TECHNISCHE UNIVERSITÄT DRESDEN

FAKULTÄT INFORMATIK
INSTITUT FÜR SYSTEMARCHITEKTUR
PROFESSUR RECHNERNETZE
PROF. DR. RER. NAT. HABIL. DR. H. C. ALEXANDER SCHILL

Diplomarbeit

zur Erlangung des akademischen Grades
Diplom-Informatiker

Erweiterung des Reputationssystems JASON um Bewerterreputation

Albrecht Uhlig
geboren am 29. Oktober 1980 in Dresden

Dr.-Ing. Stephan Groß, Dr.-Ing. Sandra Steinbrecher

Dresden, 29. März 2010

Aufgabenstellung

Im Zuge des Web 2.0 verschwimmt die Grenze zwischen den Lesern und den Produzenten von Informationen zunehmend. Viele Webseiten erlauben ihren Lesern zur Verfügung gestellte Informationen zu kommentieren oder sogar zu modifizieren, wenn der Inhalt gemeinsam erstellt werden soll. Berühmtestes Beispiel ist sicher Wikipedia (www.wikipedia.org). Dies macht es Lesern schwer, einzuschätzen, welches Vertrauen sie in den Inhalt legen können. Reputations- bzw. Wertungssysteme werden an vielen Stellen im Web bereits eingesetzt, um Besuchern einer Webseite einen Eindruck zu vermitteln, wie andere Besucher Inhalte dieser Webseite bewertet haben. Das auf RDF (Resource Description Framework) basierende Reputationssystem JASON [SGM09] sammelt Wertungen von Besuchern einer Internetseite in Form von Signaturen der Seite und stellt diese anderen Besuchern zur Verfügung.

Ziel dieser Arbeit ist eine Erweiterung des Systems um die Sammlung von Reputation der Bewerter von Webinhalten, um damit die Validität von Bewertungen einschätzen zu können. Im Einzelnen soll sich diese Arbeit folgenden Punkten widmen:

- Analyse des betrachteten Szenarios unter besonderer Berücksichtigung der mehrseitigen Sicherheitsanforderungen der Beteiligten und der Skalierbarkeit des Reputationssystems.
- Überblick über Literatur zur Berücksichtigung von Bewerterreputation bei der Bestimmung von Reputation eines Objekts mit Fokus auf Webinhalte als Objekte
- Softwaretechnisches Re-Design des vorhandenen Reputationssystems mit Erweiterung um Bewerterreputation.
- Erweiterung der vorhandenen Implementierung um die im Design ergänzten Komponenten und beispielhafte Evaluation.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tag dem Prüfungsausschuss der Fakultät Informatik eingereichte Diplomarbeit zum Thema:

Erweiterung des Reputationssystems JASON um Bewerterreputation

vollkommen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Dresden, den 29. März 2010

Albrecht Uhlig

Kurzfassung

Das bestehende Reputationssystem JASON, welches Bewertungen für Web-Inhalte sammelt, soll um Bewerterreputation erweitert werden. JASON [SGM09] ist ein sicheres, skalierbares und plattformunabhängiges Reputationssystem, welches bestehende Technologien des semantischen Web nutzt. Das System wurde bereits im Rahmen einer Diplomarbeit als Prototyp umgesetzt [Mei08]. Das Konzept von JASON wird in dieser Arbeit um Bewerterreputation erweitert und der bestehende Prototyp überarbeitet, um das neue Konzept zu validieren. Dazu werden zunächst die grundlegenden Begriffe für Reputationssysteme definiert und Reputationssysteme im Allgemeinen vorgestellt. Danach führt die Arbeit zwei Arten von Bewerterreputationen ein. Eine subjektive Bewerterreputation die sich jeder Nutzer lokal berechnet und eine objektive Bewerterreputation, die eine globale Aussage über die Kompetenz sein soll. Auf Grundlage das JASON-Konzeptes wird ein System entworfen und prototypisch validiert. Der bestehende Prototyp dient dabei als Grundlage und wird komplett überarbeitet. Die Ergebnisse werden in Form einer Evaluierung durch Nutzertests geprüft und diskutiert.

Inhaltsverzeichnis

1 Einleitung **3**
1.1 Was ist ein Reputationssystem? . . . 3
1.2 Gliederung der Arbeit . . . 6

2 Grundlagen **7**
2.1 Reputationen, Wertungen und Reputationssysteme . . . 7
2.2 Funktionsweise und Aufbau eines Reputationssystems . . . 8
2.3 JASON . . . 10
2.4 Abgrenzung und ähnliche Arbeiten . . . 13

3 Bewerterreputation **16**
3.1 Was ist Bewerterreputation? . . . 16
3.1.1 Subjektive Bewerterreputation . . . 16
3.1.2 Objektive Bewerterreputation . . . 18
3.2 Anforderungen zur Umsetzung . . . 19
3.3 Festlegungen zur Umsetzung Bewerterreputation . . . 21
3.4 Anwendungsfälle . . . 23

4 Integration von Bewerterreputation in JASON **25**
4.1 Datenhaltung . . . 25
4.2 Systemaufbau . . . 29
4.3 Systemverhalten . . . 35
4.4 Sicherheitsanforderungen und Skalierbarkeit . . . 38

5 Prototypische Validierung **43**
5.1 Der erste Prototyp . . . 43
5.2 Der neue Prototyp: JASON-RaterReputation . . . 45
5.2.1 Paketstruktur und Grundlagen . . . 45
5.2.2 Signaturen . . . 46
5.2.3 Datenverwaltung . . . 47
5.2.4 FrontEnd und JasonCore . . . 50
5.2.5 Metriken . . . 52
5.3 Sicherheitsanalyse . . . 56
5.4 Diskussion des Ergebnisses und Validierung . . . 59

6 Zusammenfassung **65**

Literaturverzeichnis **67**

A Nachrichten zwischen Client und Server **70**

B JasonCore API **73**

C Kurzanleitung zum Prototypen **77**

D Evaluierungsbogen **81**

E Beispiele für Daten als XML-Datei **84**

1 Einleitung

Bewertungssysteme in Form von Kritiken in Zeitungen und anderen Medien gibt es schon lange. Ursprünglich gab es eine begrenzte Anzahl von Autoren und Kritikern, da es mit einem vergleichsweise großen Aufwand verbunden war, in den Medien zu publizieren. Im Internet ist es jedoch sehr einfach, zu veröffentlichen. In einer Entwicklung, die meist als Web2.0 bezeichnet wird, kann fasst jeder zum Autor werden. Es liegt nahe, dass auch jeder zum Kritiker werden kann. Um diese Fülle von Informationen und die große Menge an potentiellen Bewertern (Kritikern) zu organisieren, gibt es Systeme, welche im Folgenden Reputationssysteme genannt werden.

Reputation kann eine Person in Bezug auf ihr bisheriges Handeln erwerben. In dieser Arbeit geht es um Bewerterreputation, also das bisherige Handeln einer Person in ihrer Rolle als Bewerter. In der Welt außerhalb des Internets, besonders im Bereich von Theater, Film und Kunst gibt es Personen, die nur durch Bewerten zu beachtlichem Ruf gelangen: Kritiker haben auf den Erfolg von einem Theaterstück oder einem Film großen Einfluss. Dabei ist es schwer zu sagen, wie man Kritiker wird und was einen Kritiker ausmacht. In [Boe08] findet sich eine Untersuchung zu den Praktiken von Theaterkritik. Es zeigt sich, dass es keine Standards für Kritiken oder Kritikerausbildungen gibt. Eine weitere Art von Bewerterreputation wird in [GTW09] untersucht. Dort wird gezeigt, dass Robert Parker, ein berühmter Weinkritiker, allein durch seine Wertung und seine hohe Reputation als Werter erheblichen Einfluss auf den Preis von Weinen hat. Bewerterreputation scheint also wichtig zu sein, ohne dass man genau sagen kann, wie sie entsteht. In dieser Arbeit wird deshalb eine eigene Definition von Bewerterreputation in Kapitel 3 erarbeitet.

Ziel dieser Arbeit ist eine Erweiterung des Reputationssystems JASON, um die Sammlung von Reputation der Bewerter von Webinhalten, und um damit die Validität von Bewertungen einschätzen zu können. Das folgenden Unterkapitel gibt eine Einführung in die Thematik der Reputationssysteme. Es werden drei bestehende Reputationssysteme als Beispiel aufgeführt.

1.1 Was ist ein Reputationssystem?

Die beiden wichtigsten Begriffe in einem Reputationssystem sind die Reputation und das daraus entstehende Vertrauen. Für beide Begriffe gibt es viele Definitionen [MMH02], vorerst genügt eine textuelle Beschreibung aus [Vos04]:

> Reputation ist die gesammelte Information über ein Reputationsobjekt, wie es von anderen wahrgenommen wird. Vertrauen ist eine Entscheidung, die aufgrund der Reputation getroffen wird.

Das Reputationsobjekt kann eine Person sein. Die Information über diese Person ergibt sich dann aus ihrem Handeln. Das Reputationsobjekt kann aber auch eine Einrichtung, ein Gegenstand, ein Produkt, eine Idee oder ein Inhalt sein.

Es stellt sich die Frage, wie man die Informationen sammeln kann und wie man diese Informationen interpretieren kann. Dafür gibt es objektive und subjektive Ansätze. In subjektiven Systemen teilt der Nutzer dem System seine Meinung in Form einer Bewertung über das Reputationsobjekt mit. Aus einer Sammlung dieser subjektiven Wertungen wird die Reputation berechnet. Bekannte Beispiele für subjektive Reputationssysteme findet man bei eBay und Amazon. Bei eBay wird das Handeln anderer bewertet. Bei Amazon geht es vor allem um die Bewertung von Produkten und Händlern. Dabei sind auch Bewertungen von Bewertungen möglich. Abbildung 1.1 und 1.2 zeigen Beispiele des Bewertungssystems bei Amazon. In Abbildung 1.1 sieht man den Dialog zum Bewerten eines Händlers. Man vergibt sowohl eine Note als auch einen Text und einige Zusatzinformationen. Die nächste Abbildung zeigt einen Ausschnitt der Rezensionsanzeige von Amazon. Es werden zwei Bewertungen gezeigt, die als besonders nützlich eingestuft wurden. Dazu wird die nützlichste gute und die nützlichste schlechte Bewertung gewählt. Dabei wird auch das Alter der Bewertungen und somit die Aktualität angezeigt.

Bei objektiven Wertungssystemen werden festgelegte Kriterien für die Berechnung verwendet, die sich aus den Daten des Systems berechnen lassen. Ein bekanntes Beispiel ist das Elo-System [elo], das es erlaubt, die Spielstärke von Spielern zu beschreiben (z. B. bei Schach oder Go). Dabei werden, nachdem zwei Parteien ein Spiel abgeschlossen haben, aus den alten Spielstärken neue, in Abhängigkeit vom Ausgang der Partie, berechnet. Systeme wie das Elo-System liefern objektive Ergebnisse, lassen sich aber nur auf spezielle Gebiete anwenden (z. B. die Spielstärke in einem Spiel). Abbildung 1.3 zeigt ein Beispiel aus dem Bereich des E-Sport. Man sieht die Wertung (Rating) der Teams, ihren Aufbau und ihr Sieg-Niederlage-Verhältnis.

Abbildung 1.1: Amazon: Dialog zum Bewerten eines Händlers.

Abbildung 1.2: Amazon: Ausschnitt aus der Kundenrezensionsanzeige. Zwei der am besten bewerteten Rezensionen werden zuerst angezeigt.

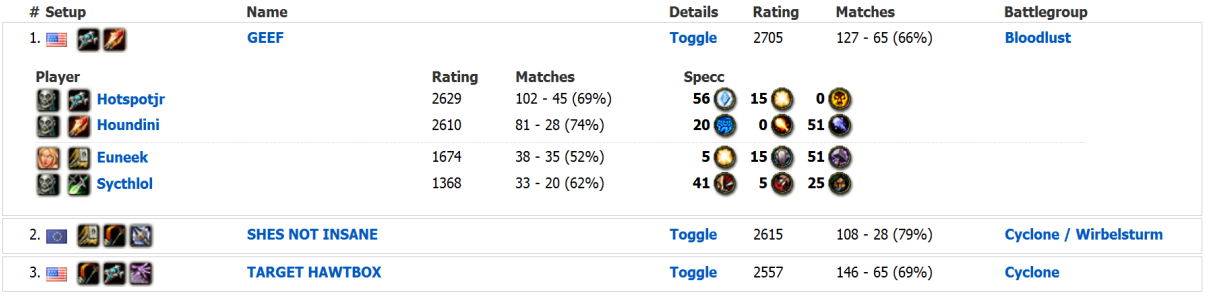

Abbildung 1.3: E-Sport: Objektives Bewertungssystem ähnlich dem Elo-System.

