas Fix)

Für das Framework an sich sind diese Fixes nicht erforderlich, sie werden aber standardmäßig (nur vom Internet Explorer) geladen, um ein konsistenteres Programmverhalten zu ermöglichen.

Da der Internet Explorer bei der Implementierung von Webstandards eine Sonderrolle einnimmt, ist es unumgänglich, einige Kompatibilitätsanpassungen vorzunehmen. Dies kann in Form von Browserweichen (Erkennen des Browsers und Ausliefern von spezialisiertem Code) oder durch das softwareseitige „Nachrüsten" von Features und Normkompatibilität erfolgen. Die beiden hier erwähnten Bibliotheken leisten eben dies für ältere Internet Explorer.

IE7.js erweitert den Internet Explorer Version 5 und 6 dahingehend, dass das Rendern von HTML und CSS eher den Standards entspricht und fügt einige neue Features hinzu, wie eine korrekte Darstellung von transparenten, PNG-kodierten Bildern.

ExCanvas ermöglicht die (im HTML5 Standard vorgesehene) Verwendung des Canvas-Tags für Darstellung von 2D Grafiken.

jQuery

jQuery ist eines der am häufigsten genutzten Frameworks für JavaScript. Sein Einsatz hat sich in vielen großen und prominenten Webapplikationen bewährt[49], es wird stetig weiterentwickelt und auf andere Plattformen portiert (zum Beispiel auf Mobile Geräte[50]). Die Hauptaufgabe liegt in effizienten DOM (Document Object Model) Operationen, Event-handling und der Bereitstellung von konsistenter und Browser-unabhängiger JavaScript Funktionalität. Zusätzlich zu diesen eher technischen Eigenschaften verfügt jQuery über ein Repertoire von Funktionen zur Animation von HTML-Elementen. Viele webgestützte Bildbetrachter, Menüs und Grafikeffekte, die in reinem JavaScript mehrere hundert Zeilen Quellcode benötigen würden, lassen sich mit jQuery äußerst kompakt und leicht verständlich implementieren, sodass die Einstiegshürde auch für Fachfremde (zum Beispiel Designer) sehr tief liegt.

[49] jQuery wird unter anderem genutzt von: Google, Dell, Bank of America, Digg, NBC, CBS, Netflix, Mozilla, http://jquery.com/

[50] jQuery Mobile, http://jquerymobile.com/

Infovis

Zur Visualisierung wird unter anderem das JavaScript Framework Infovis[51] verwendet (Infovis Sunburst). Die Infovis Bibliotheken stellen eine Vielzahl von verschiedenen Visualisierungsmethoden und Hilfsfunktionen bereit. Sie erfreuen sich in der Forschung (vor allem auf dem Gebiet der Netzwerk- und Graphen-Forschung) hoher Verbreitung, sodass vorhandene Daten leicht auch auf andere Visualisierungen portiert und dargestellt werden können (z.B. bei der Auswertung der Daten).

Dem Spiel sind weitere Infovis Implementierungen beigefügt (Spacetree, Hypertree), um das dynamische Überschreiben der Visualisierung zur Laufzeit zu demonstrieren.

CSS3 - Cascading Style Sheets 3

Cascading Style Sheets[52] sind eine grundlegende Technik in modernen Webseiten. Sie erlauben die Daten von der Darstellungsstruktur zu trennen und so zum Beispiel Webseiten für verschiedene Ausgabegeräte unterschiedlich zu rendern. CSS beschreibt ein Layout, nach dem die zugrundeliegenden Daten angeordnet werden, ohne dass auf sie selbst Einfluss genommen wird.

Das Framework nutzt CSS3 Effekte wie Schatten oder Gradienten (Farbverläufe). Diese sind allerdings rein optisch und nicht funktional, d.h. das Framework ist auch ohne CSS3 Unterstützung komplett lauffähig und abwärtskompatibel mit den Vorgängerversionen (CSS Level 2 Rev 1, CSS Level 1).

Durch die CSS3 Effekte kommt das Framework, trotz ansprechender Grafiken, ohne statische Bilddateien aus.

3.3 Python

Die Programmiersprache Python legt besonderen Wert auf Syntax-Struktur und Lesbarkeit des im Vergleich zu anderen Sprachen oft kompakten Quellcodes. Sie unterstützt verschiedene Programmier-Paradigmen (Objektorientierung, Imperative Programmierung, Elemente von Funktionaler Programmierung) und verfügt von Haus aus über eine umfangreiche Standard-Bibliothek, die vielfältige Funktionalitäten bereitstellt.

Python Code wird zu Bytecode kompiliert und dann in einer virtuellen Maschine ausgeführt (ähnlich zu Java). Python ist bereits sehr früh zu Multiplattform-Verwendung hin entwickelt worden und wird bei vielen Linux Distributionen mitgeliefert (FreeBSD, OpenBSD, Ubuntu), sowie bei Mac OS X.

Für Windows existiert sogar die Möglichkeit, Python Anwendungen in Form von ausführbaren Dateien (Executables) zu packen / kapseln[53]. Die Einsatzfähigkeit von Python ist ebenfalls sehr breit und reicht von der Verwendung als Scriptsprache für Anwendungen wie Blender, GIMP oder Paint Shop Pro über normale Desktopanwendungen, wie dem Ubiquity Installer oder dem originalen Bittorrent Client, bis hin zur Wahl als erste Programmiersprache für die Google Cloud (GAE).

---

[51] Infovis, http://thejit.org

[52] W3C: Cascading Style Sheets, http://www.w3.org/Style/CSS/

[53] Python to Executable, http://www.py2exe.org/

3.4 REST Architektur - Representational state transfer

Representational state transfer beschreibt eine Architektur, in der Clients eine Ressource, repräsentiert in Form eines Dokuments (oder etwas Vergleichbarem), von Servern anfordern. Mit Erhalten der Antwort wechselt der Client in einen neuen Zustand.

REST ist nahe an HTTP angelehnt (aber nicht darauf beschränkt) und sollte den folgenden Anforderungen entsprechen:

- Client / Server: Eindeutige Aufgabentrennung, beispielsweise Speicherung der Daten im Server, Benutzerinterface im Client.
- Zustandsloses Protokoll: Es wird kein Clientzustand im Server gespeichert. Anfragen (Requests) enthalten sämtliche benötigten Informationen, damit der Server diese bearbeiten kann.
- Schnittstellen müssen eindeutig identifizierbar sein (ID, meist in Form einer URL).
- Cachebarkeit von Ressourcen im Client muss bedacht und evtl. unterstützt / verwaltet werden.

Als Webservice wird außerdem eine feste Zuordnung von Methoden zugesichert:

- GET : Anfordern einer Ressource
- PUT : Ersetzen einer vorhandenen Ressource
- POST : Erzeugen einer neuen Ressource
- DELETE : Entfernen einer Ressource

Die erhofften Effekte sind eine reduzierte Komplexität durch Standardisierung und hohe Skalierbarkeit.

3.5 Comet Architektur

Der Begriff „Comet“[54] fasst mehrere Technologien zur webbasierten, bidirektionalen Kommunikation zusammen. Während der traditionelle Web-Ansatz und Ajax stets auf einem vom Client initiierten Datenaustausch aufsetzen, ermöglichen Comet-Techniken auch eine vom Server ausgehende Verbindung. Eine Echtzeitkommunikation beziehungsweise ereignisorientierte Programme sind ohne diese Möglichkeit nur sehr ineffizient zu realisieren (zum Beispiel durch Polling, also ständiges, aktives Abfragen des Servers).

Ursprünglich für HTML vorgesehene Kommunikation:

Client → Anfrage → Server

Server → Antwort → Client

[54] Comet Architektur, http://en.wikipedia.org/wiki/Comet_(programming)

Es bestehen verschiedene Möglichkeiten diese Limitierung zu umgehen, die auch unter den Namen „Server Push“ (im Gegensatz zum „Polling“), „long Polling“ oder „http Streaming“ bekannt sind. Realisiert wird diese Funktionalität durch künstliches Erhalten oder wenigstens langes Offenhalten der http-Verbindung:

long Polling: Der Client initiiert die Verbindung zum Server, die aber nicht direkt bearbeitet, sondern offen gehalten und erst bei Bedarf (z.B. beim Auftreten eines Events) durch den Server beantwortet wird. Daraufhin öffnet der Client erneut die Verbindung, um weitere Ereignisse (Events) oder Daten empfangen zu können.

Streaming: Permanente Verbindung durch „unendlich lange“ Datenblöcke, z.B. durch Erzeugen von offenen iFrames, die sukzessive mit JavaScript gefüllt und vom Client ausgewertet werden.

3.6 SOA: Service Orientated Architecture

Dienst (Service) orientierte Architekturen[55] sind vor allem im Bereich der Webservices anzutreffen. Zielsetzung ist die Modularisierung, Trennung und Kapselung von einzelnen, spezialisierten Diensten, um eine einfachere Koordinierung und hohe Wiederverwendbarkeit (durch Neukombination von bereits vorhandenen Diensten) zu erreichen.

Ähnlich wie in der Objektorientierung stellt ein Dienst nach außen klar definierte Schnittstellen zur Kommunikation bereit, ist aber in sich geschlossen und hat keine Seiteneffekte auf andere Dienste. Die Services sind idealerweise vielseitig einsetzbar und können untereinander kommunizieren (über Standards wie XML-RPC, eXtensible Meta Language Remote Procedure Call oder JSON). Eine vollständige Webanwendung entsteht durch das Zusammenspiel von mehreren Webservices.

Aufgrund eines möglicherweise entstehenden Overheads durch die Aufspaltung und Kommunikation der Dienste untereinander empfiehlt sich eine serviceorientierte Architektur erst für Anwendungen ab einer gewissen Größe beziehungsweise für Plattformen, die mehrere Anwendungen umfassen und so zu Synergieeffekten führen.

[55] Till Rausch: Service Orientierte Architektur Übersicht und Einordnung, 2006 ; Webarchive.org (vom 10. Oktober 2008), bit.ly/v0ya3O

4. Design & Technik

Das Framework bildet ein Netzwerk von Knoten mit einer Menge von Eigenschaften ab. Jeder Nutzer „besitzt“ einen oder mehrere ihm zugeordnete Knoten, deren Eigenschaften er verändern kann. Diese Veränderungen werden in Echtzeit zurück ins Netzwerk (und damit auch zu den anderen Benutzern) propagiert. Die Kernfunktion des Systems leistet die technische Verwaltung und sichert die Konsistenz des für alle Nutzer globalen Zustands. Um das Netzwerk zu verändern, stellt das Framework verschiedene Funktionen bereit, zum Beispiel das Austauschen zweier Knoten oder das Verändern der Eigenschaften eines Knotens.

In der konkreten Spielimplementierung stellt das Netzwerk den globalen Spielzustand dar. Die Aufgabe des Nutzers (Spielers) ist es, die Eigenschaften der eigenen Knoten durch Einflussnahme an eine vorgegebene Zielfunktion anzunähern, beispielsweise die Farbe des eigenen Knotens der Umgebung anzugleichen oder in kollaborativer Arbeit ein Bild zu zeichnen.

Das Spiel implementiert dazu vom Framework vorgegebene Interfaces, die das Aussehen der Benutzeroberfläche (GUI, Grafical User Interface) sowie eine Zielfunktion in Form einer Aufgabenstellung festlegen.

Dieses Kapitel enthält nur Auszüge von Funktionen und Variablen. Der Quellcode einiger ausgewählter Funktionen befindet sich im Kapitel „Details zur Implementierung“, eine vollständige Liste der technischen Dokumentation in „7.2 Technical Documentary“.

4.1 Cloud Client Comet Design

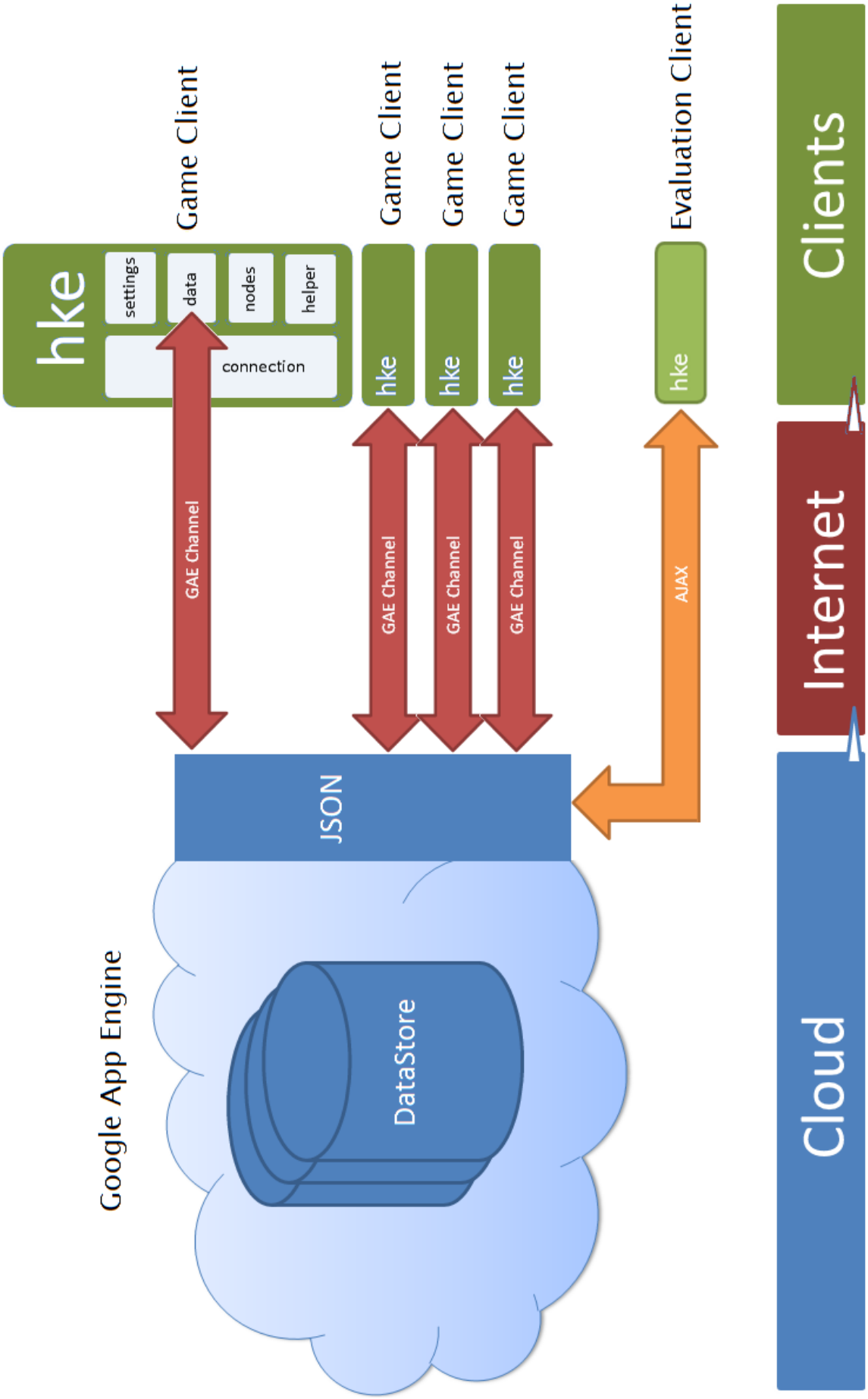
Ursprünglich war eine Client Server Architektur nach RESTful Richtlinien vorgesehen. Diese erlaubt allerdings keine asynchrone 2-Wege Kommunikation, die benötigt wird, um in Echtzeit Updates zu allen Clients zu versenden und zu empfangen und musste darum verworfen werden. Diese Anforderung erfüllt das Comet Design, das die GAE als Long Polling in Form von Channels bereitstellt. Jeder Client baut eine Channel Verbindung zum Cloud-Speicher auf und ermöglicht so zu jedem Zeitpunkt eine bidirektionale Kommunikation.