Reputationssysteme sollen ihre Teilnehmer ermutigen, sich vertrauenswürdig zu verhalten, indem Informationen über ihr Handeln gesammelt werden. Der Nutzer, in seiner Rolle als Werter, ist bei den zur Zeit eingesetzten Systemen meist kein Teil der Reputationsobjekte (Objekte, zu denen eine Reputation berechnet wird). Ein Ansatz, der in diese Richtung geht, ist die Rezensionsbewertung bei Amazon. Ziel dieser Arbeit ist es, ein konkretes Reputationssystem (JASON, siehe Kapitel 2.3) zu erweitern, um die Möglichkeit, eine Reputation für die Werter zu berechnen.

Die Bewerterreputation soll helfen, eine Wertung besser einzuschätzen. Nur anhand der Wertung kann man nicht entscheiden, ob der Erzeuger der Wertung kompetent und vertrauenswürdig ist. Das Sammeln von Informationen über das Verhalten des Wertungserzeugers kann helfen, diese Fragen zu beantworten.

1.2 Gliederung der Arbeit

Kapitel 2 definiert die theoretischen Grundlagen und stellt das Reputationssystem JASON vor, auf dem diese Arbeit aufbaut. Kapitel 3 beschäftigt sich mit der Analyse der Aufgabe. Es werden zwei Arten von Bewerterreputation definiert und Anforderungen zur Umsetzung, sowie Anwendungsfälle angegeben.

In Kapitel 4 wird ein JASON-System mit den neuen Anforderungen entworfen. Dabei wird ein komplettes Redesign des vorhanden Prototypen vorgenommen. Das Kapitel beschreibt den statischen Systemaufbau, die Datenhaltung und die wichtigsten Systemfunktionen. Außerdem befindet sich in diesem Kapitel eine Diskussion über Sicherheit und Skalierbarkeit des Entwurfs.

Die Implementierung wird in Kapitel 5 detailiert beschrieben. Da sie stark auf dem ersten Prototypen aufbaut, wird dieser erste Prototyp zunächst vorgestellt. Danach wird der Aufbau und die Funktionsweise des neuen Prototypen erläutert. Zu den wichtigsten Funktionen (z. B. die Berechnung der Reputation) werden Code-Teile angegeben und auf einige Probleme bei der Implementierung eingegangen. Technische Details zur Implementierung befinden sich in Anhang A und B. Eine Anwenderdokumentation befindet sich in Anhang C. Anhand des fertigen Prototypen, werden konkret für diesen Prototypen Angriffe diskutiert. Am Ende des Kapitels befindet sich ein Ausblick, was zukünftige Arbeiten am Systementwurf und Implementierung verbessern können und die Auswertung der vorgenommenen Evaluierung (Evaluierungsbogen in Anhang D).

2 Grundlagen

2.1 Reputationen, Wertungen und Reputationssysteme

Je nach Reputationssystem wird Reputation unterschiedlich definiert. Auch die Wertungen werden in unterschiedlicher Form abgegeben. Um eine Grundlage für diese Arbeit zu schaffen, legen die folgenden Abschnitte einige Definitionen fest. Dabei ist W die Menge der möglichen Wertungen, die ein Bewerter vergeben kann. Die Menge aller möglichen Tupel von Wertungen wird durch W^* bezeichnet. Die Reputation ist eine Abbildung von einem Wertungstupel auf einen Reputationswert:

Definition 1 (Reputation) *Reputation bildet ein Tupel von Wertungen $(w_1, \ldots, w_n) \in W^*$, die sich auf ein Reputationsobjekt B beziehen, auf einen Reputationswert $r \in R$ ab. Die Menge W ist die Menge der möglichen Wertungen. Die Menge R ist die Menge der möglichen Reputationswerte. Die Reputation ist ein subjektiver Wert aus Sicht einer Person A.*

$$rep_A(B) : W^* \rightarrow R$$

Wertungen können objektiv nach globalen Kriterien oder subjektiv aus Sicht eines Nutzers abgegeben werden. Für diese Arbeit wird die subjektive Variante genutzt, da es schwierig ist, objektive Bewertungskriterien zu definieren. Ein Mensch kann sich zu jedem Thema eine subjektive Meinung bilden. Diese Meinung gibt er in Form einer Wertung beim Reputationssystem ab. Kombiniert man viele subjektive Meinungen können sehr gute Ergebnisse erzielt werden (siehe Kapitel 3.1 und [Sur05]). Es ergibt sich folgende Definition für eine Wertung:

Definition 2 (Wertung) *Eine Wertung ist die Abbildung eines Nutzers A, mit der er seine Erfahrungen über ein Reputationsobjekt B einer Wertung $w \in W_A$ zuordnet. W_A ist die Menge der möglichen Wertungen die sich jeder Nutzer A individuell definiert. Bezeichnet man die Menge der möglichen Erfahrungen mit C, kann man den Vorgang als Funktion beschreiben:*

$$rating_A(B) : C \rightarrow W_A$$

In einem bestimmten Kontext macht A Erfahrungen mit B, woraus er sich eine Meinung zu B bildet. Diese Meinung übersetzt er maschinell oder intuitiv zu einer Wertung w. In einem Reputationssystem werden diese Wertungen gesammelt und bilden die Informationen, aus denen sich die Reputation berechnen lässt. Ein Reputationssystem lässt sich demnach wie folgt definieren:

Definition 3 (Reputationssystem) *Ein Reputationssystem sammelt und verarbeitet Wertungen, welche es dem Nutzer des Systems in Form von Reputationen zur Verfügung stellt.*

Diese Definitionen sind sehr allgemein gehalten und können unterschiedlich umgesetzt werden. Bevor in Kapitel 2.3 erklärt wird, wie das JASON-System Wertungen und Reputation umsetzt, wird im nächsten Kapitel der allgemeine Aufbau und die Funktionsweise von Reputationssytemen untersucht.

2.2 Funktionsweise und Aufbau eines Reputationssystems

Es gibt inzwischen ein breites Spektrum an Reputationssytemen im Internet. Einen guten Überblick bietet [JIB07]. [Mal01] betrachtet ebenfalls bestehende Reputationssysteme und identifiziert deren Probleme. Dieses Kapitel gibt eine allgemeine Einführung zum Aufbau und der Funktionsweise. Man kann die Systeme in drei Aufgabenbereiche einteilen. Abbildung 2.1 zeigt den allgemeinen Aufbau eines Reputationssystems nach [Vos04]. Zu sehen sind zwei Akteure A und B, drei Aufgabenbereiche und Pfeile, die den Informationsfluss darstellen. Die Reihenfolge der Aktionen (Pfeile) ist nicht festgelegt. Es ist ein ständiger Prozess und Kreislauf, bei dem sich alle Größen gegenseitig beeinflussen.

Die drei Aufgabenbereiche sind das Sammeln von Wertungen, Verarbeiten dieser und Verteilen der Ergebnisse. Nutzer des Systems geben ihre Wertungen beim Sammel-Block ab. Von dort aus werden die Wertungen in den Bestand der Datensammlung aufgenommen und verarbeitet. Der Update-Pfeil stellt die Dynamik dieser Daten dar, da ständig neue Wertungen hinzu kommen. Schließlich müssen die Daten in Form von Reputation abrufbar sein. Das ist der letzte Funktionsblock, Verteilen. Bei realen Reputationssystemen sind diese Blöcke nicht immer voneinander getrennt. Wo welche Aufgaben abgearbeitet werden, hängt von der Architektur des Systems ab. Verschiedene Möglichkeiten werden im Folgenden vorgestellt.

A ist in der Abbildung der Bewerter und B das bewertete Reputationsobjekt. Zwischen A und B muss es irgendeine Art von Kontakt geben. A muss Erfahrungen mit B gemacht haben, die er bewertet. Diese Erfahrungen werden in der Abbildung mit C bezeichnet. Wenn B ein Verkäufer ist, könnte C ein Verkauf an A sein, den A dann bewertet. Wenn B ein Text ist, dann ist C der Eindruck von A nach dem Lesen des Textes. A kann sich vom System die Reputation von B anzeigen lassen. Das wird durch die Funktion $Rep_A(B)$ dargestellt. Die Wertung der Erfahrung von A wird durch die Funktion $Rating_A(B)$ dargestellt. Meist wird C auf einige bestimmte Bereiche eingeschränkt. Abbildung 1.1 zeigt z. B. die Bereiche, die bei Amazon bewertet werden können.

Wie man die Funktionen $Rep_A(B)$ und $Rating_A(B)$ genau berechnet, ist nicht Gegenstand dieser Arbeit. Über die Berechnung von Reputationen, Wertungen und der Nützlichkeit gibt es viele Meinungen. [MMH02] versucht einen Überblick aus verschiedenen Fachbereichen zu geben und ein Modell abzuleiten. Trotzdem wird es immer unterschiedliche Meinungen zu diesem Thema geben. Deshalb sollte sich eine technische Betrachtung von Reputationssystemen damit beschäftigen, welche Daten diese Funktionen benötigen und wie sie gesammelt, verarbeitet und verteilt werden können. Die konkreten Funktionen sollten transparent und austauschbar sein. Transparent, damit der Nutzer die Werte am Ende richtig interpretieren kann, und austauschbar, um sich dem Einsatzgebiet und neuen Entwicklungen anpassen zu können.

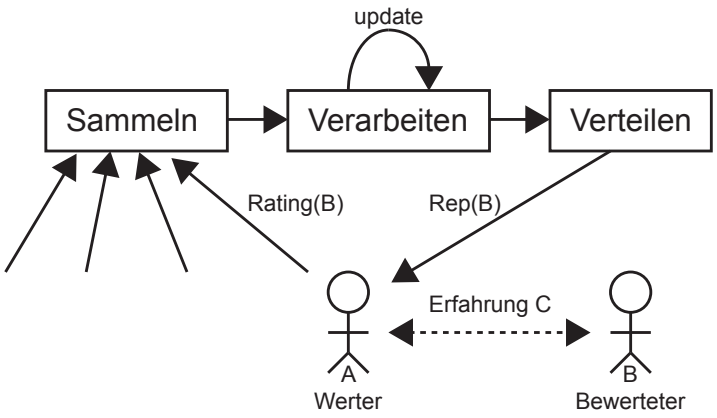

Abbildung 2.1: Funktionsblöcke und Akteure in einem Reputationssystem.

In den letzten Absätzen wurde erklärt, wie ein Reputationssystem funktioniert. Im Folgenden werden verschiedene Möglichkeiten des Aufbaus vorgestellt. Klassifikation und Abbildung richten sich wiederum nach [Vos04]. Hauptkriterium bei der Klassifikation ist die Datenhaltung: Wo sind welche Daten gespeichert und wer hat die Kontrolle darüber? Besonders für die Sicherheitsaspekte und den Datenschutz ist der Ort der Datenhaltung wichtig. Es lassen sich vier verschiedene Arten unterscheiden:

- zentral
- subjektiv, ohne Weitergabe
- verteilt, mit Weitergabe
- lokal

Abbildung 2.2 zeigt eine grafische Darstellung.

Die bekanntesten Reputationssysteme sind zentral organisiert. Die angesprochenen Beispiele eBay und Amazon gehören dazu. Zentrale Systeme sind einfach zu organisieren und gut für den Betreiber des Systems. Dieser hat die volle Kontrolle über alle Daten. Der Nutzer ist normalerweise gegenüber anderen Nutzern geschützt, muss aber dem Systemanbieter vertrauen. Im angesprochenen Fall von eBay und Amazon muss ein Nutzer ohnehin dem System vertrauen, und es wäre auch sehr schädlich für den Anbieter, wenn Datenmissbrauch öffentlich wird. Trotzdem sammeln sich viele Daten an, über die nur der Systemanbieter die Kontrolle hat.

Die zweite Möglichkeit sind subjektive Systeme ohne Weitergabe von Daten. Dabei sammelt jeder Nutzer seine persönliche Sicht der Dinge. Er speichert seine Erfahrungen, die er über andere Nutzer gesammelt hat. Daraus bildet er deren Reputation. Das funktioniert gut bei häufigen Kontakten mit den gleichen Nutzern. Nur so können ausreichend Daten für eine Beurteilung gesammelt werden.

Erweitert man den zweiten Typ um eine Weitergabe von Daten, kommt man zu den verteilten Systemen. Hierbei wird die Reputation zwischen den Nutzern ausgetauscht. So können auch unbekannte Nutzer beurteilt werden. Dabei sind transitive Ketten möglich. Es ergeben sich viele Ansätze für das Wichten und Berechnen der Reputation. Diese Berechnungen sind wie bereits erwähnt nicht Teil dieser Arbeit.

Die letzte Möglichkeit ist die lokale Speicherung von Reputation. Jeder Nutzer hält dabei seine eigenen

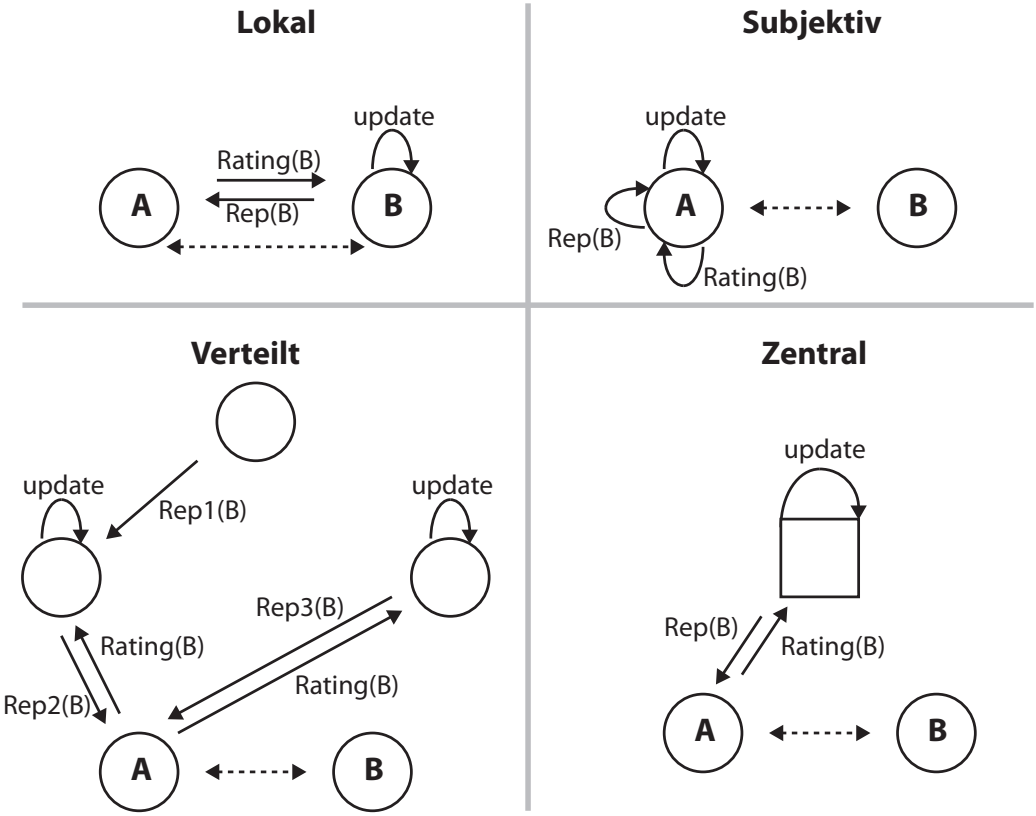

Abbildung 2.2: Klassifikation von Reputationssystemen nach [Vos04].

Reputationsdaten bei sich. Der Nutzer muss beweisen können, dass die Daten echt sind, was durch die Architektur eines solchen Systems ermöglicht werden muss.

2.3 JASON

JASON [SGM09] ist ein sicheres, skalierbares und plattformunabhängiges Reputationssystem, welches bestehende Technologien des semantischen Web nutzt. Es ist durch die Entwicklungen des Web2.0 motiviert, bei der jeder Nutzer zum Autor werden kann. Bei der Fülle an unbekannten Autoren wird es schwer, Inhalten zu vertrauen. JASON beschreibt eine Möglichkeit, jedem Inhalt seine Wertungen als Metainformation mit Hilfe des Resource Description Framework anzuhängen. Diese gesammelten Wertungen bilden die Reputation des Inhaltes.

Die in Kapitel 2.1 eingeführte Menge R der möglichen Reputationswerte ist bei JASON die gleiche wie die Menge W der Wertungen. Das bedeutet, das System berechnet für den Inhalt eine Gesamt-Wertung $winW$ aus allen verfügbaren Einzel-Wertungen. Diese berechnete Wertung stellt den Reputationswert des Inhaltes dar. Jeder Nutzer definiert sich seine eigene Menge W von Wertungen. Fremde Wertungen werden auf die eigenen abgebildet. So wird eine Bedeutung für die fremde Wertung definiert.

Die drei wichtigsten Kernpunkte von JASON sind:

- Dezentral: keine zentrale Instanz nötig, Wertungen als Metadaten am Inhalt

- Wertung = Signatur: Wertung ist eine Signatur mit einem Schlüssel, der eine feste Bedeutung hat, Schlüssel gehören zu einem Pseudonym
- Soziales Netz: Weitergabe der öffentlichen Wertungsschlüssel samt Bedeutung innerhalb des sozialen Netzes eines Nutzers

Es wird keine zentrale Instanz benötigt. Das ist besonders günstig für Sicherheitsanforderungen und Datenschutz. Es gibt keinen zentralen Angriffspunkt, der alle Nutzer auf einmal gefährdet, und man hat die Wahl, welchen Teilnehmern man vertrauen möchte und welchen nicht. Für das Thema dieser Arbeit, die Erweiterung um Bewerterreputationen, wäre eine zentrale Instanz sehr hilfreich. Das gleiche gilt für die Erweiterung um Autorenreputationen, welche in [Bar10] untersucht wird. Die Dezentralität ist aber einer der wichtigen Kernpunkte von JASON. Eine zentrale Einrichtung zu verlangen, bedeutet keine Erweiterung des Systems, sondern ein anderes System und sicherheitstechnisch einen Nachteil (siehe Kapitel 2.2).

Die Bewertungen werden in Form von Metadaten beim bewerteten Inhalt gespeichert. Eine Wertung ist eine Signatur des Inhaltes mit einem privaten Schlüssel. Der zugehörige öffentliche Schlüssel ist für jeden zugänglich, nicht aber seine Bedeutung. Jeder kann also prüfen, ob eine Wertung gültig ist, aber Wertungen interpretieren kann man nur, wenn man die Bedeutung des öffentlichen Schlüssels kennt. Jeder Nutzer kann sich beliebige Bedeutungen und zugehörige Schlüssel definieren.

Die öffentlichen Schlüssel werden dem Nutzer nicht direkt zugeordnet, sondern einem seiner Pseudonyme. Der Nutzer kann sich für verschiedene Rollen und Kontexte verschiedene Pseudonyme zulegen. Dadurch kann er in verschiedenen Bereichen werten und Inhalt produzieren, ohne dass man ein Profil über ihn erstellen kann. Da jede Wertung einen anderen Schlüssel hat, können sogar verschiedene Wertungen von ein und demselben Pseudonym nur in Verbindung gebracht werden, wenn man die entsprechenden öffentlichen Schlüssel kennt.

Damit man die Wertungen anderer Nutzer interpretieren kann, muss man die öffentlichen Schlüssel ihrer Wertungen und ihre Bedeutungen kennen. Dazu sieht JASON die Nutzung von sozialen Netzwerken vor. Jeder kann entscheiden, welche Schlüssel er samt ihrer Bedeutung weitergibt. Dabei muss er nicht alle Schlüssel weitergeben, sondern kann z. B. nur gute Bewertungen veröffentlichen, während er die Schlüssel für die schlechten Wertungen nur seinen engeren Freunden gibt. Diese könnten die Schlüssel natürlich weitergeben. Es ist also eine Frage des Vertrauens, wem man welche Schlüssel gibt. Die Bedeutungen von fremden Schlüsseln kann sich jeder Nutzer selbst umdefinieren. Hier fließt das Wissen der Nutzer über andere Teilnehmer im sozialen Netzwerk mit ein. Es können auch transitive Vertrauensstrukturen verwendet werden. Wenn man andere Nutzer indirekt über Freunde kennt, kann man auch für sie einen Vertrauenswert berechnen.

Abbildung 2.3 zeigt grob den Aufbau eines JASON-Systems. Der Kern der Anwendung, *JasonCore* verwaltet die Bekanntschaften und Pseudonyme. Dort wird auch die Metrik implementiert und werden alle vertraulichen Operationen vorgenommen, wie z. B. das Erstellen einer Wertung. Der Kern hat Zugriff auf zwei Datenbanken. Eine verwaltet die Pseudonyme. In einem fertigen System sollte hier ein datenschutzfreundliches Identitäts-Management-System (PE-IMS [CPHH02]) verwendet werden. Die zweite Datenbank beinhaltet alle Daten über die Bekannten eines Nutzers (z. B. Trust-Werte). Der Kern kommuniziert mit dem $FrontEnd$ des Systems. Dort werden die Reputationsdaten der Inhalte angezeigt.

Die Werte sollten möglichst nah am betreffenden Inhalt angezeigt werden. Um Wertungen und eigene öffentliche Schlüssel zu veröffentlichen, benötigt der Kern eine Anbindung an das *Storage Backend* (z. B. ein Webserver, auf dem der Nutzer seine Inhalte veröffentlicht). Wertungen können so an die Metadaten eines Inhaltes angehängt und die eigenen öffentlichen Daten automatisch publiziert werden.

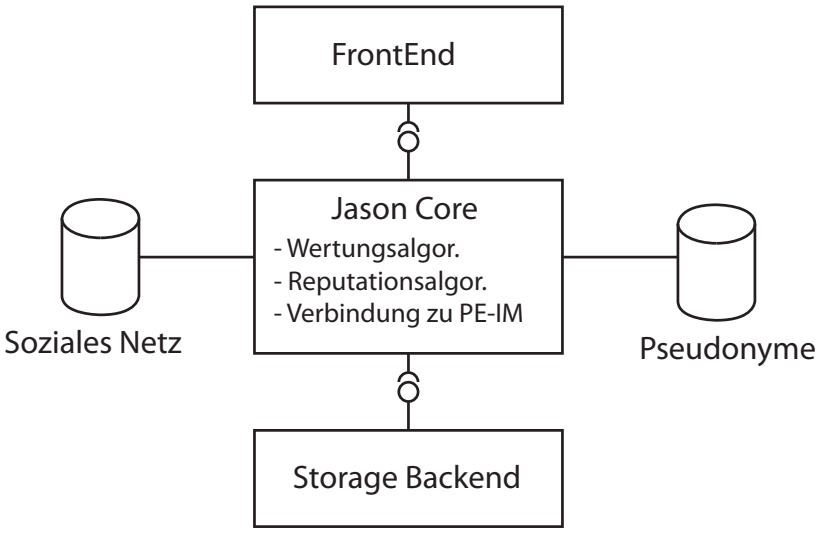

Abbildung 2.3: Aufbau eines JASON-Systems.

Abbildung 2.4 zeigt eine Einordnung von JASON nach der Kategorisierung aus Kapitel 2.2 (Abbildung 2.2). Die Strichmännlein stellen jeweils ein Pseudonym dar. Das Reputationsobjekt B ist ein Inhalt, der von einem Nutzer des Systems erzeugt wird. Nutzer A bekommt von verschiedenen anderen Nutzern öffentliche Wertungsschlüssel. Mit Hilfe dieser Schlüssel kann er die Wertungen am Inhalt B interpretieren und sich seine eigene Funktion $Rep(B)$ definieren. Diese Funktion hängt von seinem Vertrauen gegenüber den anderen Nutzern, deren Bewertungen er kennt, ab. Jeder Nutzer erhält also einen eigenen individuellen Reputationswert, abhängig von seinem Bekanntenkreis. Wenn A dem Inhalt eine Wertung gibt ($Rating_A(B)$), wird diese dem Inhalt als Metainformation hinzugefügt.