Das Design ist hauptsächlich durch die Anforderung der Skalierbarkeit und Robustheit der Anwendung geprägt und besteht im Wesentlichen aus drei unterschiedlichen Teilen:

- Globaler Zustand in der Cloud: Aktueller und vergangene Spielzustände
- HTML5 und JavaScript basierter, intelligenter Spiel-Client
- Evaluierungssoftware

Cloud und Client kommunizieren asynchron und nahezu in Echtzeit. Die Evaluierungssoftware ist vom eigentlichen Spiel entkoppelt, läuft getrennt davon und kann in anderen Programmiersprachen implementiert sein. Da die Spiellogik dezentral auf den Clients läuft ist es denkbar, auch komplexe Simulationen mit hoher Arbeitsanforderung (z.B. Interpolation des Spielzustands in die Zukunft) auszuführen, ohne dass die zentrale Architektur selbst einer Mehrbelastung ausgesetzt wäre.

Grafik der Architektur



4.2 Globaler Zustand in der Cloud

Der globale Zustand wird in einer Cloud gespeichert, deren Funktionalität bewusst stark eingeschränkt und auf reines Daten-Management begrenzt ist (Data Cloud).

Die rechenintensiven Veränderungen an den Daten (z.B. Austauschen von Knoten) werden vom Client vorgenommen und brauchen danach nur noch in der Cloud abgelegt zu werden. Die entstehende Rechenlast wird also von der Cloud zum Client hin ausgelagert. Die Cloud-Technologie stellt außerdem genug Speicherplatz zu Verfügung, um die jeweiligen Spielzustände über die gesamte Spieldauer zu archivieren. So wird bereits während des laufenden Spiels oder danach eine lückenlose Analyse gewährleistet.

Die Speicherschnittstelle ist bewusst einfach gehalten und stellt außer Daten keine weiteren Funktionen zur Verfügung. Sie ist lediglich eine „nicht intelligente“ Speicherschnittstelle ohne eigene Logik und besitzt darum ein besonders hohes Skalierungspotential.

Die Cloud gibt für jeden Benutzer die für ihn relevante Knotenmenge als JSON Datensatz aus, der dann im Client visualisiert wird. Zusätzlich wird eine Channel Verbindung aufgebaut, über die geänderte Knoten in Echtzeit zum Client gepusht und vom Client angenommen werden können.

Aufbau und Funktionen

Die Schnittstelle der Data-Cloud ist generisch gehalten und verfügt zum Teil sogar über redundante Daten (siehe „DataStore“). Die Knoten lassen sich zu beliebigen Netzwerken verknüpfen und in jedem Knoten können beliebige Daten abgespeichert werden.

In der Standardverwendung ist keinerlei Anpassung der Cloud für verschiedene Spielimplementierungen notwendig, allerdings können zwei verschiedene Spiele nicht in derselben Instanz einer Data-Cloud laufen (aufgrund der fortlaufenden Nummerierung der Knoten-IDs und eindeutigen Zuordnung von Knoten zu einem Benutzernamen, der möglicherweise in beiden Spielen vorhanden ist). Ein Parallelbetrieb in mehreren Instanzen ist selbstverständlich möglich.

Eine Veränderung der Cloud-Einstellungen oder Funktionen ist nur notwendig, wenn die Art des Netzwerks beeinflusst werden soll. Es ist beispielsweise möglich, den Aufbau des Netzwerks mit bereits einprogrammierten („hardgecodeten“) Daten fest vorzugeben (start_static()) oder die Generierung der Startdaten zu beeinflussen (start_algo()).

Nach außen stellt die Cloud mehrere Schnittstellen zur Verfügung, wobei nur zwei für den eigentlichen Betrieb benötigt werden:

„ / “ : Login- und Spieleseite
„ /channel “ : GAE Channel Schnittstelle für Echtzeitkommunikation.

Die anderen dienen Evaluationszwecken und werden vom normalen Benutzer nicht benötigt:

„ /download “ : Download Schnittstelle für gespeicherte Daten
„ /evaluation “ : JavaScript Evaluationsclient für die „download“ Schnittstelle
„ /test “ : Stresstest Client um Last zu simulieren
„ /del “ : Reset Funktion, um sämtliche Daten zu löschen, nur zu Debug-Zwecken

Unter „Daten und Funktionen“ werden die entsprechenden Funktionen genauer vorgestellt.

```
application = webapp.WSGIApplication(
                                     [('/', default),
                                      ('/channel', GAEchannel),
                                      ('/download', download),
                                      ('/evaluation', evaluation),
                                      ('/test', test),
                                      ('/del', delete)],
                                     debug=True)
```

Von der Cloud bereitgestellte Urls

Einstellungen

Eine Anpassung dieser Einstellungen ist optimal und für die meisten Spielimplementierungen wahrscheinlich nicht erforderlich.

„maxLayers“ legt die Anzahl der von der Cloud ausgegebenen Ebenen von Knoten aus. Grundsätzlich werden jedem Nutzer der ihm zugeordnete Wurzel-Knoten (Root) sowie die hier definierten Schichten von Kind-Knoten ausgegeben. Bei

maxLayers = 0	wird der Wurzel-Knoten, bei
maxLayers = 1	Wurzel- + Kind-Knoten, bei
maxLayers = 2	Wurzel- + Kind- + Kindeskind-Knoten

etc. ausgegeben. Eine unnötig hohe Wahl von „maxLayers“ führt zu redundanten Datenbankabfragen und Datentransfer (Traffic).

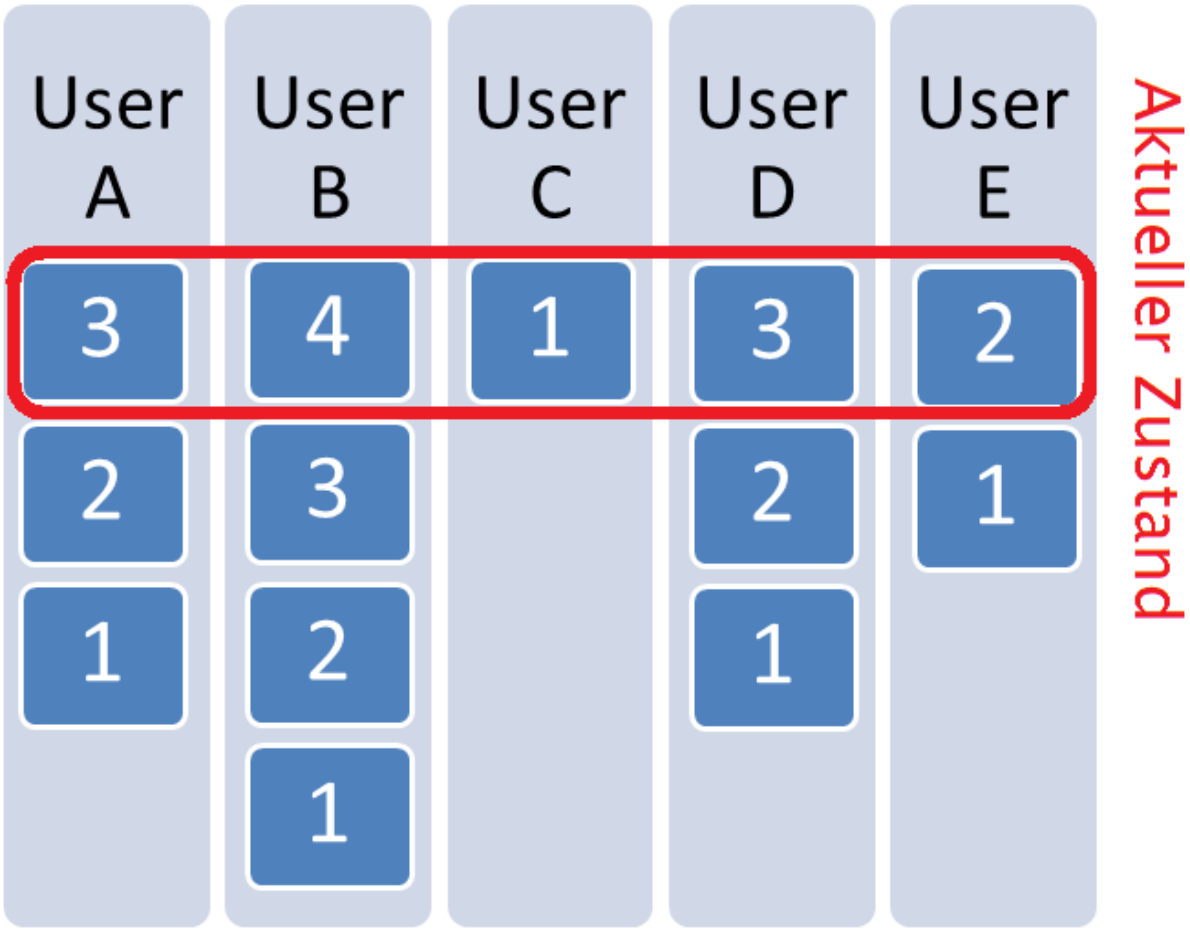
Über die Variable „startData“ lässt sich die Generierung von Startdaten in der Cloud erzwingen. In der Standardeinstellung werden diese Daten von den Clients erzeugt. Falls aktiv, lassen sich Startdaten entweder komplett statisch (start_static erzeugt im vornherein vom Programmierer festgelegte “hardgecodete” Startdaten) oder algorithmisch (start_algo erzeugt algorithmisch Startdaten) festlegen.

Daten / Funktionen

Das in der Cloud abgelegte Netzwerk wird in Form von einzelnen Knoten, bestehend aus einer Id, einem Besitzer („name“), Inhalt („content“), Nachbarn (adjazente Knoten, „peers“), Anzahl der Nachbarn („peercount“, vgl. 5.2.1.3 DataStore) sowie Datum und Zeit („date“), gespeichert.

```
class Data(db.Model):
    #store the game data
    id = db.IntegerProperty(required=True)         #id, integer, ascending
    name = db.StringProperty()                     #name (email)
    content = db.StringProperty()                  #data
    peers = db.StringProperty(default="")          #list of ids, separated by ","
    peercount = db.IntegerProperty(default=0)      #see 5.2.1.3 DataStore
    date = db.DateTimeProperty(auto_now_add=True) #date + time
```

Ein Knoten kann zwar neu erzeugt, verändert und aus dem Netzwerk entfernt (von allen anderen Knoten getrennt) werden, allerdings bleiben seine alten Daten stets erhalten. Jede der obigen Veränderungen am Netzwerk erzeugt eine aktuellere, geupdatete Version des alten Knotens. Der dann veraltete Originalknoten bleibt unverändert gespeichert.



Alte Knoten bleiben gespeichert, für den aktuellen Zustand wird jeweils der neuste zurückgeliefert.

Bei Anfragen wird stets die aktuelle Version ausgegeben. Nach einem Update bzw. einer Änderung wird der neue Knoten an all diejenigen Nutzer gesendet, in deren Sichtbereich er liegt.

Sollte, durch eine kritische Wettlaufsituation (Racing Condition), eine Dateninkonsistenz entstehen, so wird diese automatisch von der zuletzt ausgeführten, konkurrierenden Aktualisierung behoben. Jede Veränderung des Netzwerks wird in Form der vollständigen Menge aller geänderten Knoten gespeichert und beim Abschluss an alle betroffenen Clients kommuniziert, die daraufhin den Zustand nach der letzten Änderung annehmen.

Wurde eine konkurrierende Anweisung zuvor ausgeführt, wird diese im betroffenen Client mit Abschluss der zuletzt ausgeführten überschrieben und die Konsistenz wieder hergestellt. Im konkreten Beispiel eines Wettlaufs zweier konkurrierender Anweisungen auf derselben Knotenmenge, würde der betroffene (und unterlegene) Nutzer dies als einen „Glitch“ (Störung), in Form von Nicht- oder nur partieller Ausführung seines letzten Befehls wahrnehmen.

```
class Connections(db.Model):
    #store the connected clients
    name = db.StringProperty(required=True)        #name (email)
    chan = db.StringProperty(required=True)        #chan
    allpeers = db.StringProperty(default="")       #list of all peers this node sees
```

Die Connections-Tabelle verwaltet sämtliche aktuell per Channel zur Cloud verbundenen Clients. „allpeers“ ist eine Auflistung (Komma-getrennte Knoten IDs) von allen Knoten, die im Sicht- oder Einflussbereich des Nutzers „name“ liegen. Wird einer dieser Knoten modifiziert, so muss über den Channel „chan“ ein Update an den Client mit den aktualisierten Daten gesendet werden.

Folgende Klassen rendern HTML über eine Template-Funktion:

„default“	Login- (/files/name.html) oder Spielseite (/files/index.html),
„evaluation“	Evaluations-Seite (/files/evaluation.html) und
„test“	Stresstest-Seite (/files/test.html)

Die „gaeChannel“ Klasse implementiert die Channel-Schnittstelle der Google App Engine. Neue oder aktualisierte Daten werden hier in die Cloud eingespeist und direkt an betroffene Clients weiter gepusht. Der folgende Programmcode iteriert über sämtliche aktiven Channels („cons“) und testet, ob der Client ein Update benötigt (Boolean „doupdate“). Dies ist entweder der Fall, wenn der Client das Update selbst ausgelöst hat:

```
        if con.name == str(firstname)
```

oder einer der geupdateten Knoten im Sichtbereich beziehungsweise in der Knotenmenge des entsprechenden Clients liegt:

```
        if con.allpeers.split(",") in allids
```

Die Cloud versucht nun, die neuen Daten an den Client zu senden. Bei Misserfolg (zum Beispiel ausgelöst durch einen verwaisten Channel, der auf Clientseite geschlossen wurde) wird die Verbindung auch serverseitig gelöscht.

```
for con in cons:
    #if an updated node is visible for the user, update its data
    doupdate = 0
    if con.name == str(firstname):
        doupdate = 1
    for checkid in con.allpeers.split(","):
        if checkid in allids:
            doupdate = 1
    if doupdate:
        try:
            channel.send_message(con.name, nodes(con.name))
        except KeyError:
            #if sending through channel doesnt work, delete it
            q=db.GqlQuery("SELECT * FROM Connections WHERE name = '"+con.name+"'")
            results = q.fetch(10,0)
            db.delete(results)
            self.response.out.write("error")
```

Download

Es gibt ein weiteres JSON-Interface zum Zugriff auf die in der Cloud gespeicherten Daten, das von der Funktion „download“ implementiert wird. Es kann mit dem Evaluations-Client oder einer beliebigen anderen, internetfähigen Software abgefragt werden.

```
class download(webapp.RequestHandler):
    #download data (used by the evaluation client)
    def get(self):
        if self.request.get('limit'):
            limit = self.request.get('limit')
        else:
            limit = 100
        if self.request.get('start'):
            start = self.request.get('start')
        else:
            start = 0
        q = db.GqlQuery("SELECT * FROM Data ORDER BY date LIMIT "+start+", "+limit)
        results = q.fetch(int(limit),0)
        out='{"data": ['
        for res in results:
            out+='''{
                "content": '''+res.content+''',
                "id": '''+str(res.id)+''',
                "name": '''+res.name+''',
                "peers": '''+res.peers+'''
                },'''
        out+=']}'
        out = out.replace("  ","").replace(" ","").replace("\t","").
              replace("\r","").replace("\n","").replace("},]," , "}],").
              replace("},}","}}")
        self.response.out.write(out)
```

Cloud Schnittstelle zum Herunterladen der erzeugten Daten

Die Funktion überprüft, ob die Parameter „limit“ (Integer) und „start“ (Integer) gesetzt sind. Ist dies nicht der Fall, werden Standardwerte („limit“ = 100, „start“ = 0) verwendet.

Danach werden Knoten, beginnend mit (einschließlich) Datensatz „start“ und Anzahl „limit“, sortiert nach Datum, als JSON ausgegeben. Im Gegensatz zur GAE Channel Schnittstelle werden allerdings keine geschachtelten Kind-Knoten übertragen.

Existieren weniger Datensätze, wird nur die entsprechende Anzahl zurückgegeben. Um Bandbreite zu sparen werden sämtliche nicht funktionalen Whitespaces und Steuerzeichen entfernt. Wird die vorletzte Zeile der Funktion auskommentiert, so gibt sie, anstatt eines komprimierten Strings, korrekt eingerücktes und besser lesbares JSON zurück.