Das System verbindet die Vorteile von lokalen und verteilten Reputationssystemen. Die Wertungen werden beim Inhalt gespeichert. Die Authentizität der Wertungen wird durch Public-Key-Kryptographie sichergestellt und die Vollständigkeit wird durch die Systemarchitektur motiviert. Da der Besitzer des Inhaltes nicht sehen kann, ob die Wertungsdaten gute oder schlechte Wertungen sind, wird er hoffentlich alle veröffentlichen. Aus Sicht der Datenhaltung ist es also ein lokales System. Die Weitergabe der öffentlichen Wertungsschlüssel im Trustnetzwerk ist eine Eigenschaft von verteilten Reputationssystemen. Der Unterschied ist, dass keine speziellen Reputationsdaten weitergegeben werden, sondern Schlüsselzuordnungen, die es dem Nutzer A ermöglichen, die Wertungen beim Inhalt zu interpretieren. Man behält so die Sicherheitsvorteile von lokalen Systemen und gewinnt die Vorteile (Informationsweitergabe) von verteilten Systemen.

Das hier beschriebene System lässt sehr viele Freiheiten zu. Der Rating-Algorithmus und der Reputations-Algorithmus werden nicht festgelegt. Jeder Nutzer kann sich seine eigene Bewertungsmetrik definieren. Einzige Bedingung ist, dass sich die Wertungsschlüssel auf Werte in der Metrik abbilden lassen. Eine

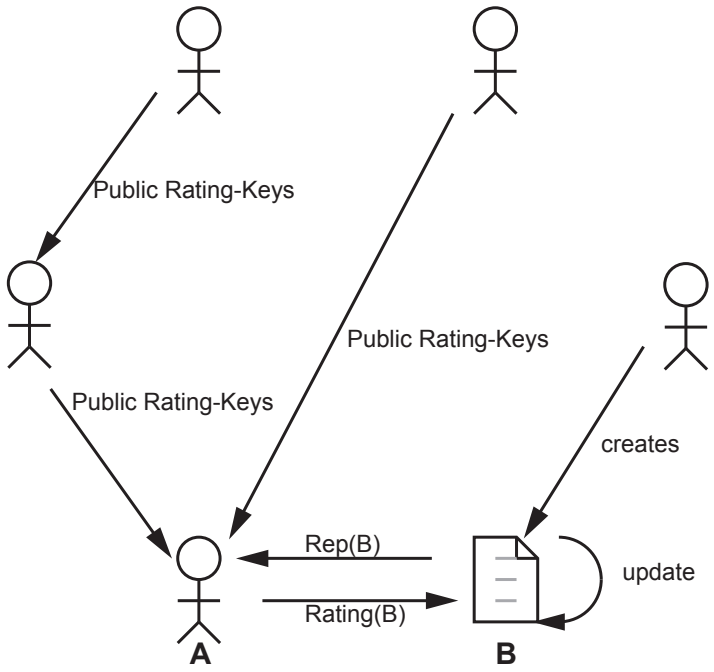

Abbildung 2.4: Einordnung von JASON in die Klassifikation von Voss: Lokale Datenhaltung bei verteiltem Informationsfluss.

einfache Metrik könnten z. B. Noten von 1 bis 6 sein. Für jede Note gibt es dann einen Schlüssel. Die Metrik muss definieren, wie mehrere dieser Noten zu einem Reputationswert zusammengefasst werden. Auch mehrdimensionale Metriken sind denkbar. Wenn ein Nutzer die Bedeutung seiner Wertungsschlüssel weitergibt, muss die Information, zu welcher Metrik die Bedeutung gehört, enthalten sein.

Es gibt weitere Freiheiten bei JASON. Die Signaturen müssen nicht nur die Verschlüsselung eines Hash sein, sondern könnten auch Text mit zusätzlichen Wertungsinformationen enthalten. Trust-Netzwerk und Identitäts-Management-System sollen bestehende Lösungen verwenden. Auch hier wird nicht festgelegt, wie es in der Praxis genau aussehen soll. Im Folgenden wird untersucht, wie man das Konzept um Bewerterreputationen erweitern kann.

Ziel ist es, eine bessere Einschätzung von unbekannten Webseiten zu erreichen. Das soll durch mehr Informationen über die Wertungen, aus denen die Reputation berechnet wird, erreicht werden. Der Ansatz den diese Arbeit verfolgt, ist das Sammeln von Informationen über die Erzeuger der Wertungen. Dadurch soll die Validität der Wertungen besser einschätzbar sein. Das folgende Kapitel gibt einen Überblick zu ähnlichen Arbeiten und vergleicht deren Ansätze.

2.4 Abgrenzung und ähnliche Arbeiten

Den Ansatz, Wertungen besser einzuschätzen, indem man den Wertungserzeuger betrachtet findet man in vielen Arbeiten. Einheitliche Bezeichnungen lassen sich nicht finden. Es wird von Punkten, Indize, Bewerterreputation usw. gesprochen. Meist werden sehr konkrete Bewertungsmetriken benutzt und die

Systeme nur innerhalb einer Internetseite verwendet oder untersucht. Diese Arbeit bezieht sich nicht auf eine bestimmte Wertungsmetrik oder eine festgelegt Webseite. Das entwickelte System ist für allgemeinen Web-Content vorgesehen, dafür weniger detailiert als die meisten der ähnlichen Arbeiten, die in den folgenden Abschnitten vorgestellt werden. Die Abschnitte verweisen zum Teil bereits auf die beiden Arten von Bewerterreputationen, die im nächsten Kapitel 3.1 beschrieben werden.

[CS01] hat ein ähnliches Ziel wie diese Arbeit. Es sollen bessere Reputationswerte berechnet werden, indem Bewerterreputation verwendet wird. Außerdem wird ein *Confidence-Level* für jeden Wert berechnet. Die Bewerterreputatoin wird in Kompetenzbereiche unterteilt. Zur Berechnung werden zunächst für jedes Reputationsobjekt die numerischen Wertungen verglichen. Wer so ähnlich wie die meisten anderen wertet, bekommt einen guten Wert. Desweiteren werden Textkommentare betrachtet, indem andere Nutzer diese Kommentare bewerten. Die berechneten Werte für die numerischen Wertungen und die Textkommentare werden dann über alle Reputationsobjekte aus einem Kompetenzbereich zusammengeführt. So erhält man die Bewerterreputation, welche als Gewichtung für die Berechnunge der Reputation eines Reputationsobjektes verwendet wird. Das Konzept wird anhand echter Daten von bestehenden Internetseiten (EBay, Yahoo, Amazon, Epionions) untersucht und verifiziert. Das Paper legt den Schwerpunkt auf Berechnungsalgorithmen, die nicht Fokus dieser Arbeit sind.

[FN03] schlägt vor, dass diejenigen eine hohe Bewerterreputation bekommen, die Reputationsobjekte mit einer guten Reputation auch gut bewertet haben. Dabei bekommt man besonders dann eine hohe Bewerterreputation, wenn es noch wenige andere Wertungen gibt. Nachahmer können also nicht einfach gute Wertungen abgeben, um eine eigene gute Bewerterreputation zu bekommen. Bei diesem System werden die Reputation der Reputationsobjekte und die Bewerterreputation abwechselnd gegenseitig aus dem anderen Wert berechnet. Damit sich die Werte nicht beliebig lange ändern, wird nach einer festgelegten Zeit der Reputationswert eingefroren. Das System wurde in einem dreimonatigen Experiment validiert.

Beide Ansätze versuchen Trust-Werte für Werter zu berechnen, um daraus Schlüsse auf die Qualität ihrer Wertungen zu ziehen. Dabei wird der Trust-Wert zum Wichten der Wertung verwendet. Weitere Ansätze, um bessere Trust-Werte zu bekommen, befinden sich z. B. in [VHP08] und [ZG07]. Das erste Paper untersucht ungewollt schlechtes Verhalten von Nutzern und die negativen Auswirkung auf deren Bewerterreputation. Es werden verschiedene Möglichkeiten vorgeschlagen, um die geschädigte Bewerterreputation wieder herzustellen. Da Reputationssysteme Nutzer zu positivem Verhalten motivieren sollen, scheint dieser Ansatz sinnvoll. Es ermöglicht dem Nutzer sein Verhalten zu verbessern. Das zweite Paper untersucht, ob es Zusammenhänge zwischen Vertrauen zwischen Nutzern und ihren Interessen gibt. Zumindest anhand der untersuchten Daten, kann dieser Zusammenhang gezeigt werden. Dieser Zusammenhang motiviert die in Kapitel 3.1 eingeführte subjektive Bewerterreputation, bei der ähnliche Wertungen (bedeuten wahrscheinlich auch ähnliche Interessen) zu einer hohen Bewerterreputation führen.

[XL04] beschreibt ein dezentrales Reputationssystem. Es zeigt einige Ähnlichkeiten zu JASON, basiert aber auf Transaktionen und nicht auf Inhalten. Ziel ist es, einen Trust-Wert für andere Systemteilnehmer zu berechnen. Die Idee ist die gleiche, wie bei der objektiven Bewerterreputation, die in dieser Arbeit eingeführt wird. Es soll durch Informationen von dritten Systemteilnehmern eingeschätzt werden, wie vertrauenswürdig der andere ist. Dazu werden in dem Paper fünf Faktoren zur Berechnung eines Trust-

Wertes verwendet. Diese Werte sind: das Feedback über die Transaktionen und die Glaubwürdigkeit der FeedBack-Erzeuger, die Anzahl der Transaktionen und die Kontextfaktoren der Transaktion und der Community. Anhand dieser Werte wird eine Trust-Metrik eingeführt und eine mögliche verteilte Datenhaltung beschrieben. Sicherheit soll durch Redundanz der Daten und digitale Signaturen gewährleistet werden. Das Konzept wird ausführlich evaluiert. Es kann Robustheit gegenüber bösartigen Teilnehmer gezeigt werden und eine Anpassung an das dynamisches Verhalten der Systemteilnehmer.

Einen ähnlichen Ansatz für Web-Services beschreibt [MB09]. [SP04] verwendet ein teilweise dezentrales System, bei dem die Reputation der Teilnehmer durch ein neuronales Netz berechnet wird, welches von einer zentralen Komponente trainiert wird. [ZC08] kombiniert mehrere der bisher angesprochenen Konzepte. Es beschreibt ein System für den Bereich des E-Commerce mit besonderer Betrachtung von unfairen Wertungen. Es werden bestehende Systeme untersucht und nach global / lokal und privat / öffentlich kategorisiert. Das in diesem Paper entwickelte System ähnelt der subjektiven Bewerterreputation aus dieser Arbeit. Nutzer berechnen sich individuelle Reputationswerte, indem sie ihre Wertungen mit den Wertungen von anderen vergleichen. Die Idee ist die gleiche wie im bereits angesprochen Paper [ZG07] (ähnliche Meinungen korrelieren mit dem Vertrauen). Gleichzeitig können auch globale Reputationswerte verwendet werden, wenn sich nicht genug Vergleiche bilden lassen. Das System scheint für den E-Commerce-Bereich viele der bisherigen Konzepte erfolgreich zu vereinen. [CKP09] stellt ein System für den gleichen Anwendungsbereich vor. Aus den Wertungen der Nutzer werden mehrdimensionale Bewerterreputationen berechnet. Es wird unterschieden nach Kompetenz, Zuverlässigkeit und Ähnlichkeit. Die Evaluierung konnte dem System eine gute Performance bescheinigen.

Alle vorgestellten Arbeiten berechnen Werte für Systemteilnehmer mit dem Ziel, deren Handlungen besser einzuschätzen. Der Unterschied zu dieser Arbeit ist, dass die Ansätze konkreter sind. Es handelt sich meist um festgelegte Bereiche wie E-Commerce, Web-Services oder Transaktion in P2P-Netzwerken. Dadurch sind die entwickelten Systeme näher an einem produktiven Einsatz. JASON beschreibt ein System für alle Art von Web-Inhalten unabhängig von einer festen Seite. Dabei wird festgelegt, welche Daten zur Berechnung der Reputation gesammelt werden, nicht aber wie diese Berechnung genau durchgeführt wird. Einige der angesprochenen Arbeiten bieten hier gute Ansätze für Berechnungsalgorithmen. JASON ist also allgemeiner, muss aber für den produktiven Einsatz noch weiterentwickelt werden. Das grösste Problem ist dabei wahrscheinlich die Anpassung an die Dynamik von Web-Inhalten. Jede Veränderung der Inhalte lässt die Wertungen ungültig werden, da diese aus digitalen Signaturen bestehen.

Eine Abgrenzung gegen die genannten Arbeiten ist schwierig, da sie alle unterschiedliche Begriffe verwenden, die sich teilweise ähneln. Folgende Liste versucht die Besonderheiten von JASON und der Erweiterung um Bewerterreputation zusammenzufassen:

- Reputationswerte werden lokal berechnet. Jeder Nutzer sieht einen individuellen Wert.
- Zweistufige Vertraulichkeit durch Pseudonyme und geheime Wertungsbedeutung.
- Verteiltes System mit Verwendung von digitalen Signaturen.
- **Bewerterreputation**: lokale (subjektive) und globale (objektive) Berechnung von Bewerterreputation durch Vergleich von Wertungen und gegenseitiges Bewerten von Nutzern.

3 Bewerterreputation

Diese Arbeit soll das JASON-Reputationssystem um Bewerterreputationen erweitern. Dabei werden zwei Ansätze verfolgt. Zum einen sollen die bestehenden Daten besser interpretiert werden, zum anderen sollen neue Daten erhoben werden, um die Bewerterreputation zu berechnen. Aus beiden Ansätzen wird je eine Art von Bewerterreputationen definiert. Mögliche Vorteile werden angesprochen und die Anforderungen an das neue System aufgelistet.

3.1 Was ist Bewerterreputation?

Im JASON-System handelt ein Bewerter, indem er Bewertungen abgibt. Seine Bewerterreputation sollte davon abhängen, wie gut er das macht. Wenn die Bewertung ein Text ist, wie z. B. bei einer Amazon-Produkt-Rezension, kann man diesen Text bewerten. Das ist bei Amazon in Form einer Ja-Nein-Frage möglich (”War diese Rezension für Sie hilfreich?”). Die Rezensionen werden dann nach Anzahl der Ja-Antworten sortiert. Das entspricht einer Bewertungsbewertung. Bei JASON sind Bewertungen aber keine Texte, sondern Signaturen mit definierten Bedeutungen, vergleichbar mit einer Note. Es ist schwierig, etwas über die Nützlichkeit einer Note zu sagen. Einfacher ist es, etwas über die Ähnlichkeit von Noten zu sagen: Wie ähnlich ist eine Bewertung der durchschnittlichen Bewertung und wie ähnlich haben zwei Nutzer gewertet?

3.1.1 Subjektive Bewerterreputation

Bei JASON gibt es keinen objektiven Reputationswert, sondern nur subjektive Reputationswerte, die sich jeder Nutzer selbst definieren kann. Ein Vergleich zwischen dieser subjektiven Reputation und einer Wertung oder zwischen zwei Wertungen ist in JASON nicht vorgesehen, man könnte dies aber definieren. Dadurch wird es möglich, alle Wertungen eines Pseudonyms mit der eigenen Wertung oder dem eigenen Reputationswert zu vergleichen und so ein Profil über das Wertungsverhalten des Pseudonyms zu erstellen. Für einen Nutzer A mit seiner Menge von Wertungen $R_A = \{r_1, r_2, \ldots, r_n\}$ benötigt man also eine Funktion s_A, die etwas über die Ähnlichkeit der Wertung von A zu einem Inhalt und der Wertung eines anderen Nutzers mit seinem Pseudonym B zum gleichen Inhalt aussagt. Sei $r_i \in R_A$ die Wertung von A und $r_j \in R_B$ die Wertung von B, dann lässt sich s_A folgendermaßen darstellen:

$$s_A(B) : (r_i, r_j) \rightarrow S_A$$

S_A ist die Menge der zulässigen Ähnlichkeitswerte, die zwei Wertungen haben können. Alle Ähnlichkeitswerte $s_i \in S_A$, die A durch Vergleich mit den Wertungen von B erhält, ergeben ein Tupel

$(s_1, \ldots, s_k) \in S_A^*$. Aus diesem Tupel lässt sich ein subjektives Maß für Bewerterreputationen definieren:

Definition 4 (Subjektive Bewerterreputation) *Die subjektive Bewerterreputation $sbrep_A(B)$ aus Sicht von Nutzer A über einen Nutzer B ist ein Maß für die Ähnlichkeit der Wertungen von B im Vergleich zu den Wertungen von A.*

$$sbrep_A(B) : S_A^* \rightarrow S'_A$$

S'_A ist die Menge der möglichen Bewerterreputationen, die sich A frei definieren kann. Je nach Implementierung kann $S'_A = S_A$ sein, muss aber nicht. Wie auch bei der Beschreibung von JASON, werden keine konkreten Berechnungsvorschriften angegeben. Jede Implementierung kann die für ihren Kontext am besten geeigneten Funktionen verwenden. Damit die Ergebnisse für den Nutzer transparent bleiben, sollten folgende Eigenschaften vorhanden sein: Wertet ein Pseudonym B sehr ähnlich zu Nutzer A, so ist $sbrep_A(B)$ groß. Gleichzeitig ist die Wahrscheinlichkeit, dass der Nutzer hinter B zu einem für A unbekannten Inhalt eine ähnliche Meinung wie A hat, falls A den Inhalt kennen würde, hoch. Man könnte sogar versuchen, gezielt Inhalte zu finden, die B gut bewertet hat, da sie für A wahrscheinlich interessant sind. Die subjektive Bewerterreputation wird im Folgenden auch mit SBRep abgekürzt.

Die subjektive Bewerterreputation kann folgenden Vorteil bringen: Das JASON-System sieht Trust-Netzwerke vor, die eine Grundlage geben, wie sehr man einer Wertung vertraut. Abhängig vom Vertrauen gegenüber anderen Nutzern kann man einschätzen, wie sehr man einem unbekannten Inhalt vertraut. Dieses Vertrauen zu einem Nutzer muss aber nicht unbedingt etwas über den Nutzen der Wertung zu einem Inhalt aussagen. Es gibt bei vielen Dingen unterschiedliche Ansichten, ohne dass sich ein Richtig oder Falsch festlegen lässt. Das System könnte bei der Berechnung der subjektiven Reputation Wertungen von Nutzern mit einer hohen subjektiven Bewerterreputation bevorzugen und somit bessere Ergebnisse liefern.

Ein Nachteil beim Ansatz von JASON ist es, dass man nur Wertungen von Nutzern interpretieren kann, zu denen man eine transitive Trust-Kette aufbauen kann, die man also direkt oder indirekt kennt. Normalerweise kennt man die Bewertungsschlüssel von unbekannten Personen nicht, kann also ihre Wertungen nicht erkennen und interpretieren. Es ist aber denkbar, dass ein Nutzer die zu einem seiner Pseudonyme gehörenden Schlüssel veröffentlicht. Dann kann man die Wertungen erkennen, hat aber keinen Trust-Wert zu diesem Nutzer und weiß nicht, ob man der Wertung vertrauen kann. Dafür wäre eine Art objektive Bewerterreputation erforderlich, die etwas über die Qualifikationen des Nutzers aussagen, der hinter dem Pseudonym steht. Die Frage stellt sich allerdings, wie man die objektive Bewerterreputation kennen soll, wenn man den Bewerter nicht kennt. Das JASON-System kommt ohne zentrale Komponente aus, so dass man nirgendwo zentral Informationen ablegen kann, die für alle verfügbar sind. Wie man das Konzept umsetzen kann, wird in den späteren Kapiteln erarbeitet. Zunächst soll untersucht werden, woraus man objektive Bewerterreputation bestimmen kann.

3.1.2 Objektive Bewerterreputation

Es sei A wiederum der Nutzer, der die Bewerterreputation von B wissen möchte. Ein Vergleich von Wertungen führt wieder zur subjektiven Bewerterreputation. Außerdem kennt man vielleicht nicht genug Wertungen von B, um die subjektive Bewerterreputation aussagekräftig zu berechnen. Die subjektiven Bewerterreputationen anderer Nutzer über B sind auch nicht nützlich, da sie auf den Nutzen des jeweiligen Erzeugers gerichtet sind. Eine möglichst objektive Aussage über die Qualifikation von B ist gefragt. Woher kann diese Bewertung über die Qualifikation eines Nutzers kommen? Objektive Aussagen bekäme man durch standardisierte Prüfungen, wie sie z. B. in einigen Studiengängen mit Staatsexamen üblich sind. Die Einführung von solchen Prüfungen ist denkbar, aber nicht praktikabel, da wahrscheinlich niemand freiwillig daran teilnehmen würde.

Objektivität lässt sich aber auch anders annähern. Wenn man genügend subjektive Aussagen nimmt, können sich die Fehler gegenseitig aufheben und sehr gute Ergebnisse entstehen. [Sur05] beschreibt dieses Phänomen als Weisheit der Masse. Demnach kann die Entscheidung einer Masse besser sein als die besten einzelnen Entscheidungen, wenn vier Faktoren vorhanden sind. Ob die Faktoren erfüllt werden, lässt sich nur abschätzen. Bei konkreten Entscheidungen kann aber die Qualität evaluiert werden. Kapitel 5.4 versucht das bei der aus diesem Ansatz gewonnenen Lösung. Die vier Faktoren sind:

- Vielfalt: Jeder soll seine eigenen Informationen und Interpretationen haben.
- Unabhängigkeit: Die Meinungen Einzelner dürfen nicht zu stark voneinander abhängen.
- Dezentralität: Teilnehmer der Entscheidung verfügen über lokales Wissen.
- Zusammenfassung: Die Meinungen vieler können zu einer Gesamtmeinung zusammengefasst werden.

Die ersten drei Punkte treffen wahrscheinlich sehr gut auf Internetgemeinschaften zu. Es werden Leute zusammengebracht aus allen Bereichen und Gegenden, die nur schwach voneinander abhängen. Was fehlt, ist eine Möglichkeit der Zusammenfassung. Im Fall von JASON muss ein Nutzer bereits Einschätzungen über andere Nutzer vornehmen, indem er ihnen einen Trust-Wert gibt. Es wäre ein vertretbarer Mehraufwand, diesen Trust-Wert mehrdimensional zu gestalten. Anstatt allgemein zu sagen, wie sehr man einem anderen vertraut, teilt man das Vertrauen auf Kontexte auf. Man bewertet das Vertrauen zu einem Nutzer in einem bestimmten Kontext, was einer Bewertung seiner Kompetenz auf diesem Gebiet gleichkommt. Beispielsweise vertraut man einem befreundeten Arzt wahrscheinlich mehr bei Gesundheitsfragen als bei Computerproblemen. Problematisch könnte es werden, wenn sich Teilnehmer zu gut kennen und somit ihre Unabhängigkeit verlieren. Die Wertung, die man anderen gibt, muss sichtbar sein. Das würde aber auch einschließen, seinen Freunden negative Bewertungen zu geben. Üblicherweise will man seinen Freunden nicht schaden und verliert somit seine Unabhängigkeit. Dieses Problem muss beim Systementwurf in Kapitel 4 beachtet werden.

Die Aufteilung der Kompetenz auf Kontexte bzw. Fachgebiete ist schwierig. Es gibt Universalklassifikationen, die man verwenden könnte ([klaa] und [klab]). Um das System aber möglichst einfach zu halten und schnell zu einem funktionierenden Prototypen zu gelangen, gehen wir im Folgenden von einem festgelegten Kontext aus. Die Erweiterung auf beliebige dynamische Kompetenzgebiete bleibt für Nachfolgearbeiten offen.