Zu Debugzwecken existiert mit der Klasse „delete“ die Möglichkeit, sämtliche gespeicherten Daten (in „Data“ und „Connection“) zu löschen. ACHTUNG, keine Sicherheitsabfrage! In einer Produktivumgebung kann die Klasse leicht durch das Entfernen des Kommentaroperators in Zeile 2 entschärft beziehungsweise deaktiviert werden.

```
    #return false
```

Die übrigen Funktionen dienen als Hilfsfunktionen für algorithmisches Erzeugen von Startdaten. „randomColor“ erzeugt einen zufälligen Farb-String (#000000 - #ffffff), „connectNodes“ verbindet zwei bisher unverbundene Knoten. „nodes“ liefert die Knoten von Spieler “Name“ sowie sämtliche Knoten in seinem Sichtbereich (Nachbarn) zurück.

Data Store

Zur Speicherung der Daten wird die Data Store Schnittstelle der GAE genutzt. Hierbei handelt es sich nicht um eine relationale Datenbank im konventionellen Sinne, sondern um eine speziell an die Skalierungsbedürfnisse einer Cloud angepasste Datenbank. Dies geht mit einigen Besonderheiten einher, von denen exemplarisch einige hier erwähnt werden sollen:

In der Anwendung ist es an mehreren Stellen erforderlich, die aktuelle Knotenanzahl zu ermitteln (z.B. um eine Überschreitung der maximalen Nachbarknoten zu verhindern). Diese Information ist in der Liste der adjazenten Knoten bereits enthalten, eine weitere Ablage im Feld PEERCOUNT erzeugt also Redundanz.

Mit einem SQL „LIKE“ Query in Kombination mit Wildcards lässt sich diese Information effizient und ohne weitere, redundante Felder abfragen:

```
SELECT * FROM 'Data' WHERE 'peers' LIKE '%,%,%,%,%';
```

Die Abfrage liefert nur Knoten mit n >= 5 Nachbarn zurück. Der Data Store erlaubt aber keine unscharfen „LIKE“ Queries, sodass alle Datensätze per SQL geladen und danach mit Python (oder einer entsprechenden anderen Programmiersprache) gefiltert werden müssten. Das neue Feld PEERCOUNT erlaubt einen genauen Test

```
SELECT * FROM 'Data' WHERE 'peercount' = 5;
```

der gewünschten Eigenschaft auf Ebene des Data Stores.

Ähnliches gilt für die „COUNT“ Expression.

```
SELECT COUNT(*) FROM 'Data';
```

quittiert die GAE mit einer Fehlermeldung. In der ersten Version der GAE war eine derartige Abfrage nicht ohne explizites Einfügen eines eigenen Zähl-Schlüssels oder der Iteration über die gesamte Tabelle möglich.

Seit Version 1.3.6 ist ein unbegrenzter Datastore count() – Query möglich (vorher begrenzt auf eintausend Datensätze). Die folgende Abfrage liefert also erst ab Version 1.3.6 der GAE ein korrektes Ergebnis:

```
all = Data.all()
count = all.count()
```

Die stark skalierbaren, Cloud-basierten Datenspeicher, wie der DataStore der GAE, stellen eine andere Architektur dar als die konventionellen, relationalen Datenbanken wie MySQL. Informationen und Abhängigkeiten lassen sich nicht 1:1 in die Cloud „heben“, sondern müssen gegebenenfalls von Grund auf neu entworfen werden.

Diese Limitierungen sind weniger dem jungen Alter dieser Technologien geschuldet, als vielmehr der neuen Ausrichtung der zugrundeliegenden Architektur und ihrer neuen Paradigmen, hin zu mehr Skalierbarkeit und dem Operieren auf gigantischen Datensätzen und Tabellengrößen.

4.3 Smart Client: HTML5 / JavaScript

Der Spiel-Client, auf HTML5 und JavaScripts basierend, implementiert die komplette Spiellogik. Diese Herangehensweise macht einen eigenen (Spiele-) Server überflüssig und entlastet zusätzlich die Speicherschnittstelle in der Cloud. Bis auf den globalen Spielzustand enthält der Client sämtliche Daten und alle Funktionalitäten. Das Verwenden von weithin akzeptierten und in allen gängigen Browsern implementierten Standards ermöglicht maximale Reichweite bei gleichzeitiger Verwendung moderner und für die Spieler ansprechenden Technologien.

JavaScript stellt von Haus aus nur begrenzt Möglichkeiten zur sauberen Implementierung von Frameworks bereit, allerdings lassen sich einige notwendige Methoden wie zum Beispiel Namespaces simulieren. Durch einen Trick lässt sich in JavaScript sogar eine Klassen- beziehungsweise Interface-Objektorientierung darstellen[56], allerdings schafft diese Methode architekturbedingt einen Mehraufwand („Overhead“) an Programmcode und Komplexität, der nicht im Verhältnis zu seinem Nutzen steht.

Der gesamte Programmcode wird in einer anonymen Funktion gekapselt und dann als einzelnes Objekt bereitgestellt. Sämtliche Deklarationen geschehen im Namensraum der anonymen Funktion und haben darum keinerlei Einfluss auf den Code außerhalb.

```
(function(){
        var hkengine = {
            /* sämtliche Variablen & Funktionen */
        }
        if(!window.hke){window.hke=hkengine;}
})();
```

Dadurch wird ein „Verseuchen“ des globalen Namespace vermieden und der Einsatz mehrerer Frameworks und JavaScript Bibliotheken parallel möglich (konkret jQuery und Infovis).

Außerdem erlaubt die Kapselung als Objekt eine beliebig tiefe Schachtelung und damit Modularisierung zur besseren Aufteilung.

```
hke.connection.init();
```

`hke`	globales Objekt
`connection`	verbindungsspezifisches Objekt
`init()`	Funktion um neuen Channel zu initialisieren

Jede Funktion lässt sich so einem Vater-Objekt zuordnen, das wiederum zum Hauptobjekt des Frameworks gehört. Dieser Aufbau wird im folgenden Kapitel erläutert.

Die JavaScript Datei ist in doppelter Ausführung im Framework enthalten, sowohl als kommentierte und sauber eingerückte („files/hke.js“) als auch als komprimierte Version für den Produktiveinsatz[57] („files/hke_minified.js“).

[56] Programming to the Interface in JavaScript, http://knol.google.com/k/programming-to-the-interface-in-javascript-yes-it-can-be-done-er-i-mean-faked#

[57] Komprimiert mit dem JavaScript Packer http://jscompress.com/

Aufbau und Funktionen

Das Framework besteht aus einem zentralen Objekt (hke), das sämtliche Funktionen und Daten in sich vereint. Es ist in die folgenden sechs Bereiche aufgeteilt, die sich ihrerseits weiter unterteilen:

„settings", „data", „nodes", „gui", „connection" und „helper".

Aus strukturellen Gründen ist empfehlenswert, auch spielspezifischen Code und Variablen in den entsprechenden Bereichen abzulegen, beziehungsweise diese zu erweitern.

hke.settings

Settings beinhaltet die Einstellungen. Voreingestellt sind Default-Werte, die aber je nach Spiel von der Spielimplementierung überschrieben werden müssen, wie zum Beispiel die Länge einer Spielrunde (minutes) oder die Anzahl der Handlungen (actions).

Auch technische Einstellungen wie der Debugmodus zur Ausgabe von Fehler- und Statusmeldungen (debug) oder die maximale Anzahl von zugelassenen Nachbarknoten (maxPeers) und deren Schachtelungstiefe (maxLayers) lassen sich hier festlegen.

hke.data

In „data" ist der für den Nutzer sichtbare Teil des Netzwerks als JSON abgelegt. Der Wurzelknoten (Root) ist der Knoten des Nutzers, die adjazenten Knoten sind als Kind-Knoten in „children" gespeichert.

Ein einzelner Knoten ist folgendermaßen aufgebaut:

```
{
    "id": "1",                  //eigene id (muss einzigartig sein)
    "name": "nutzer@abc.de",    //Nutzer (Besitzer des Knotens
    "content" : "#333",         //beliebiger Content
    "peers": "2,3,4",           //ids adjazenter Knoten
    "children" : []             //kann mehrere Kind-Knoten enthalten
}
```

Adjazente Kind-Knoten können rekursiv bis zu einer Tiefe von maxLayers in „children" geschachtelt sein.

hke.nodes

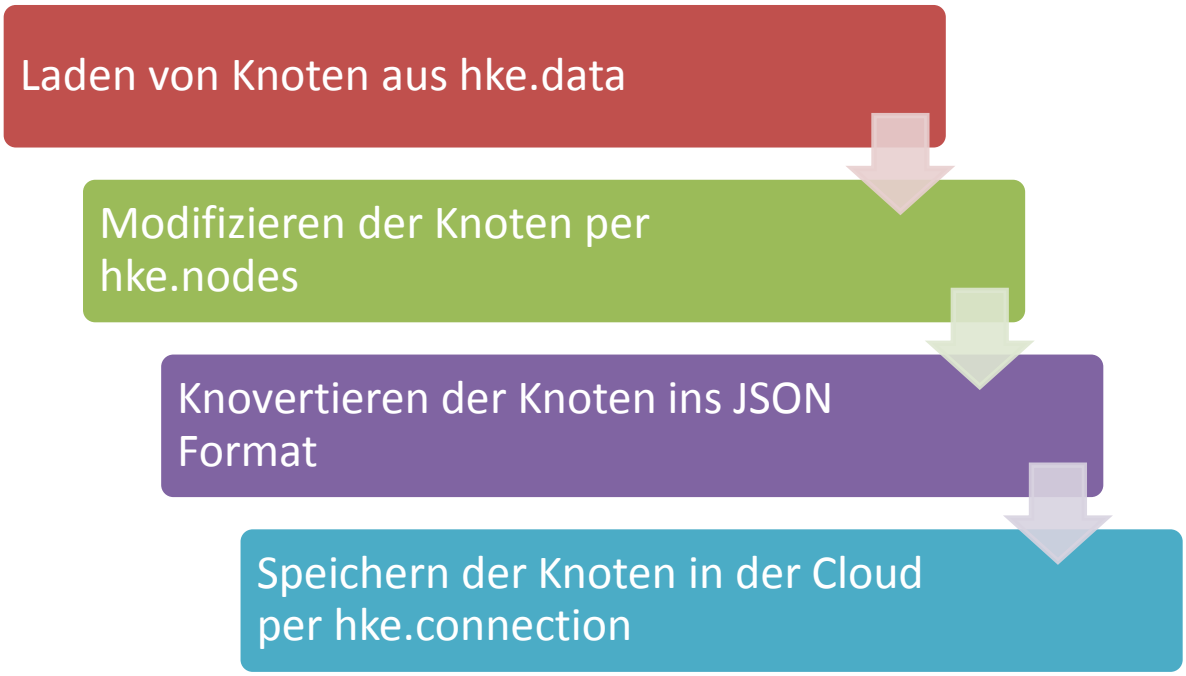
Nodes enthält alle Operationen, um das Netzwerk oder einzelne Knoten zu verändern. So lassen sich z.B. Knoten anlegen (create), verbinden (connectNodes), trennen (disconnectNodes), austauschen (switchNodes), löschen (removeNode), der Inhalt verändern (content) und einiges mehr.

Knoten werden dabei nicht als Objekt, sondern stets über ihre ID geladen. Alle Operationen geben ein Array von geupdateten Knoten (auch den von Seiteneffekten der Operation betroffenen) zurück, die dann per

hke.nodes.node2json(nodeArray)	ins JSON Format konvertiert und danach mit
hke.connection.sendMessage(data)	in der Cloud gespeichert werden können.

```
hke.connection.sendMessage(hke.nodes.node2json(hke.nodes.switchNodes("A","B")));
```

Beispiel für das Vertauschen von Knoten mit ID „A“ und ID „B“ und Speichern in der Cloud



Ablauf einer Knotenoperation

Kann oder darf eine Operation nicht ausgeführt werden (beispielsweise wegen Überschreitung von maxPeers, der Obergrenze von Nachbarknoten, oder weil die Operation bereits ausgeführt wurde), gibt sie anstatt eines Ergebnisarrays den Boolean „False“ zurück.

hke.gui

Das grafische Benutzerinterface (GUI) besteht aus zwei Teilen: Game (das Spielfeld) und Control (Kontrollelemente wie die Zeitanzeige für die Runden), die beide jeweils gleichnamige HTML-Divs[58] für die Anzeige der Oberfläche nutzen.

Das „control“-Div enthält die Anzeigen für die Rundenzeit („timer“-Div) und für die verbleibenden Handlungen („actions“-Div). Die Verwaltung und das Updaten dieser Anzeigen übernimmt das Framework automatisch (die Häufigkeit der Aktualisierung kann über die „updateInterval“ Variable konfiguriert werden).

Über die „msg“ Funktion lassen sich Nachrichten im Informationsbereich anzeigen (Einblendung oben auf der Seite).

[58] HTML „Div“-Tag, http://www.w3schools.com/tags/tag_div.asp

hke.connection

Connection regelt die Verbindung zur Cloud. Es gibt für die Clients zwei verschiedene Möglichkeiten, mit der Cloud zu kommunizieren:

GAE Channel:

Die asynchrone 2-Wege-Kommunikation kann zu einem beliebigen Zeitpunkt sowohl von der Cloud als auch vom Client genutzt werden, um Daten zum jeweils anderen zu transportieren. Nach ihrer automatischen Initiierung besteht sie permanent, bis zum Schließen der Anwendung.

```
function sendMessage(data) {…}
```

sendMessage(data) wird genutzt, um den String „data“ (normalerweise neue, JSON codierte Knoten) von den Clients zur Cloud zu senden. Diese werden beim Empfang gespeichert und (auch per Channel) von der Cloud an alle von Änderungen betroffenen Clients weitergesendet.

```
function receiveMessage(newData) {…}
```

Die receiveMessage(newData) Funktion empfängt die von der Cloud gesendeten JSON codierten Daten (String „newData“). Sie sollte von der Spielimplementierung überschrieben werden, um die empfangenen Daten weiterzuverarbeiten.

Ajax (ajax):

Sollen nur Daten aus der Cloud abgerufen werden, so ist es nicht nötig, dafür permanent eine Channel Verbindung offen zu halten. Per Ajax können die benötigten Knoten von der „/download“ -Schnittstelle abgerufen werden. Menge und Startknoten lassen sich dabei per

limit : Anzahl der Knoten
start : ID des ersten Knoten

definieren. Sind diese Parameter nicht gesetzt, wird auf Standardwerte (limit = 100, start = 0) zurückgegriffen.

hke.helper

In helper sind zusätzliche Hilfsfunktionen abgelegt. Sowohl Framework spezifische als auch „log“ zur Ausgabe von Debugmeldungen und „ready“, mit der sich per Callback Programmcode ausführen lässt, nachdem das DOM geladen wurde (ähnlich wie jQuery und andere JavaScript Frameworks). Per „init“ initialisiert sich das Framework nach erfolgreichem Laden automatisch selbst.

helper enthält aber auch allgemeine Funktionen wie das assoziative Sortieren, Testen und Kopieren eines Arrays, „loadScript“ zum dynamischen Nachladen von Daten und Code, eine Cookie-Verwaltung und eine Serialisierungs-Funktion für beliebige JavaScript Objekte (auch Arrays etc.).

Details zur Implementierung

Einige ausgewählte und technisch interessante Funktionen des Frameworks sollen hier näher beleuchtet werden.

Dynamisches Nachladen von Code und Daten

Die Helper Funktion hke.helper.loadScript(src) erlaubt das dynamische Nachladen von externen JavaScript Dateien, CSS Dateien oder JavaScript Code zu einem beliebigen Zeitpunkt.

```
loadScript : function(src){
    if (src.substr(-3,3).toLowerCase() == ".js" ){
        //load js file
        var newscript = document.createElement('script');
        newscript.setAttribute('type','text/javascript');
        newscript.setAttribute('src',src);
        document.getElementsByTagName('head')[0].appendChild(newscript);
    } else if (src.substr(-3,3).toLowerCase() == "css" ){
        //load css file
        var newscript = document.createElement('link');
        newscript.setAttribute('rel','stylesheet');
        newscript.setAttribute('type','text/css');
        newscript.setAttribute('href',src);
       document.getElementsByTagName('head')[0].appendChild(newscript);
    } else {
        //run js code
        eval(""+src+"");
        src = "js code";
    }
    hke.helper.log("loading script "+src);
}
```

Neue externe Dateien werden in der Header-Sektion des aktuellen HTML Dokumentes als neue DOM-Knoten eingefügt und dann vom Browser direkt eingebunden und ausgewertet. Übergebener JavaScript Code wird ausgeführt. Am Ende wird (falls aktiviert) eine Debugmeldung ausgegeben.