Zur Definition wird eine Wertungsmenge R, deren Elemente etwas über die Kompetenz aussagen, benötigt. Nutzer A definiert sich dann zu Nutzer B eine Funktion $q_A(B)$, welche die Kompetenz von B auf eine Wertung $r_i \in R'$ abbildet. Die Menge R' kann die gleiche wie R sein, aber auch eine andere Wertungsmenge, deren Elemente etwas über die zusammengefasste Qualifikation von B aussagen. Eine zweite Art von Bewerterreputation, die wir objektive Bewerterreputation nennen, lässt sich wie folgt definieren:

Definition 5 (Objektive Bewerterreputation) *Die objektive Bewerterreputation eines Nutzers B setzt sich aus allen Bewertungen $r \in R$ zusammen, die er von anderen Nutzern des Systems bekommt.*

$$obrep(B) : R^* \rightarrow R'$$

Die objektive Bewerterreputation wird im Folgenden teilweise mit OBRep abgekürzt. Mit der OBRep ist die letzte der vier Bedingungen für eine intelligente Gruppenentscheidung gegeben: Eine Funktion der Zusammenfassung, die den Namen objektive Bewerterreputation bekommt. Die objektive Bewerterreputation eignet sich gut bei wissenschaftlichen Fachbereichen, die sich gut klassifizieren lassen. Die subjektive Bewerterreputation eignet sich gut für schwer klassifizierbare Inhalte.

Zwei Beispiele zum besseren Veständnis: Eine Gruppe von Kinofreunden bewertet jeden Film, den sie sehen. Da sie sich gegenseitig kennen, kennen sie viele der Wertungen der anderen und können sich subjektive Reputationswerte voneinander ausrechnen. Obwohl vielleicht alle viel von Filmen verstehen und somit objektiv gesehen eine hohe Qualifikation auf diesem Gebiet haben, haben sie alle einen unterschiedlichen Geschmack. Die subjektive Bewerterreputation kann einem dieser Kinofreunde helfen zu entscheiden, ob er sich einen Film, den die anderen bereits gesehen haben, anschauen sollte. Dazu könnte er sich die Bewertungen des Filmes von den Freunden mit der höchsten subjektiven Bewerterreputation ansehen.

Für das zweite Beispiel nehmen wir an, einer dieser Freunde ist Student und sichtet Quellen für eine Studienarbeit, die von einem ihm unbekannten Professor einer anderen Universität bewertet wurden. Dieser Professor veröffentlicht seine Bewertungsschlüssel auf seiner Homepage, seine Wertungen sind also erkennbar. Der besagte Student kann nicht einschätzen, ob der Professor auf dem Gebiet der Quelle eine Koryphäe ist oder seine Stärken doch eher in einem anderen Fachbereich hat. Es gibt aber sehr viele Studenten an der anderen Universität, die eine Meinung zur fachlichen Qualifikation des Professors haben und eine Grundlage für die objektive Bewertereputation sein können. Mit dieser objektiven Bewerterreputation kann die Bewertung der Quelle besser eingeschätzt werden.

Tabelle 3.1 zeigt eine Übersicht der beiden Bewerterreputationen. B ist wie bisher immer das Pseudonym des Nutzers, auf den sich die Bewerterreputation bezieht. A ist der Nutzer, der die Reputation wissen will.

3.2 Anforderungen zur Umsetzung

Wie bereits dargestellt, kommen durch die Einführung von Bewerterreputation folgende Anforderungen an das JASON-System zu den bisherigen hinzu:

	objektiv	**subjektiv**
Entstehung	aus direkten Bewertungen über B durch Personen, die mit B zu tun hatten (nutzt Wissen im sozialen Netz)	aus dem Verhalten (Werten) von B im Vergleich zu A
Einsatz	gut für wissenschaftliche Fachbereiche	gut für schwer klassifizierbare Inhalte oder wenn Geschmack eine Rolle spielt
Bedingung	nur eine Wertung von B muss bekannt sein (gut für Personen außerhalb des sozialen Netzes)	viele bekannte Wertungen von B notwendig (für Personen innerhalb des sozialen Netzes)

Tabelle 3.1: Übersicht zu objektiver und subjektiver Bewerterreputation von B aus Sicht von A.

- Eine Ähnlichkeitsfunktion s_A zwischen zwei Wertungen.
- Die subjektive Bewerterreputationsfunktion $sbrep(B)$.
- Eine Metrik für das Bewerten der Kompetenz anderer Nutzer (die Mengen R und R').
- Die objektive Bewerterreputationsfunktion $obrep(B)$.

Folgende Anforderungen aus dem JASON-System müssen angepasst werden:

- Die Reputationsfunktion eines Inhaltes muss die Bewerterreputation mit einbeziehen.
- Die Datenhaltung muss um Bewerterreputation erweitert werden.

Die Ähnlichkeitsfunktion und die subjektive Bewerterreputation lassen sich ohne große Veränderung des JASON-Systems hinzufügen. Es müssen zwei zusätzlichen Funktionen definiert und die neue Information bei der Berechnung der Reputationswerte verwendet werden. Die subjektiven Bewerterreputationswerte sollte jeder lokal speichern, damit sie nicht jedes Mal neu berechnet werden müssen. Die objektive Bewerterreputation erfordert deutlich mehr Veränderungen und lässt sehr viele Designoptionen zu. Das nächste Unterkapitel diskutiert deshalb Vereinfachungen, damit in Kapitel 4 ein funktionierendes System entworfen werden kann.

Neben den funktionalen Anforderungen sollten beim Entwurf auch die üblichen Sicherheitsanforderungen beachtet werden:

Verfügbarkeit Verfügbare Bewerterreputation ermöglicht bessere Reputationswerte. Das System bleibt zwar auch ohne Bewerterreputation funktionsfähig, indirekt bietet sich hier aber die Möglichkeit, den Reputationswert zu manipulieren.

Integrität Die Bewerterreputation soll vor Manipulation sowohl im Datenbestand als auch in der Weitergabe geschützt sein. Bewerterreputation kann sehr wichtig werden und muss deshalb besonders gut vor Manipulation geschützt werden (siehe Beispiel aus Kapitel 1, Zusammenhang zwischen Weinpreisen und Reputation).

Vollständigkeit Es soll keine Manipulation der Bewerterreputation durch Unterschlagung von Infor-

mation möglich sein.

Pseudonymität Nutzer sollen unter Pseudonymen am System teilnehmen können, ohne dass man ihnen ihr Pseudonym zuordnen kann. Da die Bewerterreputation für Pseudonyme berechnet wird, bleibt dieser Schutz aus dem JASON-System bestehen.

Anonymität Das Berechnen von Bewerterreputation soll anonym möglich sein. Besonders bei der objektiven Bewerterreputation ist zu prüfen, ob derjenige, der die Kompetenz eines anderen Systemteilnehmers bewertet, anonym bleiben kann.

Vertraulichkeit Die Menge an Informationen, die über einen Nutzer durch die Bewerterreputation preis gegeben werden, soll begrenzt bleiben. Während andere Nutzer einen Vorteil aus möglichst vielen Information ziehen können, bedeutet dies zugleich einen Verlust von Vertraulichkeit für den einzelnen. Hier muss ein Kompromiss gefunden werden.

3.3 Festlegungen zur Umsetzung Bewerterreputation

Eine der Anforderungen ist eine Metrik für das Bewerten der Kompetenz anderer Nutzer. Dabei gibt es einige Schwierigkeiten. Wenn sich Nutzer im sozialen Netz gegenseitig bewerten sollen, gibt es viele Probleme. Wie genau soll eine Wertung aussehen? Welches Bewertungsskala soll zugrunde liegen? Will man seine Freunde nicht immer positiv und Personen, die man nicht mag, negativ bewerten? Die folgenden Abschnitte diskutieren einige Möglichkeiten und führen letztendlich den Begriff des *Sterns* für eine möglichst einfache Umsetzung ein.

Bisher wurde nicht festgelegt, wie eine Wertung aussieht. Jeder Nutzer kann seine eigene Bewertungsmetrik verwenden. Diese kann nur er richtig interpretieren. Bei der objektiven Bewerterreputation benötigt man Wertungen, die sich zusammenfassen lassen und die jeder interpretieren kann. Deshalb erscheint es sinnvoll, festzulegen, wie die Bewertung der Kompetenz eines anderen Nutzers aussieht. Im Folgenden werden drei Möglichkeiten diskutiert und eine davon ausgewählt:

Binär Die einfachste Form der Bewertung ist eine binäre Aussage. Eins, bzw. Ja, bedeutet, dass man den anderen Nutzer für kompetent hält. Null oder nein kann verschiedene Dinge bedeuten. Entweder man hält den Nutzer nicht für kompetent, oder man hat dazu keine Meinung, oder noch keine Meinung abgegeben. Dieses System lässt keine Unterscheidung zwischen keiner Meinung und einer negativen zu. Das hat Vorteile, aber Verlust an Aussagekraft zur Folge. Durch binäre Wertungen ist kein gezieltes Schlechtwerten möglich (Bad Mouthing[1]). Außerdem schwächt es den sozialen Druck ab, seine Bekannten nicht negativ zu bewerten. Ein dritter Vorteil ist, dass der Sternempfänger keinen Grund hat, Sterne zu unterschlagen. Das motiviert die Vollständigkeit der Sternliste. Der Nachteil ist, dass es keine negativen Wertungen gibt. Eventuell sind die negativen Wertungen aber gar nicht die, nach denen man sucht. Ob man eine Inhaltsbewertung nicht interpretieren kann, weil man den Bewerter nicht kennt, oder ob man weiß, dass man die Bewertung aufgrund einer schlechten objektiven Bewerterreputation nicht verwenden sollte, führt beides zum gleichen Ergebnis: Der Inhaltsbewertung sollte nicht vertraut werden.

[1] Bad Mouthing ist ein Angriff auf Reputationssysteme, bei dem man versucht, die Reputation eines anderen zu schädigen.

Eindimensional Ein etwas kompliziertes Bewertungssystem wäre ein eindimensionaler Wert wie z. B. Schulnoten. Fasst jedes Land hat sein eigenes Notensystem in der Schule. Der Umgang mit solchen Bewertungen ist also bei den meisten Menschen bereits trainiert. Problematisch ist die Umrechnung zwischen verschiedenen Systemen. Auch werden die Systeme oft unterschiedlich interpretiert. In verschiedenen Schulen sind unter Umständen sehr unterschiedliche Leistungen notwendig, um die bestmögliche Schulnote zu erhalten. Was an Genauigkeit gegenüber der binären Form gewonnen wird, kann so wieder verloren gehen. Ein weiteres Problem sind die negativen Bewertungen, die das System einführt. Der negativ Bewertete hat ein Interesse daran, diese negative Wertung geheim zu halten. Die Wertung nicht interpretierbar zu machen, wie es bei der Inhaltsbewertung in JASON funktioniert, widerspricht dem Anwendungsfall der objektiven Bewerterreputation. Die OBRep soll eben für möglichst viele interpretierbar sein. Die Frage, wohin mit negativen Wertungen in einem System ohne zentrale Instanz, bleibt offen.

Mehrdimensional Eine mehrdimensionale oder textuelle Form der gegenseitigen Bewertung ist denkbar, aber für unseren Fall ungünstig. Die Grundidee der objektiven Bewerterreputation ist es, viele Meinungen zusammenzufassen. Je einfacher die Meinungen dargestellt werden, desto einfacher ist die Zusammenfassung. Beispiele für eine mehrdimensionale Bewertung sind Zeugnisse mit vielen Noten oder Bewertungsschreiben. Geeignet scheint diese Form bei wenigen Bewertern (die selbst eine hohe Qualifikation haben müssen), die als Quelle für die objektive Bewerterreputation dienen. Für diese Arbeit trifft das nicht zu, mehrdimensionale Wertungen werden deshalb nicht weiter betrachtet.

Diese Arbeit entscheidet sich für **binäre positive Wertungen** aus folgenden Gründen:

Vorteile binärer Wertungen:

- lassen sich einfach umsetzen
- kein Bad Mouthing möglich
- weniger sozialer Druck gegenüber Freunden
- international verständlich (kein landestypisches Notensystem)
- fehlen von negativen Wertungen eventuell nicht störend, da positive Wertungen interessanter
- Ungenauigkeit einzelner Wertungen mindert sich durch Zusammenfassung vieler Wertungen

Umsetzen lässt sich dieses System sehr einfach, indem sich Nutzer gegenseitig signierte Texte schicken (ähnlich PGP, bei dem sich Nutzer gegenseitig die Schlüssel signieren). Ein Nutzer A schickt z. B. einem Nutzer B einen Text "Ich, Nutzer A, denke, dass Nutzer B kompetent ist.". Diesen Text signiert A mit seinem privaten Schlüssel. Dadurch kann B jedem beweisen, dass ihn A wirklich für kompetent hält. Im Folgenden führen wir für diesen Vorgang einen Begriff ein. A vergibt einen *Stern* an B. Sterne gibt es z. B. bei der Bewertung von Hotels, bei der Bewertung von YouTube-Videos und Köche müssen hart arbeiten, um einen Stern zu bekommen. Der Begriff Stern lässt sich wie folgt definieren (zu beachten ist, dass der Kompetenzbereich wiederum als festgelegt angenommen wird):

Definition 6 (Stern) *Ein* Stern *wird von einem Nutzer A an einen Nutzer B vergeben. Er bedeutet, dass A den Nutzer B für kompetent hält. Der Stern besteht aus einem Text, kombiniert mit den Identitätsdaten*

von B, der von A signiert wird.