Durch das dynamische Nachladen lassen sich auch bereits vorhandene Variablen und Funktionen überschreiben, beispielsweise kann eine vorhandene Infovis Visualisierung (Sunburst) durch eine andere ersetzt werden (Hypertree, Spacetree), ohne dass die Seite neu geladen werden muss.

Vorhandene JavaScript Objekte können im Nachhinein neu angelegt, überschrieben oder erweitert werden.

Bei sehr großen externen Dateien ist zu beachten, dass das Parsen der neu geladenen Daten je nach Datenmenge unterschiedlich viel Zeit in Anspruch nimmt. Es handelt sich zwar um Verzögerungen im Millisekundenbereich, große Frameworks wie jQuery können jedoch nach dem Laden eventuell nicht sofort verfügbar sein. Diese Problematik lässt sich umgehen, indem abhängige Aufrufe verzögert ausgeführt werden, wie in der zweiten Spielimplementierung gezeigt:

```
//due to dynamic loading of jQuery(takes some time) this call has to be delayed
setTimeout( function() {
    hke.gui.info(hke.gui.text.welcome);
}, 1000 );
```

Diese Vorgehensweise ist nur bei Aufrufen direkt nach Einbinden der entsprechenden externen Quellen notwendig.

Channels

Leider gibt es keine Möglichkeit, im Server das Schließen einer Client-Verbindung festzustellen. Die gesamte Verwaltung (und Entfernung verwaister Client-Verbindungen) muss also serverseitig implementiert werden, da sonst mit steigender Spieleranzahl ein Overhead durch veraltete Channels entstehen könnte.

Nicht jeder Nutzer wird explizit eine Abmeldefunktion, welche den Channel schließt, benutzen. Auch andere Ereignisse wie ein Neu-Laden des Browserfensters (Refresh) oder Schließen des Browsers (Close) sollten eine Abmeldung auslösen. Beim Verlassen der aktuellen Seite wird deshalb ein Ereignis (Event) ausgelöst, welches den Befehl zum Schließen des Channels auf Serverseite (GAE Cloud) sendet.

Hierbei müssen einige Dinge beachtet werden: So dürfen beispielsweise nicht einfach alle dem Benutzer zugeordneten Channels geschlossen werden, wenn er sein Browserfenster schließt, sondern nur der jeweils geöffnete. Werden alle Channels anhand des Benutzernamens gelöscht, kann beim Aktualisieren des Browsers der folgende Fehler auftreten:

Race Condition[59] zwischen veraltetem „close Channel“ und neuem „open Channel“ Befehl:

- Der Benutzer aktualisiert seinen Browser („Seite neu laden“, F5, etc.).
- Das „onClose“ Callback sendet den Befehl zum Löschen aller Channels des Nutzers an den Cloud Speicher.
- Die neue Seite wird dem Nutzer bereitgestellt, inklusive eines neuen Channels.
- Der (inzwischen veraltete) Befehl zum Schließen aller Channels trifft ein und schließt den aktuellen (neuen) Channel zum Benutzer. Die Datenverbindung geht verloren.

Stattdessen muss der Channel explizit per Channel-ID beendet werden. Der hier dargestellte Fehler ist nicht theoretischer Natur, sondern tritt beim Beenden des Channels per Benutzername häufiger auf.

```
<body onunload="hke.connection.closeChannel(channel_id)">
<script type="text/javascript">
    close : function(cid) {
       var xhr;
       if(window.XMLHttpRequest) {
           try { xhr = new XMLHttpRequest(); }
           catch(e) { xhr = false; }
       }
       else if(window.ActiveXObject) {
          try { xhr = new ActiveXObject("Microsoft.XMLHTTP"); }
          catch(e) { xhr = false; }
       }
       if (!xhr) return false;
       xhr.open('POST', '/channel', true);
       var msg = '{ "close" : "'+cid+'" }';
       hke.helper.log("closed channel to cloud");
       xhr.send(msg);
    }
</script>
```

connection.close erzeugt einen XMLHttpRequest[60] an die Cloud zum Schließen des Channels

[59] Wettlaufsituation, die unvorhergesehenes Verhalten auslöst, http://en.wikipedia.org/wiki/Race_condition

[60] XMLHttpRequest, Browser Api zur Hintergrundkommunikation, http://www.w3.org/TR/XMLHttpRequest/

jQuery

Zur schnelleren und „schöneren“ Entwicklung der Spielimplementierung wird auf jQuery zurückgegriffen. jQuery ist das wohl verbreitetste JavaScript Framework und leistet hauptsächlich hinsichtlich Browserkompatibilität und GUI Animationen gute Dienste. Für die Kernfunktionalität des Frameworks ist jQuery nicht erforderlich, allerdings lassen sich das GUI und grafische Effekte sehr effizient und schnell damit implementieren.

Die Spielimplementierung von Spiel 2 ersetzt das Ein- und Ausblenden von Informationsnachrichten durch ein animiertes „Einschieben“ der Nachrichten, indem die Standardfunktion durch die folgende überschrieben wird:

```
hke.gui.info = function(msg){
    msg+="<span style='float:right;'>Click to close [x]   </span>";
    document.getElementById("info").innerHTML=msg;
    $("#info").html(msg).animate( {height: 80}, 1000, 'swing' );
    document.getElementById("info").onclick=(function () {
        $("#info").animate( {height: 0}, 1000, "swing" );
    });
    hke.helper.log("displayed top message");
}
```

Zeile 2 und 3 setzen den Inhalt für die Meldung.

Zeile 4 bis 7 überschreiben die Veränderung der Sichtbarkeit mit einer jQuery Animation, die grafisch ansprechender ist („Swing“ Animation[61]).

In Zeile 8 wird eine Debugmeldung ausgegeben.

4.4 Evaluations Client

Der Evaluations Client ist eine Software zur Auswertung der von der Anwendung erzeugten Daten. Er greift per Ajax auf die Download-Schnittstelle der Cloud zu und hat unbeschränkten Zugriff auf die Knoten aller Nutzer. Die Programmiersprache des Clients muss dabei nicht zwangsläufig HTML5/JS sein. Jede beliebige webfähige Sprache kann zur Implementierung verwendet werden.

Ein vorhandener JSON Parser ist vorteilhaft, um die bereitgestellten Daten weiter verarbeiten zu können (aber ebenfalls optional).

Für das Picture Drawing Game ist bereits ein HTML5/JS basierter Evaluationsclient enthalten. Dieser ruft die Knoten in im Client definierten Paketgrößen (Standard: 500 Knoten pro Anfrage) aus der Cloud ab und visualisiert die Entstehung und Veränderungen des Netzwerks in chronologischer Reihenfolge.

[61] jQuery Swing Animation, http://api.jquery.com/animate/

4.5 Spielimplementierungen

Das Framework an sich propagiert lediglich den in der Cloud gespeicherten globalen Zustand zu den Clients und stellt eine einfache, technisch motivierte Visualisierung dieser Daten bereit. Um das Framework sinnvoll und reichweitenstark einzusetzen, ist die konkrete Implementierung eines Spiels notwendig.

Diese Spielimplementierung soll dem Nutzer eine ansprechende Oberfläche bereitstellen, über die er mit dem Spielzustand (und damit mit den anderen Mitspielern) interagiert. Das Framework sieht hierfür eine Unterteilung in „game“ (Visualisierung der Daten) und „control“ (Steuerung) vor. Diese Trennung kann, wie alle Bereiche des Frameworks, von einer Spielimplementierung überschrieben oder vollständig andersartig aufgebaut werden.

Die Visualisierung ist für den Nutzer einer der wichtigsten Faktoren überhaupt, da sich ihm das Spiel über Grafik und Text darstellt.

Im Umfeld der sozialen Netze, in dem sich das Spiel befindet, sind hauptsächlich Gelegenheitsspieler und Casual Gamer anzutreffen, die auch die größte und Haupt-Zielgruppe darstellen. Hardcore Gamer machen nur einen geringen Teil aus. Insbesondere der deutsche Markt für Spiele in sozialen Netzen (Social Games) explodiert geradezu. Die Qualität und Quantität der konkurrierenden Spiele sowie die Erwartungshaltung der Nutzer sind sehr hoch.

Unter diesem Konkurrenzdruck ist es essentiell, dass auch unerfahrene Spieler nicht von einem zu komplexen Spielprinzip, im konkreten Fall der visuellen Darstellung des Spiels, abgeschreckt werden.

Die Anforderungen an die Visualisierung sind (in der Reihenfolge der Wichtigkeit):

- leichte Verständlichkeit und Zugänglichkeit,
- (regelmäßige) Interaktionsmöglichkeiten,
- Motivation des Spielers durch ansprechendes Design, z.B. Belohnen von Interaktion, Levelsysteme mit regelmäßigem Aufstieg, Fortschrittsbalken, Belohnungen (Achievements), Leistungsabzeichen (Badges).

Die Spiele können auf den Funktionen des Frameworks aufsetzen, kommen daher mit wenig Code aus und können entsprechend schnell und effizient entwickelt werden.

Zu Demonstrationszwecken sind bereits zwei einfache Spiele enthalten:

Beim ersten handelt es sich um ein netzwerkbasiertes Spiel, in dem jeder Spieler einen farbigen Knoten besitzt. Die Spieler können das Netzwerk verändern und zum Beispiel Knoten austauschen.

Im zweiten Spiel (Collaborative Picture Game) geht es darum, gemeinsam ein Bild zu malen. Jeder Nutzer ist dabei auf eine eigene Fläche begrenzt, sodass nur in kooperativer Arbeit das Ziel erreicht werden kann.

Welche Spielimplementierung geladen wird, lässt sich in der Datei „/files/index.html“ mit dem letzten JavaScript Aufruf festlegen:

```
        <!-- JS: load the game -->
        <script type="text/javascript" src="game2/game.js"></script>
```

Laden der Spielspezifischen Daten & Programmcodes

Spiel 1 nutzt die algorithmische Datenerzeugung in der Cloud und eine Knotentiefe von 3:

```
maxLayers = 3                   #depth (# of layers) of children peers
startData = 1                   #if true, the cloud generates the start data
```

Im Gegensatz zu Spiel 2, mit Datenerzeugung im Client und einer Knotentiefe von 1:

```
maxLayers = 1                   #depth (# of layers) of children peers
startData = 0                   #if true, the cloud generates the start data
```

Spiel 1 besteht lediglich aus dem Spiel-Programm an sich, für Spiel 2 (Collaborative Picture Drawing Game) sind auch eine Evaluationssoftware und ein Stresstest implementiert.

Überschreiben von Funktionen

Sämtliche Funktionen und Variablen des Frameworks können überschrieben werden, um sie zu modifizieren oder an eine Spielimplementierung anzupassen.

Einige Funktionen, wie das GUI oder das Laden von spielspezifischen Einstellungen, müssen vom Spiel implementiert werden. Initialisierungsfunktion des Picture-Spiels:

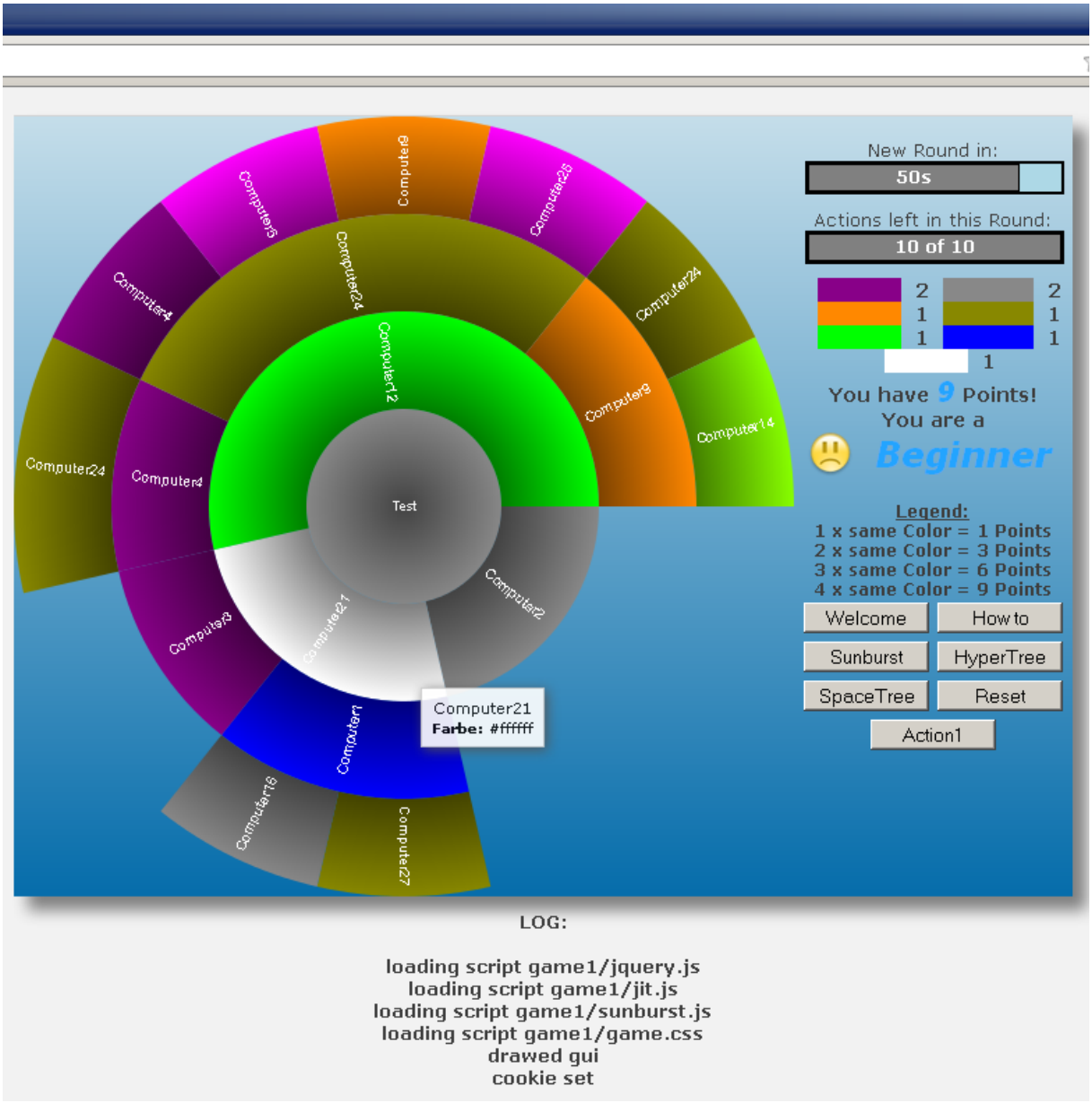
```
hke.helper.init = function(){
    hke.settings.actions = 5;         //5 actions per round
    hke.settings.minutes = 15;        //15min rounds
    hke.settings.maxPeers = 24;       //max peers= 5x5 = 24+root
    hke.settings.maxLayers = 1;       //load 1 layer of peers only (contains all 24)
    hke.settings.tiles = 5;           //# of tiles in 1 line. total number = tiles^2
    /*playfield is 500x500px, size of tiles is automatically adjusted*/

    var scripts = new Array(          //load the needed files
        "game2/jquery.js",            //load jquery to beautify some animations
        "game2/game.css"              //laod css file for the game
    );
    for (var sc in scripts) hke.helper.loadScript(scripts[sc]);

    hke.gui.init();                  //initialize gui
    hke.gui.title("Picture Game");   //display custom game title
}
```

Im ersten Abschnitt werden die Einstellungen des Spiels festgelegt, danach die benötigten zusätzlichen Dateien geladen (die jQuery JavaScript Bibliothek und eine Datei, die CSS Anweisungen enthält).

Mit dieser Methode lassen sich zu jedem beliebigen Zeitpunkt dynamisch JavaScript Code und CSS Dateien nachladen und ausführen, beispielsweise verschiedene Infovis Visualisierungen.

Im letzten Abschnitt wird das Spiel initialisiert (Spielstart).

Spiel 1



Screenshot Game 1, FireFox7, WindowsXP

Die erste Spielimplementierung realisiert ein Netzwerk, dessen Knoten verschiedene Farben annehmen können. Der Spieler kann die Knoten in seiner unmittelbaren Umgebung (<= 2 Hops) durch Klicken vertauschen, um mehrere gleichfarbige Knoten zu erzielen. Basierend darauf erhält er Punkte und einen Titel (oben: „Beginner“).

Das Spiel nutzt für die Visualisierung das JavaScript Framework Infovis, das eine andere, etwas komplexere Datenstruktur als das Framework verwendet. Daher müssen sämtliche Daten erst ins Infovis Format, und später zum Abspeichern wieder zurück transformiert werden. Diese Transformation geschieht transparent, sodass der Infovis Code selbst nicht verändert werden muss und direkt, „out of the box“ lauffähig ist.

Framework Datenstruktur:

```
{
      "id": "42",
      "name": "max_mustermann",
      "content": "#00aaff",
      "peers": "3",
      "peercount": "41,100,21",
      "children": […]
}
```

Infovis Datenstruktur:

```
{
      "id": "42",
      "name": "max_mustermann",
      "data": { "$color": "#00aaff" }, #Infovis spezifisch
      "peers": "3",
      "peercount": "41,100,21",
      "children": […]
}
```

Durch den relativ kleinen Unterschied kann die Transformation zur Infovis Datenstruktur durch eine Serialisierung, Stringsuche (hke.helper.jitEnc(newData)) und Rückführung ins JSON Format geschehen:

```
   //creates the specific data structure necessary for infovis
   hke.helper.jitEnc = function(newData){
       return newData.replace(
           /content":"#([a-fA-F0-9]{6}|[a-fA-F0-9]{3})"/g, 'data":{"\$color":"#$1"}'
       );
   }
```

Transformation der Daten in ein von Infovis lesbares Format

Vor dem Speichern in der Cloud müssen die Daten wieder zurück transformiert werden, dies geschieht transparent in der Funktion zum Umwandeln der Daten ins JSON Format.

```
  //creates json string from the given array of nodes
  //returns json string or false on error
  hke.nodes.node2json = function(narray){
     if (!hke.helper.isArray(narray)) return false;
     var jnwrap="";
     for (var i=0;i<narray.length;i++){
        if (narray[i].content == undefined || narray[i].content == ""){
            narray[i].content = narray[i].data['$color'];
        }
        jnwrap+=  '{"id" : "'+narray[i].id+'", "name" : "'+narray[i].name+
                  '", "content" : "'+narray[i].content+
                  '", "peers" : "'+narray[i].peers+
                  '", "peercount" : "'+narray[i].peercount+'"},';
     }
  jnwrap=jnwrap.substring(0,jnwrap.length-1);
  return jnwrap;
  }
```

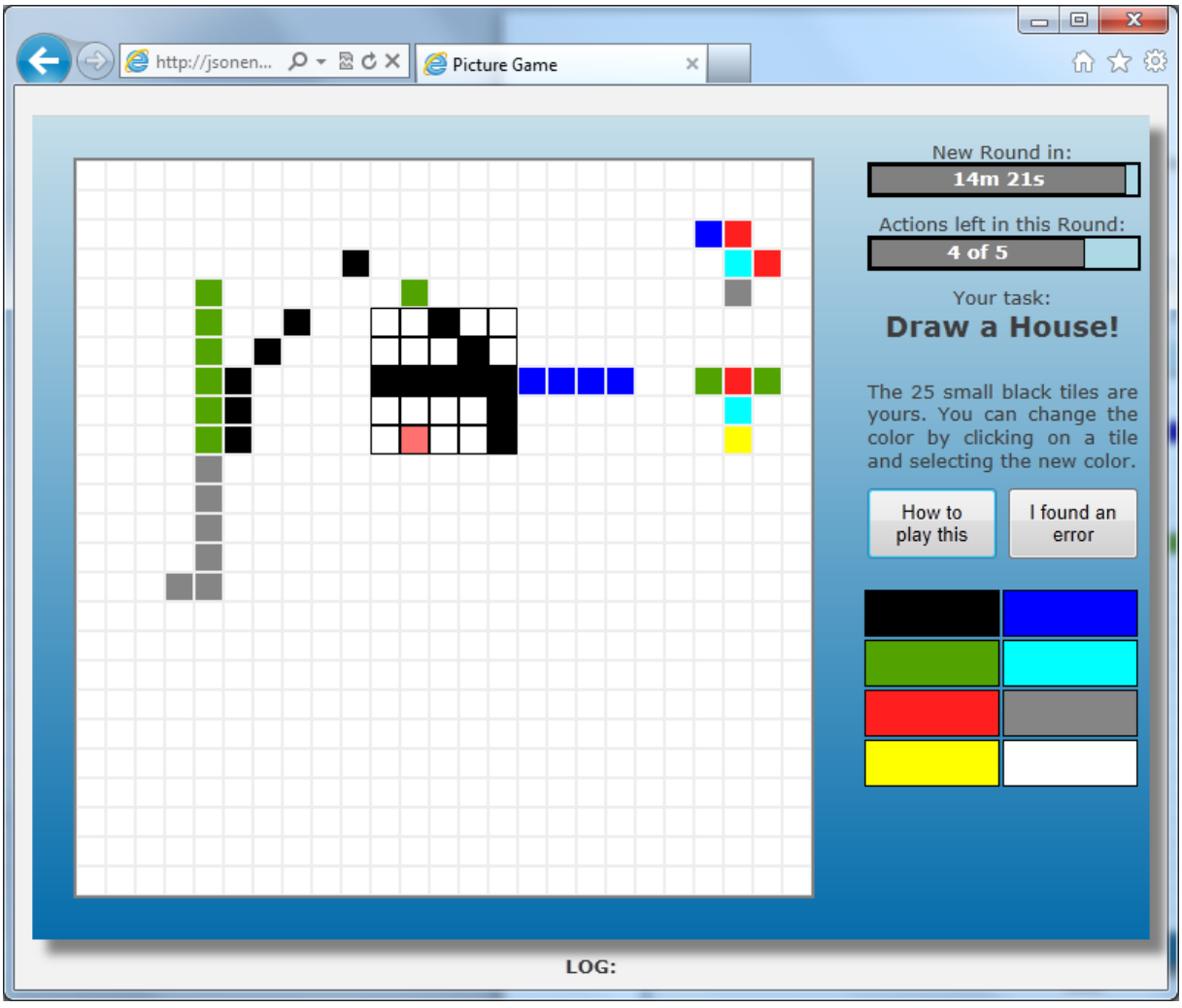
Umwandlung von Infovis Knoten ins generische JSON Format

Das Spiel benutzt mehrere externe JavaScript Bibliotheken, die beim Start geladen werden.

```
    var scripts = new Array(            //load the needed files
        "game1/jquery.js",              //load jquery to beautify some animations
        "game1/jit.js",                 //load infovis main file
        "game1/sunburst.js",            //load custom infos sunburst visualisation
        "game1/game.css"                //laod css file for the game
    );
    for (var sc in scripts) hke.helper.loadScript(scripts[sc]);
```

Zuerst wird jQuery, dann Infovis, danach eine spezielle Infovis Visualisierung (Sunburst) und zuletzt die CSS geladen.

Spiel 2: collaborative picture drawing game



Screenshot Game 2, Internet Explorer 9, Windows7 x64

Ziel des Spiels ist es, gemeinsam mit einer Gruppe von Mitspielern ein vorgegebenes Bild zu malen. Jeder Spieler ist in der Interaktion auf einen eigenen, ihm zugewiesenen Bereich beschränkt. Er hat dabei aber Sicht auf die Mitspieler in seiner Umgebung, wird also von deren Handlungen beeinflusst und muss seine eigene Vorgehensweise entsprechend anpassen.

Das Spielfeld (500 x 500 Pixel) besteht aus einem Raster von großen Kacheln, die ihrerseits in kleine Kacheln unterteilt sind. Die Anzahl der Kacheln ist konfigurierbar, ihre Größe (Höhe und Breite) wird dynamisch berechnet (und zusätzlich auf jeder Seite ein Pixel Rand abgezogen).

Das generierte Stylesheet wird beim Einfügen in die Seite vom Browser geparst und sofort angewendet. „tiles“ bezeichnet die Anzahl der Kacheln pro Reihe.

```
var html = "<style>"; //set custom css for dynamic number of tiles
    html += '.large{width:'+(500/tiles)+'px;height:'+(500/tiles)+'px;}';
    html += '.tile{width:'+((500/tiles)/tiles-2)+'px;height:'+
            ((500/tiles)/tiles-2)+'px}';
    html += "</style>";
```

Generierung eines Stylesheets für dynamische Kachelanzahl und –größen

```
//colors to draw with
hke.helper.colors = new Array( "#000000", "#0000FF", "#52A300", "#00FFFF",
                               "#FF1F1F", "#858585", "#FFFF00", "#FFFFFF" );
```

Zur Verfügung stehende Farben zum Einfärben der Kacheln

```
//changes the color of a tile and stores the change in the cloud
hke.helper.color = function(divId,nr){
    document.getElementById("tile-"+divId).style.background=hke.helper.colors[nr];
    var t=divId.split("-");
    var narray = new Array( hke.nodes.loadId(t[0]) );
    var content = narray[0].content.split(",");
    content[t[1]] = hke.helper.colors[nr];
    narray[0].content = content.join(",");
    hke.connection.sendMessage(hke.nodes.node2json(narray));
    $("#colors").animate( {height: 1}, 1000, "swing" );
}
```

Einfärben einer Kachel und Speicherung in der Cloud

Die Funktion färbt die Kachel mit ID „divId“ mit Farbe Nummer „nr“ (x-te Farbe im Colors Array). Nach dem Einfärben auf der Benutzeroberfläche (HTML Background) muss der entsprechende Knoten per loadId() geladen und eine seiner neun enthaltenen Kacheln überschrieben werden. Danach wird der Knoten wieder zusammengesetzt, ins JSON Format konvertiert und in der Cloud abgelegt.

Im letzten Schritt entfernt eine jQuery Animation das Auswahl-Panel für die Farben vom GUI.

Das implementierte Collaborative Picture Game kommt mit ca. 180 Zeilen nicht komprimiertem Programmcode (+CSS, +Grafiken) aus.

4.6 Sicherheitsaspekte

Dieser Abschnitt behandelt die Robustheit der mithilfe des Frameworks implementierten Spiele gegenüber Betrugsversuchen. Informationen bezüglich Cloud Sicherheit befinden sich in Kapitel 3.1 Cloud.

Das Framework verfolgt einen optimistischen Sicherheitsansatz. Betrug oder eigennützige Manipulation sind grundsätzlich möglich (erfordert jedoch Kenntnis von Webprogrammierung und entsprechende Werkzeuge).

Da die Implementierung von Sicherheitsabfragen einige Nachteile mit sich bringt (siehe Cloud), wird darauf verzichtet.

Ist die Qualität der gewonnen Daten durch bösartige Clients (bzw. Spieler) gefährdet, so lassen sich diese invaliden Daten relativ einfach beim Auswerten markieren und filtern. Ein die Daten auswertendes Programm kann anhand der Zeitstempel der Aktionen falsche Requests eindeutig identifizieren.

Client

Durch die Verwendung einer (im Klartext lesbaren) Scriptsprache ist auf Clientseite kein robuster Mechanismus zur Kontrolle und Validierung der Daten oder des Codes möglich. Das Komprimieren bzw. Verschlüsseln des Clients mit Laufzeitpackern kann bestenfalls einen oberflächlichen Schutz bieten, aber keine echte Sicherheit.

Moderne Debugger und Analysewerkzeuge wie Firebug[62] oder LiveHTTPHeaders[63] erlauben das Modifizieren von JavaScript Daten. Da die Kontrollmechanismen clientseitig implementiert sind (um die Cloud zu entlasten), lassen sich z.B. Runden vorzeitig neustarten oder beliebig viele Aktionen hintereinander ausführen (Replay Angriff).

Es ist herauszustellen, dass die Verwendung einer einsehbaren und unkompilierten Programmiersprache die Hürde zu einer erfolgreichen Kompromittierung nur tiefer setzt als ein Client in einer anderen Sprache (beispielsweise Java oder C++). Vom theoretischen Standpunkt der Sicherheit her spielt es keine Rolle, ob der Angreifer eine einfache Textdatei oder, aufwendiger, Bits im Speicherbereich einer kompilierten Anwendung zu seinen Gunsten modifiziert.

Cloud

Starke Validierungstechnik kann nur von der Datacloud implementiert werden, da diese nicht, wie der Client, von einem Angreifer kompromittiert werden kann.

Eine Kontrollinstanz müsste, optimaler Weise vor die eigentliche Cloud geschaltet, sämtliche Anfragen auf Validität prüfen. Dazu wäre jeweils eine Überprüfung der letzten Aktionen des Nutzers und damit mehrere Datenbankabfragen nötig (auf einer eigens dafür geschaffenen Datenbank).

Diese Modifikationen stehen jedoch im Gegensatz zu dem Konzept, eine möglichst einfache und gut skalierbare Datenschnittstelle bereitzustellen.

---

[62] Debug Tool Firebug, https://addons.mozilla.org/de/firefox/addon/firebug/

[63] Debug Tool LiveHTTPHeaders, https://addons.mozilla.org/de/firefox/addon/live-http-headers/

5. Evaluation & Auswertung

Die Auswertung der vom Framework erzeugten Daten kann über mehrere Interfaces erfolgen. Einerseits das Webinterface der Google App Engine, welches der Verwaltung und Evaluation der Cloud Anwendung dient und andererseits die bereits vorgestellte Download-Schnittstelle der Cloud.

5.1 Interfaces

Webinterface GAE

Das Dashboard der GAE Applikationen erlaubt die Einsicht und Protokollierung von allen relevanten Leistungsgrößen wie

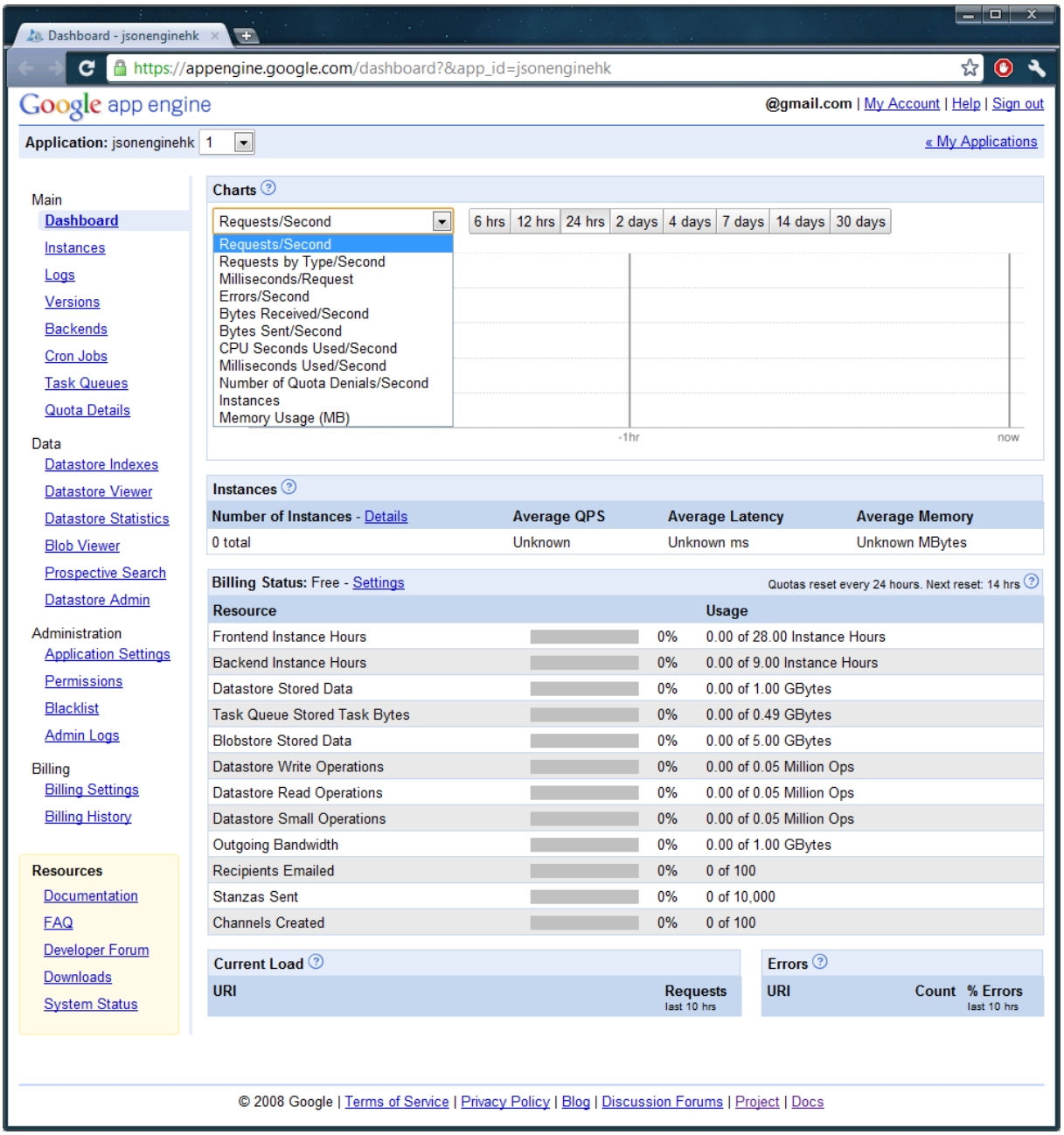
- CPU Rechenzeit,
- eingehende und ausgehende Datenbandbreite,
- gespeicherte Daten,
- Anzahl der Instanziierungen

aufgeschlüsselt über

- ihre Verteilung auf die verschieden Teile der Applikation,
- in Echtzeit und auf die Gesamtdauer.