Anders als z. B. bei Hotel-Sternen wird hier nur ein Stern vergeben. Entweder, man hält B für kompetent, oder nicht. Je nach Implementierung könnte man den Text nutzen, um die Bedeutung des Sterns genauer zu beschreiben. Die einfachste Möglichkeit ist es aber, wenn der Text nur aus dem öffentlichen Schlüssel von B und eventuell noch dem Pseudonym-Namen von B besteht. In dem Fall ist der Stern einfach eine Signatur der Identitätsdaten von B, wobei die Signatur eine festgelegte Bedeutung hat.

3.4 Anwendungsfälle

Die beiden Hauptanwendungsfälle aus JASON sind die Berechnung der Reputation eines Inhaltes und die Abgabe einer Wertung für einen Inhalt. Bei der Abgabe einer Wertung gibt es kaum Unterschiede zum bisherigen JASON. Interessant ist jedoch die Berechnung der Reputation. Hier müssen die beiden Bewerterreputationen mit einbezogen werden. Je nachdem, ob eine subjektive oder objektive Bewerterreputation zur Verfügung steht, sollte dies bei der Berechnung beachtet werden. Nicht dargestellt ist der Anwendungsfall ”Autorenschaft angeben”, da sich dieser nicht verändert im Vergleich zum originalen JASON-System.

Abbildung 3.1 zeigt ein Anwendungsfall-Diagramm. Die Akteure stehen für jeweils ein Pseudonym, hinter dem ein Nutzer des Systems steht. Der Inhalt ist eigentlich kein Akteur, wird aber zum besseren Verständnis mit dargestellt. Im Folgenden wird jeder Anwendungsfall kurz beschrieben.

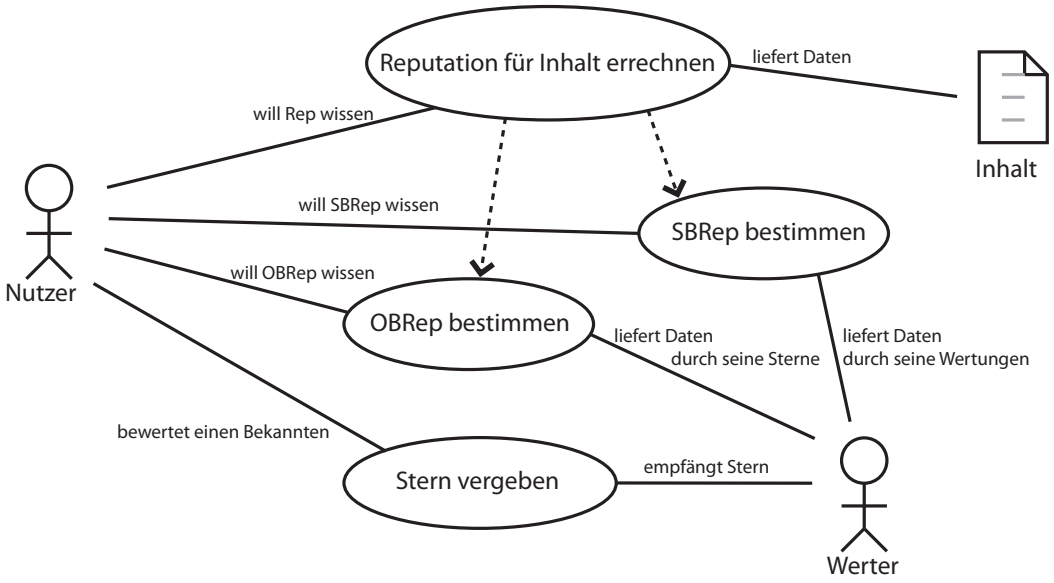

Abbildung 3.1: Anwendungsfälle.

Reputation für einen Inhalt errechnen: Der *Nutzer* will seinen individuellen Reputationswert für einen Inhalt wissen. Dazu werden alle Wertungen, die am Inhalt hängen, vom System erfragt. Diese Wertungen wurden von anderen Systemnutzern erstellt. Im originalen JASON wird für die Gewichtung einer Wer-

tung der Trust-Wert zum Erzeuger der Wertung verwendet. Der Trust-Wert sagt aus, wie sehr man der Person hinter dem Pseudonym vertraut. In dieser Arbeit kommen zu dem Vertrauen zu einer Person noch deren subjektive und objektive Bewerterreputation hinzu. Falls sich die Bewerterreputationen berechnen lassen (was nicht immer der Fall sein muss), sollten sie für die Berechnung der Reputation des Inhaltes mit verwendet werden.

OBRep (objektive Bewerterrep.) bestimmen: Will der *Nutzer* die objektive Bewerterreputation von einem *Werter* wissen, muss er von diesem seine Sterne erfragen. Wo diese gespeichert werden und wie *Nutzer* genau an die Sterne kommt, wird in Kapitel 4 beschrieben.

SBRep (subjektive Bewerterrep.) besitmmen: Bei der Berechnung der subjektiven Bewerterreputation (SBRep) sind wieder zwei Akteure beteiligt. Der *Nutzer* will die SBRep wissen, während der *Werter* die Daten dafür liefert. Das macht er indirekt, indem er selbst Wertungen beim System abgibt, die der *Nutzer* verwendet, um sich den SBRep-Wert zu errechnen.

Stern vergeben: Der letzte Anwendungsfall betrifft die Vergabe von Sternen. Dabei erzeugt der *Nutzer* einen Stern für den *Werter* und gibt ihn beim System ab, welches den Stern an diesen *Werter* weitergibt. Der *Werter* kann nun diesen Stern für andere sichtbar machen und liefert so die Daten für die Berechnung der OBRep.

Es stellt sich die Frage, wie man einen Stern interpretiert. Wenn man den Erzeuger des Sterns direkt oder indirekt kennt, kann man ihm einen Vertrauenswert zuordnen. Man könnte auch einen minimalen Vertrauenswert für unbekannte Nutzer verwenden[2]. Eine weitere Frage ist, wer Sterne vergeben darf. Zunächst ist hier keine Einschränkung vorgesehen, da sich schwer ein Kriterium finden lässt, um dies zu entscheiden. Eventuell könnte man aber die Sterne unterschiedlich wichten, z. B. abhängig von der OBRep des Stern-Erzeugers oder ihrem Alter. Diese Entscheidungen sind Teil der Funktion $obrep(B)$ und hängen von der Implementierung des Systems ab. Im Kapitel 4.3 wird der Ablauf der Anwendungsfälle genau beschrieben.

[2]Dann sollte das System gut gegen Sybil-Angriffe geschützt sein. Das bedeutet, dass man nicht einfach beliebig viele Identitäten erstellen kann, um viele Sterne zu vergeben.

4 Integration von Bewerterreputation in JASON

Die bestehende Implementierung des JASON-Systems hat gezeigt, dass das Konzept funktioniert. Allerdings ist der entstandene Prototyp schwer erweiterbar und fehleranfällig. Außerdem werden einige Teile des Konzeptes (z. B. austauschbare vertrauliche Metrik) nicht umgesetzt. Deshalb wird in diesem Kapitel, mit dem Wissen des ersten Prototypen, ein Redesign durchgeführt. Dabei werden die im Analyse-Kapitel 3 erarbeiteten neuen Anforderungen und Definitionen mit einbezogen. Die objektive Bewerterreputation wird so einfach wie möglich umgesetzt, um schnell zu einem funktionierenden Prototypen zu gelangen. Eine möglichst einfache erste Umsetzung soll helfen, praktische Erfahrungen für Nachfolgearbeiten zu sammeln.

4.1 Datenhaltung

Bei einer verteilten WEB-Anwendung, die besonderen Wert auf mehrseitige Sicherheit legt, ist es besonders wichtig zu klären, welche Daten wo gespeichert werden und ob sie vertraulich bleiben müssen oder nicht. Das JASON-System benötigt folgende Daten:

- Die Pseudonyme und ihre Autorenschlüssel.
- Die Wertungsschlüssel.
- Die Rep-Dateien[1] zu einem Pseudonym.
- Die RDF-Dateien in denen die Wertungen von Inhalten aufgelistet werden.
- Die Wertungen (Signaturen).
- Die Bedeutung von Wertungen und Metrik zum Berechnen von Reputation.
- Die Bekannten (Soziales Netz).
- Die Einstellungen zum Arbeiten, wie z. B. FTP-Zugangsdaten, Schlüssellängen, usw.

In dieser Arbeit kommen Daten für die objektive und die subjektive Bewerterreputation hinzu. Für die objektive Bewerterreputation sind das die Sterne, die in Kapitel 3.3 eingeführt wurden. Bei der subjektiven Bewerterreputation sind das die Vergleiche zwischen Wertungen (siehe 3.1). Theoretisch könnte man diese Vergleiche bei jeder Berechnung der subjektiven Bewerterreputation neu vornehmen. Die Wertungen sind aber über das gesamte Internet verteilt und es gibt kein zentrales Verzeichnis, um sie zu finden. Die Berechnung würde sehr lange dauern. Deshalb sollten die Vergleiche gespeichert werden, um schnell die subjektive Bewerterreputation berechnen zu können. Folgende Daten kommen demnach neu hinzu:

- Sterne für die objektive Bewerterreputation

[1] "Rep" steht für Reputation. Der Name wurde in der Vorgängerarbeit [Mei08] eingeführt.

- Vergleiche von Wertungen für die subjektive Bewerterreputation

Die Daten lassen sich in ”vertraulich” und ”nicht vertraulich” einteilen. Diese Einteilung ist in Tabelle 4.1 dargestellt. Zu beachten ist, dass die Bedeutungen und die Metrik an Bekannte weitergegeben werden können, damit diese die Wertungen interpretieren können. Vertraulichkeit ist deshalb nur noch schwach gegeben und hängt vom Verhalten der Bekannten ab.

nicht vertraulich	**vertraulich**
Pseudonymname	Zuordnung Pseudonym ↔ Nutzer
öffentlicher Autorenschlüssel	privater Autorenschlüssel
öffentliche Wertungsschlüssel	private Wertungsschlüssel
fremde Wertungsschlüssel mit gleicher Bedeutung	Bedeutung eigener Wertungen und Metrik zur Berechnung der Reputation
RDF-Dateien	Programmeinstellungen
Sterne der OBRep	Vergleiche der SBRep
Wertungen	

Tabelle 4.1: Einteilung der Daten in ”vertraulich” und ”nicht vertraulich”.

Bevor geklärt wird, wo die neuen Daten gespeichert werden, sollen die alten näher betrachtet werden. Wo werden die alten Daten gespeichert? Was bedeuten sie und welche Teile sind vertraulich (dürfen nur dem Besitzer bekannt sein)? Ein Pseudonym besteht aus nichts weiterem als einem Namen. Dieser muss eindeutig zu einem Autorenschlüssel gehören und kann beliebig viele Wertungsschlüssel besitzen. Alle Schlüssel bestehen aus zwei Dateien. Eine enthält den öffentlichen Schlüssel und eine den privaten. Der Name eines Pseudonyms und sein zugehöriger öffentlicher Autorenschlüssel sollten für alle Systemnutzer sichtbar sein. JASON sieht dafür ein Identitäts-Management-System vor. Die öffentlichen Wertungsschlüssel und der öffentliche Autorenschlüssel eines Pseudonyms sind in einer Datenstruktur verlinkt (Rep-Datei). Außerdem enthält diese Datei Links zu Wertungsschlüsseln anderer Pseudonyme mit gleicher Bedeutung (nach Meinung des Besitzers der Rep-Datei). Die Bedeutung der Wertungsschlüssel ist vertraulich und nur den Freunden eines Nutzers bekannt. Mit den privaten Wertungsschlüsseln kann jeder Nutzer hinter einem Pseudonym Signaturen erzeugen und ins WEB hochladen. Die Links zu diesen Signaturen werden in RDF-Dateien gesammelt. Eine RDF-Datei enthält in diesem Fall einen oder mehrere Links zu einem Inhalt im Netz und den dazugehörenden Signaturen. Jeder Nutzer legt sich eine Datenbank in seinem Vertrauensbereich an. Diese enthält alle seine privaten Schlüssel, seine Bekannten und sonstige Einstellungen, die das System benötigt (Passwort, FTP-Zugangsdaten, Metriken, usw.).