Da Cloud-Anwendungen im Gegensatz zu herkömmlichen Serverapplikationen nicht nach statischen Gesichtspunkten (Webspace und Traffic Pakete) abgerechnet werden, sondern nach tatsächlich genutzter Rechenleistung, ist eine saubere und transparente Protokollierung von Seiten der Anbieter unerlässlich. Diese Protokolle lassen sich exzellent zur Evaluierung und für die Leistungsdokumentation nutzen. Sämtliche Datenbanken, Cronjobs[64], Statistiken und vieles mehr lassen sich über das Webinterface komfortabel abfragen und bearbeiten.

[64] Jobsteuerung Cron, http://en.wikipedia.org/wiki/Cron



App Engine Dashboard: https://appengine.google.com/dashboard?&app_id=jsonenginehk

Das Dashboard für eine App Engine Applikation findet sich unter der Url (APPID = ID der jeweiligen Applikation):

```
https://appengine.google.com/dashboard?&app_id=APPID
```

Spezieller JavaScript Client mit erweiterten Fähigkeiten

Normale Clients (Spieler) fragen nur die für sie relevanten Daten ab und erhalten jeweils nur die aktuellsten Datensätze. Ein spezieller Evaluationsclient nutzt grundsätzlich dieselbe Technik, kann aber alle Daten anfordern. Damit erhält er nicht nur Zugriff auf kleine Ausschnitte des Netzwerks, sondern auf das gesamte Netzwerk in der Gegenwart und in vergangenen Zuständen.

Durch eine iterative Abfrage sämtlicher Daten über die Zeit lässt sich die Entstehung des Netzwerks rekonstruieren und mithilfe geeigneter Methoden ansprechend visualisieren und im Zeitraffer beobachten.

Diese Schnittstelle kann aber auch zum reinen Download und zur Speicherung der Daten im JSON-Datenformat genutzt werden, zum Beispiel zur Datensicherung oder zur Weiterverarbeitung durch dritte Programme und beliebige andere Software.

5.2 Reichweite

Grundsätzlich sollte das Framework auf allen HTML5 fähigen Geräten funktionieren. Aufgrund beschränkter Test-Hardware wurden nur die unten folgenden Hard-und Softwarekombinationen getestet.

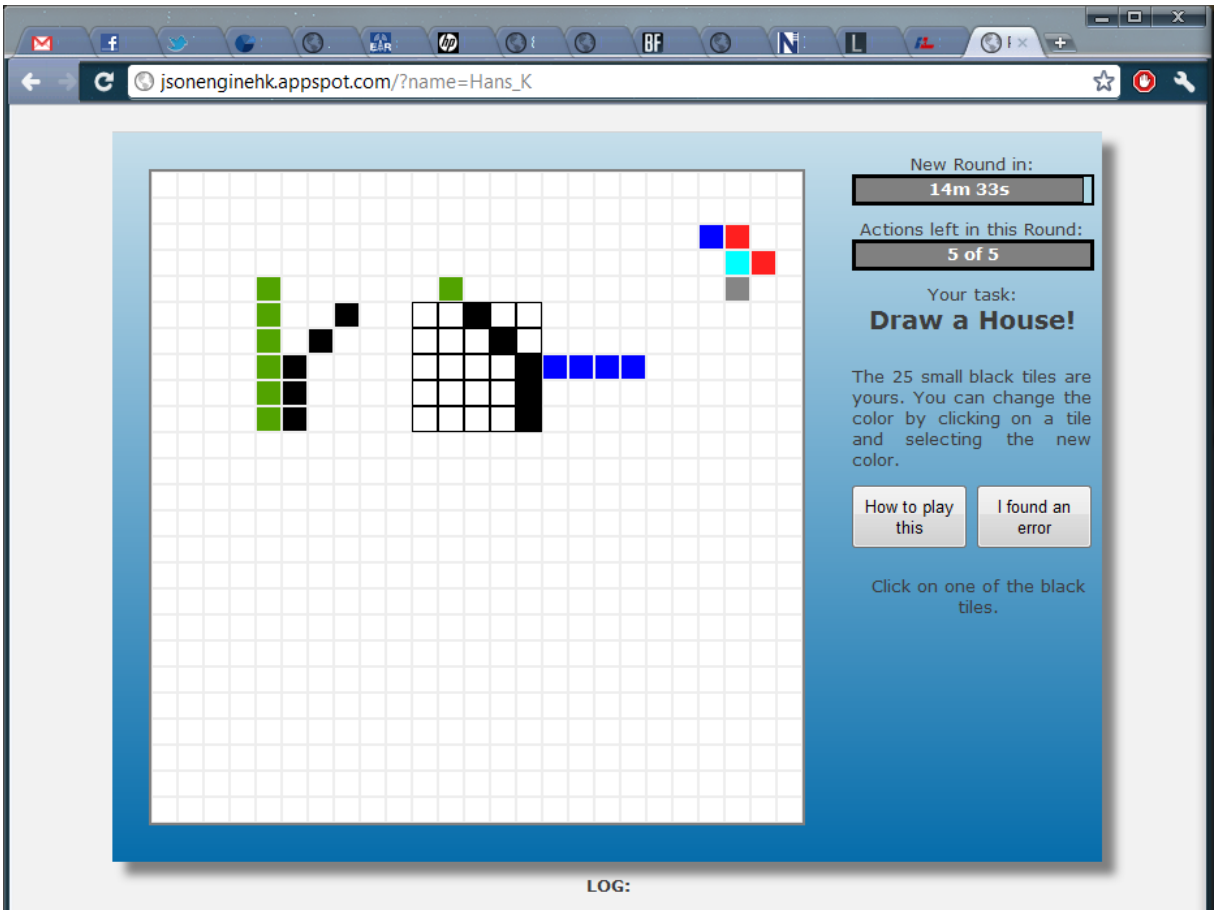
Dabei ist anzumerken, dass bis auf den Internet Explorer 8 (Windows XP) und die Spielekonsole Sony Playstation 3 sämtliche Tests erfolgreich verliefen. Die verwendete grafische Oberfläche erlaubt eine konsistente Bedienung sowohl auf kleinen (HTC Desire, 48 x 80mm, 480 x 800 Pixel) als auch großen Displays (LG Flatron E2750, 68,5cm = 27,5“, 1920 x 1080 Pixel) und verschiedene Steuerungsarten (Touchscreen, Maus).

Durch Verwendung von CSS-erzeugten Gradienten und Farben anstatt statischer Dateien skalieren die verwendeten Bilder (Hintergrund etc.) auch auf sehr hohen Auflösungen, ohne dass Artefakte (grobe Pixel) entstehen.

Desktop PC

Betriebssysteme: Windows XP
Windows 7

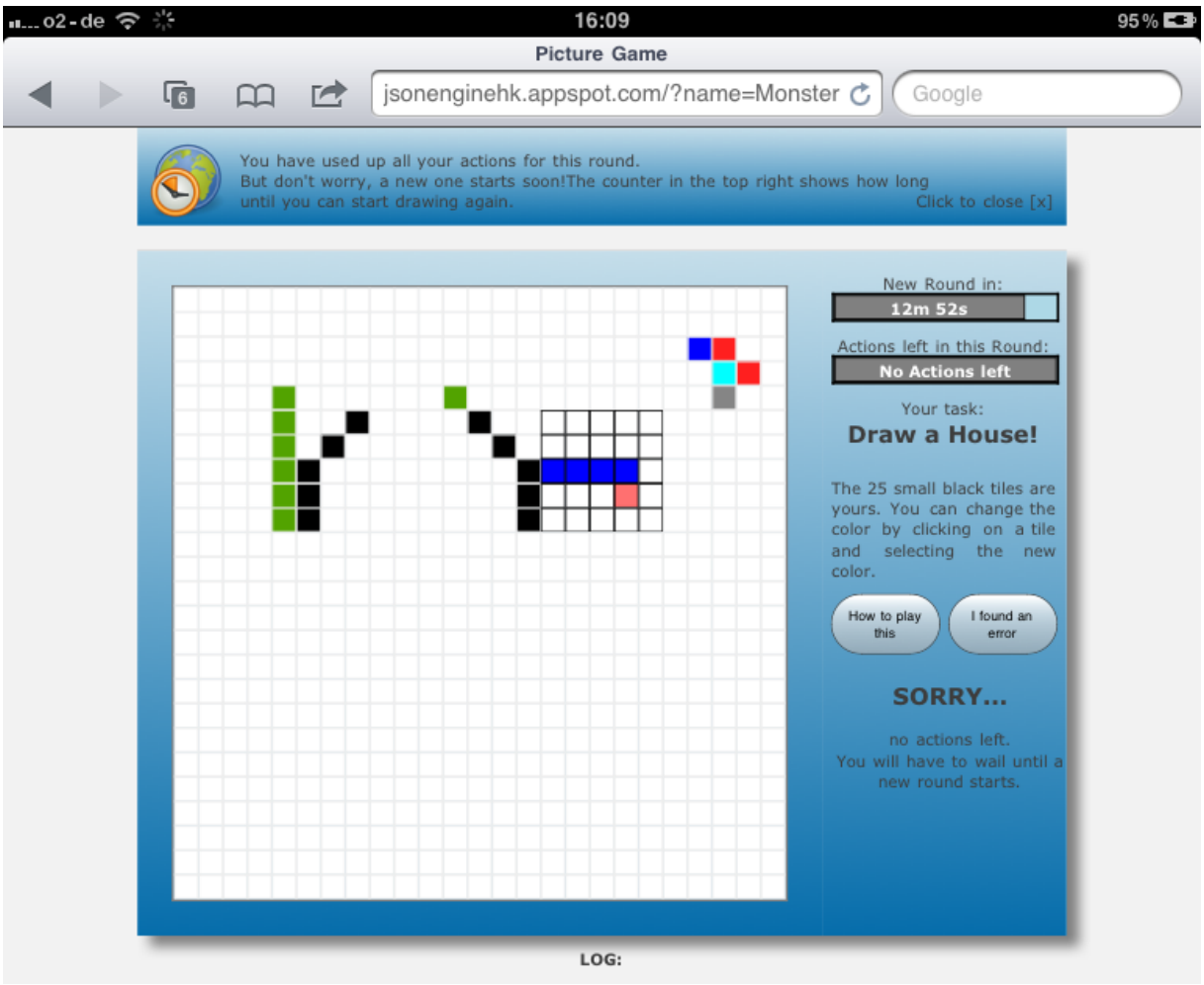
Browser: Firefox7
Google Chrome 14
Internet Explorer 9 (nur Windows 7)
Safari 5.1
Internet Explorer 8 (Windows XP) : nicht uneingeschränkt lauffähig



Game 2 im Chrome Browser auf Windows 7

Tablet

Betriebssysteme:	HP Touchpad WebOS 3.0.2
	iOS 4.3.5
Browser:	WebOS Browser
	iOS Browser



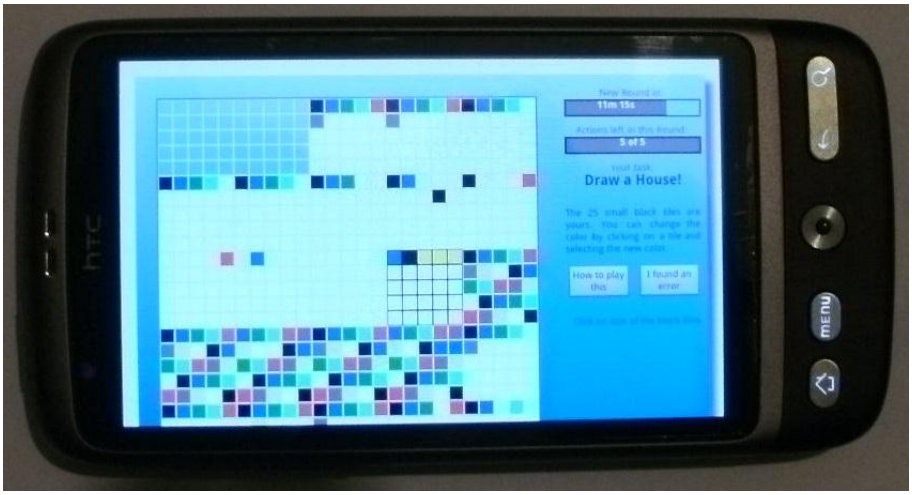
Game 2 auf dem iPad (iOS 4.3.5)

Handy

Betriebssysteme: Android 2.3.5 MIUI

Browser: MIUI Browser / Miren 1.9.23

Game 1 auf Android



Game 2 auf Android

5.3 Theoretische Leistungsfähigkeit der Architektur

Obwohl sämtliche „teuren“, also rechenaufwendigen Operationen auf den Clients ausgeführt werden, können diese keinen Leistungsengpass darstellen (die zur Verfügung stehende Hardware wächst linear mit der Spieleranzahl).

Eine kritische, wachstumsbegrenzende Stelle („Bottleneck“) muss in der zentralen Verbindung, der Cloud, liegen.

Wie bereits ausgeführt, ist eine der zentralen Eigenschaften von Cloud Plattformen ihre Skalierbarkeit für große Datenmengen und Nutzerzahlen. Es ist also nicht zu erwarten, dass beispielsweise anwachsende Datenmengen oder ähnliches einen limitierenden Faktor darstellen.

Die relevanten (also limitierenden) Platzkosten und Laufzeiten sind einerseits die reinen Speicherplatzkosten für die entstehenden Daten und andererseits ihre Propagierung zu den Clients und die dazu benötigte Verwaltung der Kommunikation.

Sie lassen sich, abhängig von der erlaubten Nachbarknotenmenge pro Nutzer, in zwei Fälle untergliedern:

Allgemeiner Fall

Im (unter praktischen Bedingungen unrealistischen) allgemeinen Fall kann sowohl der Platzbedarf als auch der Kommunikationsaufwand quadratisch werden. Um diese Schranke zu erreichen müsste jeder Nutzer Verbindungen zu allen anderen Knoten aufbauen und gleichzeitig mit sämtlichen anderen Nutzern im System angemeldet und aktiv sein.

In diesem Fall hat jeder Nutzer Sicht auf das gesamte Netzwerk, daher betrifft eine Änderung alle Knoten und muss auch zum gesamten Netz (allen Nutzern) propagiert werden.

Begrenzte Nachbarknotenmenge / Begrenzung des Ausgangsgrads

Durch eine nach oben begrenzte Anzahl von Nachbarknoten pro Spieler (beziehungsweise Limitierung der Kanten pro Knoten) wächst die Gesamtknotenmenge und damit die Gesamtdatenmenge linear.

Da jeder Nutzer in seinen Interaktionsmöglichkeiten auf die ihm adjazenten Knoten (möglicherweise über mehrere Hops) beschränkt ist, müssen Änderungen auch nur zu den Besitzern dieser Knoten propagiert werden, das heißt, dass der Kommunikationsaufwand ebenfalls linear steigt.

Für die meisten Realanwendung ist zu erwarten, dass kein oder sehr wenige Nutzer Verbindungen zum gesamten Netzwerk aufbauen, also tatsächlich eine exponentielle Last erzeugen. Diese Möglichkeit muss außerdem von der Spielimplementierung explizit erlaubt, also vom Programmierer vorgesehen worden sein. Auch eine Visualisierung der Daten wird unter diesen Umständen sehr schwer (falls beispielsweise hunderte oder tausende von Nachbarknoten angezeigt werden sollten).

Eine maximale Anzahl von ausgehenden Kanten pro Knoten stellt lediglich eine schwache Einschränkung dar, insbesondere, da vereinzelte Überschreitungen die lineare Laufzeit nicht zerstören.

5.4 Praktische Leistungsfähigkeit: Stresstest

Das Framework verfügt über rudimentäre Fähigkeiten zu Test-Zwecken künstliche Lastsimulationen zu erzeugen.

Ziel

Der Stresstest soll eine Evaluation der Skalierbarkeit des Systems und eine Analyse unter Last ermöglichen. Dazu wird eine hohe Anzahl an gleichzeitig angemeldeten Clients (Spielern) simuliert, von denen jeder über eine gewisse Zeit verteilt verschiedene Handlungen ausführt (spielt). Das Ziel ist dabei, eine möglichst große Last zu erzeugen (sei sie auch unrealistisch), um die Engpässe und Grenzen der Architektur aufzuzeigen.

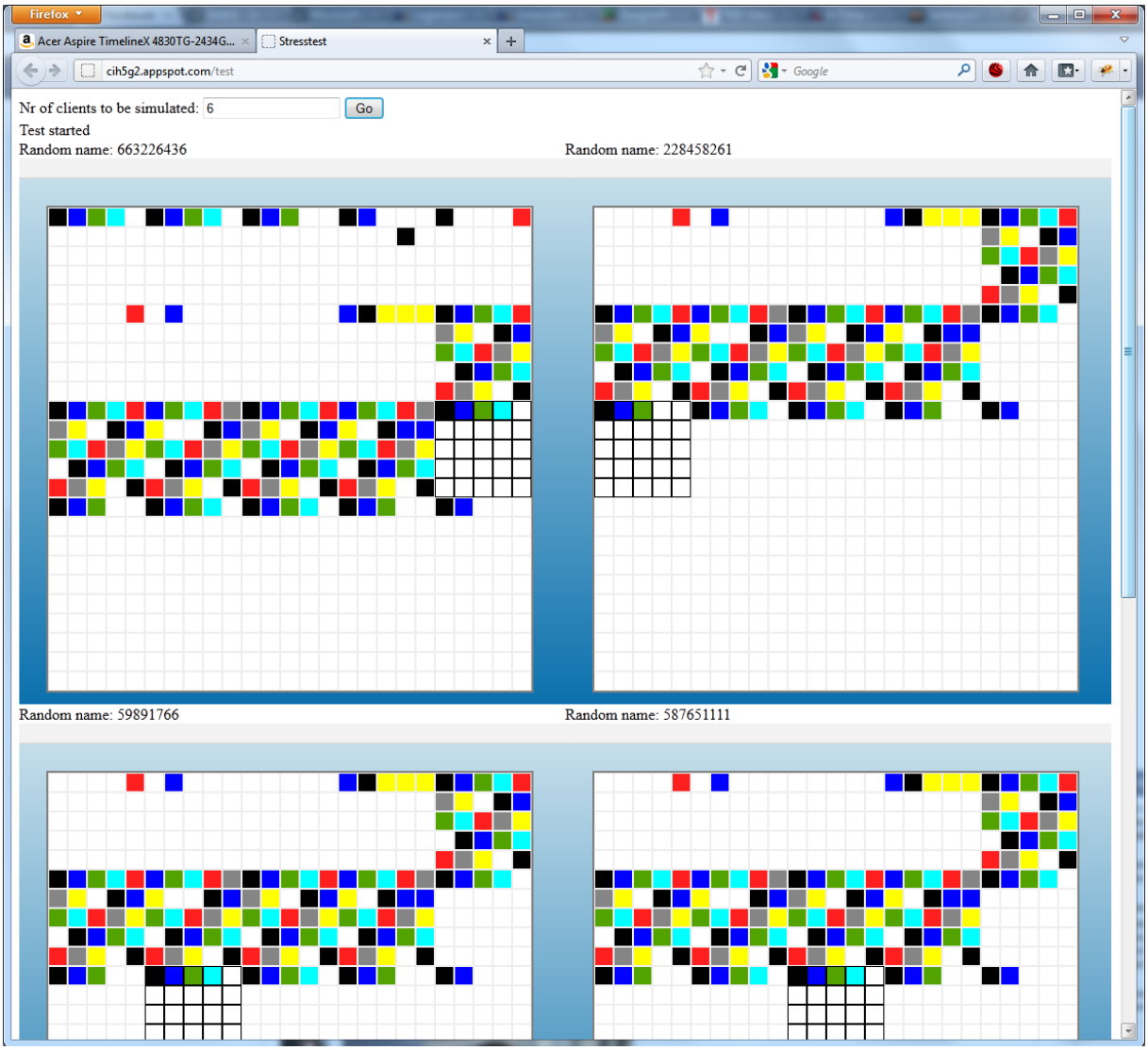
Testumgebung

Da das gesamte Programm im Browser abläuft, lassen sich mehrere Instanzen problemlos in mehreren Browsern öffnen. Diese lassen sich wiederum in einem Browser mit mehreren Tabs oder noch weiter, zu einem einzigen Testsystem mit mehreren Spielinstanzen, die jeweils in einem Iframe[65] laufen, zusammenfassen.

Um ein Spielverhalten zu simulieren, enthält die Testumgebung zusätzlichen Programmcode, der die Handlungen der Spieler nachahmt. Diese Handlungen sind zufallsbasiert, orientieren sich aber an einem sinnvollen und realistischen Spielverhalten. Um mehr Last erzeugen zu können, werden Beschränkungen wie eine Obergrenze der Handlungen pro Runde für Stresstest-Clients aufgehoben, wodurch diese theoretisch im Sekundentakt Daten erzeugen und in die Cloud einspeichern können. Durch das künstliche Anlegen der Clients zur gleichen Zeit sind diese alle benachbart, also untereinander stärker vernetzt, als dies unter „normalen“, realistischen Bedingungen der Fall wäre. Es wird also, solange die Anzahl der simulierten Clients unter der Grenze der sichtbaren Nachbarknoten liegt, eine exponentiell steigende Kommunikations- und Speicherlast erzeugt.

Die so aufgebaute Testumgebung erlaubt die Simulation von mehreren unabhängigen Clients auf einem einzelnen Computer beziehungsweise von einer großen Spieleranzahl auf einer moderaten Anzahl von Desktop PCs.

[65] Inline Frame, HTML Tag, http://en.wikipedia.org/wiki/HTML_element#Frames



Mehrere (hier: sechs) automatisierte Clients in einem Browserfenster

Ergebnisse

Der Stresstest offenbart tatsächlich einen Engpass in der Architektur, allerdings an einer gänzlich unerwarteten Stelle. Während bei theoretischer Betrachtung eher der Kommunikationsaufwand bedenklich erscheint, tritt bedingt durch die künstliche Last des Tests eine große Anzahl an Leseoperationen in der Datenbankanbindung auf. Diese Operationen wachsen so schnell, dass innerhalb kürzester Zeit die kostenfreien Quotas der GAE erschöpft sind und einen Abbruch der Simulation erfordern. Der Effekt tritt bereits bei weniger als 10 Clients und ~10 Anfragen pro Minute auf. Über das GAE Dashboard lässt sich dieser Prozess gut sichtbar mitverfolgen und auch das auftretende „short-term quota limit“ einsehen.

Dieser Effekt lässt sich durch die folgende, annähernde Rechnung erklären:

10 Clients * 10 neue Knoten pro Minute = 100 Knoten / Min

Durch die bereits beschriebene künstlich erzeugte Umgebung sind alle Clients maximal vernetzt, also pro neuen Knoten zehn Channel Verbindungen. Pro 1 Channel entstehen dabei:

1 Verbindungsabfrage + 10 Knotenabfragen = 11 Abfragen / Chan

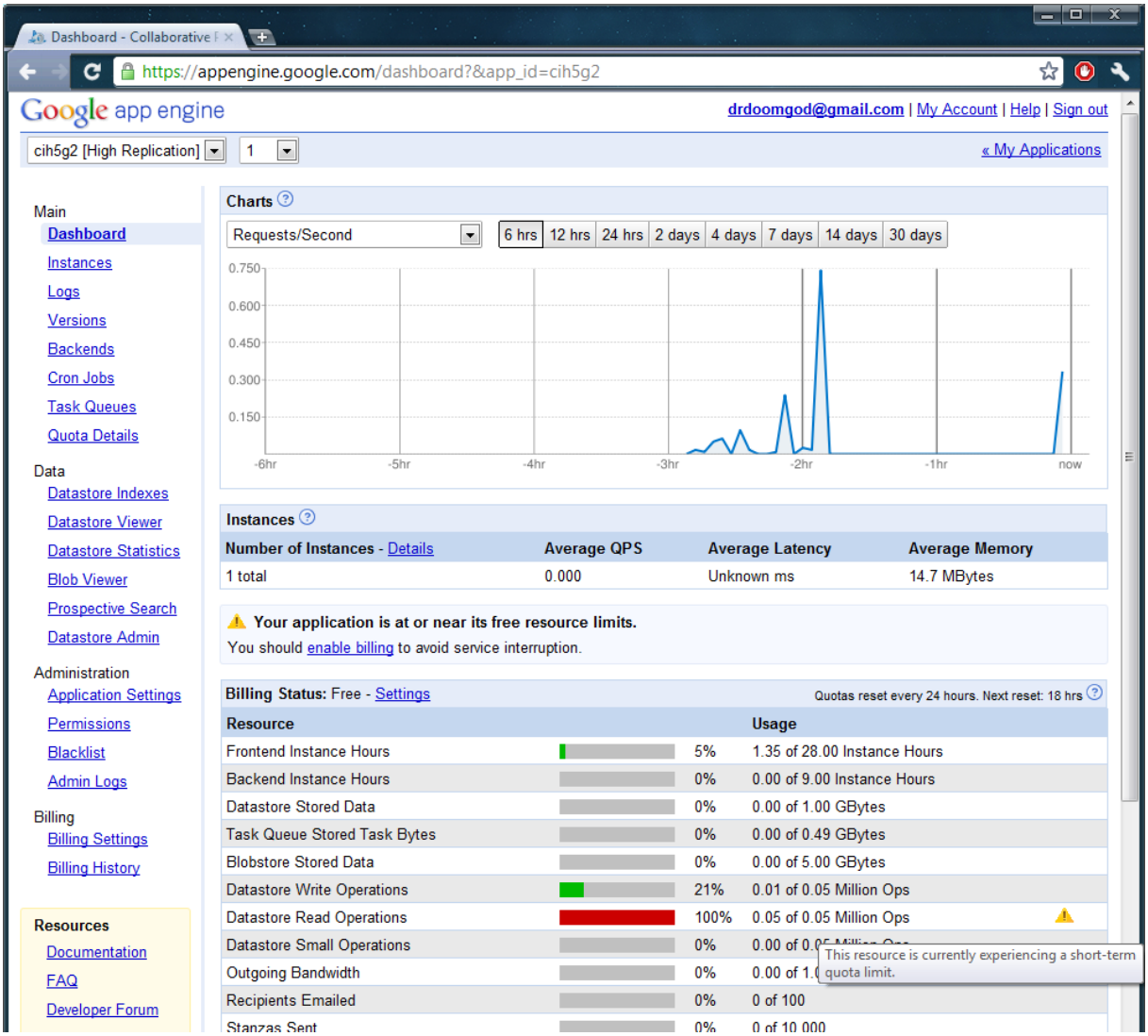
Es entstehen also ca.:

100 Knoten / Min * 110 Abfragen = 11000 Abfragen / Min

Wobei ein Knoten aus sechs und ein Channel aus drei Datensätzen besteht.

Obwohl die zugrundeliegenden Annahmen nicht einer realen Anwendungsanforderung entsprechen, sondern eher ein Worst-Case-Szenario darstellen, ist der Umfang der entstehenden Last bemerkenswert. Der Hauptgrund dafür ist das atomare Laden der Knotendatensätze. Hier ist Raum für Optimierungen, beispielsweise durch Zusammenfassen der Abfragen, insbesondere, weil die hierzu nötigen Informationen bereits im System vorliegen.

Konkret ließen sich alle Nachbarknoten gemeinsam über eine UND-Verknüpfung der IDs in einer Operation laden (Liste der Nachbarn ist nach erstmaligem Laden in „connection.allpeers“ abgelegt). Auch Caching-Mechanismen bieten ein weiteres Einsparungspotential um Leseabfragen zu vermeiden.



„This resource is currently experiencing a short-term quota limit.“

6. Fazit

Die hier in Form eines Frameworks realisierte Infrastruktur erlaubt die Implementierung einer verteilten Anwendung, die einen Teil der Vorzüge neuer Technologien wie Cloud Computing und HTML5 ausnutzt.

Die erzielte Reichweite ist durch die konsequente Verwendung von Standards um ein Vielfaches höher als bei konservativen Desktopapplikationen und umfasst mit Handys, Tablets, Desktop Computern und Fernsehern nahezu jedes webfähige Gerät mit einem modernen Browser. Ein persistenter Speicher oder proprietäre Software-Plugins werden im Client nicht benötigt.

Die gewählte Architektur macht einen dezidierten Server für die Anwendungslogik obsolet und verschiebt die Rechenlast zum Client hin. Durch die Speicherung der Daten in der Cloud kann die einzige zentrale und limitierende Komponente virtualisiert und mit hohem Skalierungspotential ausgestattet werden.

Die Evaluierung kann sowohl mit der hier verwendeten Technik HTML5 als auch mit beliebigen anderen Plattformen und Werkzeugen, entkoppelt von der eigentlichen Infrastruktur, vorgenommen werden. Mit dem App Engine Dashboard besteht außerdem die Möglichkeit der Einsicht in die Leistungsdaten im laufenden Betrieb.

Noch bis vor wenigen Jahren wäre eine annähernd leistungsfähige Architektur nur mit überproportional größerem Aufwand an Ressourcen (Hardware, Komplexität der Software) denkbar gewesen. Gerade die Kombination der neuen Technologien wie Cloud Computing und ihr ergänzendes Zusammenwirken ermöglicht es, hochskalierbare Software zu realisieren, die einigen Beschränkungen konventionell designter Webapplikationen nicht mehr unterworfen ist. Das hier entworfene System zielt auf eine möglichst hohe Anzahl paralleler Benutzer ab, die über die Zeit eine große Datenmenge generieren und stellt daher genau jene Anforderungen, die auch an eine moderne Cloud Applikation gestellt werden. Diese neuen Möglichkeiten sind weniger der Verdienst einzelner, hier genutzter Software wie der GAE oder einzelner Features von HTML5, die bis auf wenige inessentielle Merkmale austauschbar sind oder hier von vornherein ungenutzt blieben. Vielmehr ist die allgemeine Entwicklung der Browser von einem reinen Markup Renderprogramm hin zu einer mächtigen, standardisierten Plattform mit Hardwarezugriff die treibende Kraft.

Obwohl die Kommunikation für eine Webanwendung nahezu in Echtzeit abläuft, ist die Architektur für einige Anwendungsfälle noch nicht geeignet. In einem 3D-Action Multiplayer Spiel kann bereits eine Verzögerung im Bereich von wenigen hundert Millisekunden als störend empfunden werden. Derartige Antwortzeiten sind mit Bordmitteln in aktuellen Browsern nicht erreichbar. Auch die limitierten grafischen Fähigkeiten von Browsern stellen noch eine starke Einschränkung dar, so ist beispielsweise noch keine konsistente und performante Darstellung von 3D Grafikengines möglich (mangels vollständiger Implementierungen von Standards wie WebGL). Hier sind Plugin basierte Systeme wie Adobe Flash oder Microsoft Silverlight der nativen Implementierung von HTML5 noch überlegen.

Trotz der vielfältigen Vorteile und positiven Eigenschaften kratzen die heutigen Webapplikationen erst an der Oberfläche des künftig Machbaren. An diesen Beschränkungen wird stark gearbeitet und selbst die hier genutzten Techniken wie die Google App Engine oder die verschiedenen HTML5-Browser sind noch sehr jung (und werden hochfrequent aktualisiert). Es erscheint also nur als Frage

der Zeit „wann“ und nicht „ob“ entsprechende Technologien entwickelt werden, um diese Beschränkungen aufzuheben.

Durch die während der Implementierung und des Stresstests gewonnenen Erfahrungen lässt sich feststellen, dass die Infrastruktur noch weiter optimierbar ist. Dies ist vor allem auf den Übertragungsprozess von theoretischem Konzept zu tatsächlicher Implementierung zurückzuführen. Während die Gesamtkonzeption der Architektur robust ist, wirkt sich auch die Wahl von Detailentscheidungen während der Programmierung in nicht unerheblichem Maße auf die Performance aus. Diese Erfahrungswerte sind schwer von vornherein zu antizipieren, stellen aber auch keine unüberwindbaren Probleme dar, da sie an der grundsätzlichen Architektur nichts verändern, sondern nur lokal begrenzter Änderungen bedürfen.

7. Appendix (English)

7.1 A distributed, cloud based infrastructure with HTML5

The framework implements a technical core functionality to build a distributed, cloud based infrastructure upon (e.g. a social game).

It allows the user to access and interact with a global network (graph), which is stored in the cloud, and controls data consistency and propagation in real time. The actual game implementation defines the graphical user interface (the user's view of the network) and the target function. The framework offers functions to modify the network in an easy way. All functions can also be overwritten to enrich basic functionality, as has been shown in the actual game examples.

All actions and changes are logged and can be reproduced and analysed.

The framework uses standards like HTML5 and JSON to create portable code and data and integrates well with other JavaScript frameworks (the games use jQuery and Infovis for gui animations).

It runs on a variety of platforms (like desktop pcs, tablets or mobile phones) and operating systems (like Windows, Linux, Mac OS, WebOs, Android or iOS).

All source codes are commented in English.

7.2 Technical Documentary

This is just a brief listing of functions and variables, for more precise information please refer to 4 Design und Technik (German).

Cloud

```
maxLayers = 1     #integer, depth (# of layers) of children peers
startData = 0     #boolean, let cloud generate startdata (instead of clients

class Data(db.Model):
  #store the game data
  id = db.IntegerProperty(required=True)      #integer, id, ascending
  name = db.StringProperty()                  #string, name (email)
  content = db.StringProperty()               #string, data
  peers = db.StringProperty(default="")       #string, list of connected ids
  peercount = db.IntegerProperty(default=0)   #integer, needed to find peers
  date = db.DateTimeProperty(auto_now_add=True)    #date, date + time

class Connections(db.Model):
  #store the connected clients
  name = db.StringProperty(required=True)     #string, name
  chan = db.StringProperty(required=True)     #string, gae channel
  allpeers = db.StringProperty(default="")    #string, list of peers this
                                                node sees

class default(webapp.RequestHandler):
  #default page: renders login or game page

class gaeChannel(webapp.RequestHandler):
  #Google App Engine Channel backend

class evaluation(webapp.RequestHandler):
  #evaluation client

class test(webapp.RequestHandler):
  #stresstest client

class download(webapp.RequestHandler):
  #download data (used by the evaluation client)

class delete(webapp.RequestHandler):
  #delete all data in db, reset app

def nodes(name):
  #load own and connected nodes for "name"
  #returns nodes as JSON or maxId if user is new

def start_algo():
  #create some random nodes and connect them

def start_static():
  #create some hardcoded start nodes

def randomColor():
  #return a random color

def connectNodes(id, id2):
  #connect the two nodes => add to peers, raise peercount
```

Client

The frameworks main object „hke“ consists of the following components:

settings:	username etc
data:	node data the user can see and interact with
nodes:	operations on nodes (modify the data)
gui:	grafical user interface
connection:	connection to cloud
helper:	helper functions

Client JavaScript Object:

```
hke:
    //root object of the framework that holds everything, defines the
    //namespace for all functions and objects. This is the only global var
    settings:
        //settings: username, debugmode etc
        debug : false
            //boolean, debug mode set to TRUE to enable debug messages
        actions : 10
            //integer, user actions per round are limited. this number
defines how many actions per round are allowed
        minutes : 1
            //integer, how long a round is in minutes
        maxPeers : 5
            //integer, maximum number of connected peers to one node (this
is definded in cloud)
        maxLayers : 3
            //integer, maximum depth (# of layers) of nested children peers
        name : ""
            //string, users name
        channelId : ""
            //string, channel ID to identify the client

    data : ""
        //object, node data that is automatically updated from the cloud

    nodes:
        //operations on nodes: all write operations return an array of new
        //nodes (narray) which can then be JSON encoded and send to cloud
        //through hke.connection.sendMessage(hke.nodes.node2json(narray))
        switchNodes : (id1, id2){function}
            //switch 2 nodes
            //returns an array of nodes
        connectNodes : (id1, id2){function}
            //connect 2 nodes: add itself to the others peers (only if
            //MAXPEERS allows it). returns an array of nodes or false on
            //error or nodes allready connected
        disconnectNodes : (id1, id2){function}
            //disconnect 2 nodes
            //returns an array of nodes or false if nodes not connected
        addPeer : (nodeId, id){function}
            //adds an id to the peerlist of nodeId
            //returns an array of nodes or false if nodes connected or
            //peercount >= maxpeers
        areConnected : (id1, id2){function}
            //checks if the 2 nodes are connected directly
            //returns true or false
        removeNode : (id){function}
            //delete a node: disconnect the node from all its peers
```

```
        //returns an array of nodes
    exchange : (id){function}
        //helper for switchNodes: first call stores id1 second call
        //switches id1 and id2
    copyNode : (obj){function}
        //returns node object obj without the nested children
    content : (id, content){function}
        //sets content of node id
        //returns an array of nodes
    create : (id, content, peers){function}
        //create new node with id, content, peers [peers = comma
        //seperated list of peers, eg 1,2,4]. returns an array of nodes
    createNodeArray : (tmpData){function}
        //create an assoc Array of all nodes labeled by id
        //returns array
    loadId : (id){function}
        //return the node with id = ID from hke.data
    newUser : (maxId){function}
        //is called when a new User has no node data present in the
        //cloud. returns an array of newly created nodes
        //function should be overwritten by game
    node2json : (narray){function}
        //creates json string from the given array of nodes
        //returns json string or false on error

gui:
    //grafical user interface
    updateInterval : 1000
        //integer, update timer every x milliseconds
    init : (){function}
        //set timer and its gui updates
    draw : (){function}
        //redraws the gui (game and control)
    info : (msg){function}
        //display message at top
    title : (str){function}
        //set HTML title attribute (browser window name) to str
    action : (p){function}
        //check if user has enough actions to perform action. resets
        //actions on a new round. returns TRUE id user is allowed to do
        //action false if user has not enough points
   game : (){function},
        //draws the game
        //function should be overwritten by game
    control : (){function},
        //draws the control
        //function should be overwritten by game

connection:
    //Connection to Cloud
    channel : ""
        //string, channel for connection to cloud
    socket : ""
        //string, socket variable for channel
    init : (){function}
        //create Channel
        //function is automatically called when the framework is
        //loaded, so communication is ready when game is initializing
    sendMessage: (data){function}
        //send data to server
    receiveMessage = (newData){function}
        //is called when new data is reveived from the cloud
```

```
        //default updates hke.data,
        //function should be overwritten by game
    close : (cid){function}
        //close channel cid on cloud
    ajax : (url){function}
        //send an ajax request to the cloud
        //function should be overwritten by game

helper:
    assocSort : (arr) {function}
        //sort assoc array
        //returns sorted array
    copy : (obj) {function}
        //return a copy of an object or array
        //js copies simple variables by value and arrays by reference
    loadScript : (src){function}
        //load and execute an external js file, css file or js code
        //src can be path to js / css file or plain js code
    loadJit : (src){function}
        //load a jit js file and reset game div
    stringify : (obj){function}
        //create string from object (for debug etc)
        //returns string
    isArray : (obj) {function}
        //test if OBJ is an array
    log : (str){function}
        //log data for debug purposes
    ready : (callbackFunction){function}
        //execute a function when DOM is loaded, alternative method to
        //body:onload
    setCookie : (key, val, min){function}
        //set cookie with name KEY and value VAL valid for MIN minutes
    getCookie : (key){function}
        //get cookie with name KEY return null if not existent
    init : (){function}
        //initialize some stuff (gui)
        //function should be overwritten by game
    stresstest : (){function}
        //this function is run if the calling url is "stresstest.html"
        //function should be overwritten by game
```

8. Quellen

In lexikographischer Ordnung

A

Android OS, http://www.android.com/

Apple iOS, http://www.apple.com/ios/

Apple Safari, http://www.apple.com/safari/

AS3, Amazon Simple Storage Service, http://aws.amazon.com/s3/

B

BoxCryptor, http://www.boxcryptor.com/

C

Cloud Computing, http://de.wikipedia.org/wiki/Cloud_Computing

Cloud Computing Schaubild,
http://de.wikipedia.org/w/index.php?title=Datei:Cloud_computing.svg

Comet, http://en.wikipedia.org/wiki/Comet_(programming)

Cron, Jobsteuerung, http://en.wikipedia.org/wiki/Cron

CSS3, Cascading Style Sheets, http://www.w3.org/Style/CSS/

D

„Div"-Tag, HTML, http://www.w3schools.com/tags/tag_div.asp

E

ECMA Script, ECMA-262 Spezifikation, http://www.ecmascript.org/

Excanvas, http://excanvas.sourceforge.net/

F

Firebug, https://addons.mozilla.org/de/firefox/addon/firebug/

„Fully Homomorphic SIMD Operation", Nigel P. Smart und Frederik Vercauteren, erscheint 2011,
http://homes.esat.kuleuven.be/~fvercaut/papers/DCC2011.pdf
http://bit.ly/uT5cLz

G

GAE Channel API, http://code.google.com/intl/de/appengine/docs/python/channel/
http://bit.ly/sfjOoJ

GAE Google App Engine, http://code.google.com/appengine/,
http://code.google.com/intl/de/appengine/

GAE SDK Python,
http://code.google.com/appengine/downloads.html#Google_App_Engine_SDK_for_Python
http://bit.ly/Ng50E

GAE Streaming API, http://arstechnica.com/web/news/2010/12/app-engine-gets-streaming-api-and-longer-background-tasks.ars
http://bit.ly/h0bIYQ

GAE Quotas, http://en.wikipedia.org/wiki/Google_App_Engine

Go, http://golang.org/

Google Chrome Experiments, http://www.chromeexperiments.com/

Google Chrome, http://www.google.com/chrome

H

HTML5, http://www.w3.org/html/logo/

Hunch, http://www.hunch.com

I

IE7.js, http://code.google.com/p/ie7-js/

Information Management: A Proposal.", Tim Berners-Lee, CERN (March 1989, May 1990),
http://www.w3.org/History/1989/proposal.html

Infovis, http://thejit.org

J

Java, http://www.oracle.com/technetwork/java/index.html

JavaScript / ECMAScript, http://www.ecma-international.org/publications/standards/Ecma-262.htm, http://bit.ly/FpAue
http://en.wikipedia.org/wiki/JavaScript

JavaScript Missbrauch, http://de.wikipedia.org/wiki/JavaScript#Missbrauch

JavaScript Obfuscator, http://www.javascriptobfuscator.com/

JavaScript Packer, http://dean.edwards.name/weblog/2007/04/packer3/
http://bit.ly/493Cja

jQuery Mobile, http://jquerymobile.com/

jQuery Swing Animation, http://api.jquery.com/animate/

JQuery, http://www.jquery.com

JsCompress, http://jscompress.com/

JSON, JavaScript Object Notation, http://www.json.org/

K

Kai Riemer, Fabian Brüggemann, 2009: Personalisierung der Internetsuche – Lösungstechniken und Marktüberblick, http://eprints.rclis.org/bitstream/10760/12738/1/Personalisierung.pdf
http://bit.ly/tVCw6N

L

Lamport, 2001, Paxos made simple, SIGACT News 32(4):18-25,
http://research.microsoft.com/users/lamport/pubs/paxos-simple.pdf
http://bit.ly/EZOqD

LastFM, http://www.last.fm/

„Letzte Chance für die VZ-Netzwerke", http://www.golem.de/1109/86675.html

Linux, https://www.linux.com/

LiveHTTPHeaders, https://addons.mozilla.org/de/firefox/addon/live-http-headers/
http://bit.ly/uYi5cj

M

Microsoft Internet Explorer, http://windows.microsoft.com/en-US/internet-explorer/products/ie/home
http://bit.ly/dMOR0S

Microsoft Windows, http://windows.microsoft.com/en-US/windows/home
http://bit.ly/1Wl1ZM

Most Confusing High Tech Buzzwords: Cloud Computing, Green Washing & Buzzword Compliant",
http://www.languagemonitor.com/about/news/top-10-most-confusing-high-tech-buzzwords030/
http://bit.ly/vG3NfM

Moviepilot, http://moviepilot.com

Mozilla, http://www.mozilla.org/en-US/firefox/new/

N

„News is now yours", Personalisierte News für Bing / Windows Phone 7,
http://www.bing.com/community/site_blogs/b/search/archive/2011/03/12/introducing-project-emporia-powered-matchbox-technology-news-is-now-yours.aspx
http://binged.it/dTDqA0

Nionex, http://cloud.nionex.de/

Node.js, http://nodejs.org

P

„Personalized Search for everyone", http://googleblog.blogspot.com/2009/12/personalized-search-for-everyone.html
http://bit.ly/71RcmJ

Pironet NDH, http://www.pironet-ndh.com

Programming to the Interface in JavaScript, http://knol.google.com/k/programming-to-the-interface-in-javascript-yes-it-can-be-done-er-i-mean-faked#
http://bit.ly/l552jx

Python, http://www.python.org/,
http://en.wikipedia.org/wiki/Python_(programming_language)

Python to Executable, http://www.py2exe.org/

R

Race Condition, http://en.wikipedia.org/wiki/Race_condition

RDFa, Resource Description Framework, http://en.wikipedia.org/wiki/RDFa

Richtlinie 95/46/EG: "Datenschutzrichtlinie",
http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:DE:NOT
http://bit.ly/6a7sY0

S

SDK, Software Developement Kit, http://en.wikipedia.org/wiki/Software_development_kit

SGML, Standard Generalized Markup Language, http://www.w3.org/MarkUp/SGML/

SOA, Service Orientated Architecture, "Service Orientierte Architektur Übersicht und Einordnung", Till Rausch, 2006, http://web.archive.org/web/20081010033719/http:/www.till-rausch.de/assets/baxml/soa_akt.pdf
http://bit.ly/v0ya3O

„Statistical physics of social dynamics" ,Castellano, Fortunato, Loreto, 2009,

T

TyphoonAE, Typhoon App Engine, http://code.google.com/p/typhoonae/

W

W3C, http://www.w3.org/html/logo/

WebGL Specification http://www.khronos.org/registry/webgl/specs/latest/
http://bit.ly/eFqoAU

Websockets, http://dev.w3.org/html5/websockets/

X

XMLHttpRequest, http://www.w3.org/TR/XMLHttpRequest/

8.1 Weitere Quellen

Bilder in Game1

August by Austin The Heller, License: Free for commercial use

Gnome Web Icons by Gnome Project, License: Free for commercial use

CSS Gradient Code

http://www.colorzilla.com/gradient-editor/