Abbildung 4.1 soll die Zusammenhänge der Daten veranschaulichen. In der Abbildung werden drei Rep-Dateien dargestellt. Die Schlüssel symbolisieren die öffentlichen Wertungsschlüssel. Die Pfeile zwischen den Schlüsseln stellen die Verlinkungen der Wertungen dar. Jede Rep-Datei gehört zu einem Pseudonym. Zu jedem Pseudonym gehört ein Vertrauensbereich. Im Bild ist nur ein Pseudonym mit dem Namen *Bob* dargestellt. Ihm gehören drei Wertungsschlüsselpaare. Der Vertrauensbereich enthält außerdem die Einstellungen des Systems, die Bekanntenliste (Soziales Netz) mit den Trust-Werten und die Bedeutungen

der Wertungen samt Metrik. Die RDF-Datei verweist auf einen Inhalt und zugehörige Signaturen (Wertungen, dargestellt durch den Stift). Eine Signatur wiederum verweist auf den Schlüssel, mit dem sie geprüft werden kann. In der Abbildung sind die hinzukommenden Daten der Bewerterreputation grau eingezeichnet. Die Sterne der objektiven Bewerterreputation werden in den Rep-Dateien gespeichert (es gibt andere Möglichkeiten, die weiter unten diskutiert werden) und die Vergleiche für die subjektive Bewerterreputation innerhalb des Vertrauensbereiches.

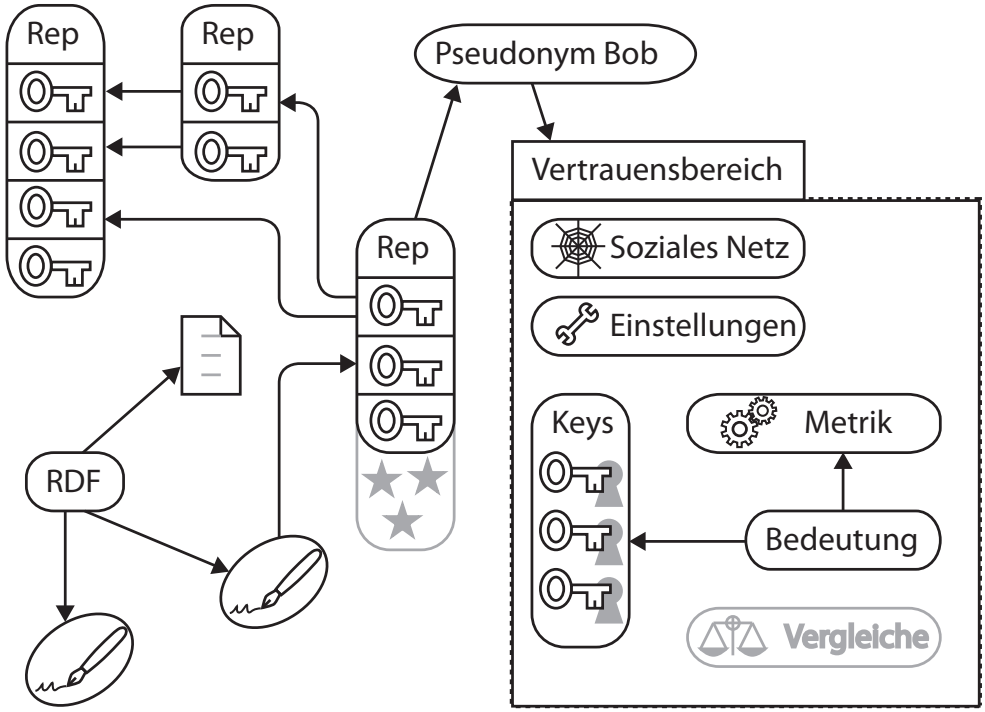

Abbildung 4.1: Daten im JASON-System und ihre Verlinkungen.

Es gibt zwei Möglichkeiten, die subjektive Bewerterreputation zu berechnen. Entweder, man vergleicht die eigene Wertung zu einem Inhalt mit allen fremden Wertungen, die man interpretieren kann, oder man vergleicht den Reputationswert, den man sich zu einem Inhalt errechnet, mit den fremden Wertungen. Letzteres ist immer dann sinnvoll, wenn man den Inhalt nicht bewerten kann oder möchte. Ansonsten sollte man eigene und fremde Wertungen vergleichen, da der berechnete Reputationswert nur eine Schätzung der eigenen Wertung ist. Der eigentliche Vergleich ist Teil der Bewertungsmetrik, die sich jeder Nutzer selbst definiert. Sowohl die Metrik als auch die Eingangsdaten (Wertungen samt Bedeutung) sind vertraulich. Demnach sollten auch die Ergebnisse der Vergleiche vertraulich sein und mit in der vertraulichen Datenbank eines Nutzers gespeichert werden. Theoretisch müsste man die Vergleiche gar nicht speichern, da man sie immer wieder neu berechnen kann. Es handelt sich jedoch um Zugriffe auf Ressourcen im Internet, die offline oder sehr langsam sein können. Deshalb sollten die Vergleichsdaten in der vertraulichen Datenbank gespeichert werden.

Bei der objektiven Bewerterreputation sind die Daten die Sterne, die sich Nutzer gegenseitig geben. Es gibt mehrere Möglichkeiten, wo man diese speichert, da sie nicht vertraulich sind. Es ist sogar gerade wünschenswert, dass möglichst viele Nutzer Zugriff auf sie haben. Weiterhin muss geklärt werden, wie

die Sterne vom Erzeuger zum Empfänger gelangen. Tabelle 4.2 gibt einen Überblick zu den möglichen Speicherorten und deren Vor- und Nachteilen.

Ort	**Vorteile**	**Nachteile**
Inhalt	einfach, verfügbar	nicht aktuell, skaliert schlecht
Nutzer (Sternempfänger)	Aktualität, Vertraulichkeit	Verfügbarkeit
Zentral	negative Wertungen möglich	Sicherheitsanforderungen, widerspricht Systemgrundlagen

Tabelle 4.2: Vor- und Nachteile der verschiedenen Speicherorte der OBRep.

Es gibt drei mögliche Orte, die Sterne zu speichern. Beim Inhalt (in der RDF-Datei des Inhaltes), beim jeweiligen Empfänger eines Sterns (beim Nutzer) oder zentral. Ein zentraler Speicherort wiederspricht den Grundlagen von JASON, würde aber negative Wertungen vereinfachen. Diese Möglichkeit kommt hier nicht in Frage und muss bei anderen Reputationssystemen untersucht werden. Theoretisch sind auch mehrere der drei Möglichkeiten zugleich denkbar. In dieser Arbeit wird jedoch zunächst eine Möglichkeit ausgewählt und umgesetzt. Verteilte Lösungen werden hier nicht untersucht, da zuerst eine einfache Lösung umgesetzt werden soll, um Erfahrungen zu sammeln.

Speichern beim Inhalt bedeutet, man speichert die Sterne zusammen mit den Wertungen (Signaturen) ab. Das hat den Vorteil, dass sie immer dann verfügbar sind, wenn auch die Wertungen verfügbar sind (und nur dann sind sie auch interessant). Allerdings stellt sich die Frage, wie man Aktualität gewährleistet. Jeder Nutzer müsste bei all seinen bisherigen Wertungen neue Sterne hinzufügen (wozu man eine Update-Funktion benötigt) oder man akzeptiert, dass die Sterne nur eine Sicht im Moment der Wertungserzeugung darstellen. Ein weiteres Problem ist die Skalierbarkeit. Ein Nutzer kann Tausende Sterne besitzen. Diese jedes mal beim Werten alle mit an die Wertung zu hängen, ist nicht praktikabel. Man müsste also einige auswählen oder mehrere zu einer Art Zertifikat zusammenfassen.

Die zweite Möglichkeit ist, die Sterne mit in der Rep-Datei des Nutzers zu speichern. Das hat den Vorteil, dass es nur eine Liste gibt, die immer aktuell ist. Außerdem sollte es kein Problem sein, sehr viele Sterne zu verwalten. Selbst mehrere Tausend wären weniger als 1MB Daten[2], was bei heutigen Internet-Anbindungen kein Problem ist. Der Halter der Sterne hat die Kontrolle über seine OBRep und kann selbst entscheiden, ob er sie verfügbar hält. Bei der ersten Lösung hängt es hingegen von der Implementierung ab, ob der Nutzer die Daten z. B. löschen kann, wenn er nicht mehr will, dass sie öffentlich verfügbar sind. Der einzige Nachteil dieser Lösung ist die Verfügbarkeit. Es ist denkbar, dass eine Wertung verfügbar ist, nicht aber die Rep-Datei mit den Sternen des Erzeugers der Wertung. Trotzdem scheint die Speicherung beim Nutzer der beste Weg und wird in dieser Arbeit umgesetzt.

Eine letzte Frage, die geklärt werden muss, betrifft den Datenfluss. Wie kommen die erzeugten Sterne zu ihrem Empfänger? Bisher legt JASON nicht fest, wie Nutzer untereinander kommunizieren, setzt aber voraus, dass sie kommunizieren, um die Bedeutungen ihrer Wertungen auszutauschen. Auf die gleiche Art könnten auch die Sterne ausgetauscht werden. Wie dieser Austausch stattfindet, hängt von

[2]Bei der aktuellen Implementierung benötigt ein Stern weniger als 300Byte in einer XML-Datei (könnte also noch komprimiert werden).

der Implementierung des Systems ab. Denkbar sind z. B. ein direktes Hochladen des Sterns auf einen Webserver des Empfängers oder ein asynchrones Verschicken per E-Mail.

4.2 Systemaufbau

Im bisherigen Text wurden fasst alle Teile des Systems, die benötigt werden, indirekt erwähnt. Konkret sind das folgende Komponenten:

- eine Metrik zum Berechnen von Reputation, Vergleichen und Interpretieren von Wertungen
- ein Komponente zum Laden und Speichern aller Daten (aus dem WEB und aus der vertraulichen Datenbank)
- ein Systemkern, der alle Berechnungen vornimmt, Signaturen erstellt, Freunde kennt
- zu diesem Kern eine GUI, mit der er sich kontrollieren und verwalten lässt
- ein FrontEnd, das die Reputation für einen Inhalt anzeigt und die Daten sammelt, um diese zu berechnen

Diese Komponenten lassen sich in drei Pakete aufteilen, die aufeinander aufbauen. Grundlage ist das Paket *Data*, welches das Laden und Speichern aller Resourcen aus dem WEB und aus dem Vertrauensbereich auf der Festplatte des Nutzers erledigt. Das Paket *Core* nutzt diese Daten und kennt den aktuellen Nutzer, seine privaten Schlüssel und seine Bekannten. Angezeigt werden die errechneten Reputationswerte durch ein Paket *FrontEnd*. Abbildung 4.2 zeigt ein Diagramm zu dieser Architektur. Die drei Pakete werden im Folgenden genauer erklärt.

Data Enthält einen Manager für alle Resourcen. Dieser Manager lädt alle Daten zu den Pseudonymen und ihren Wertungen aus dem WEB oder aus der vertraulichen Datenbank des Nutzers auf seiner Festplatte (kann auch ein USB-Stick sein). Des Weiteren lädt er die Metrik, welche die Bedeutungen der Wertungen kennt und diese verarbeiten kann.

Core Der Kern des Systems besteht aus einem Server, welcher per Messages mit dem FrontEnd kommuniziert. Er läuft auf dem gleichen Rechner wie das Paket *Data* und ist Teil des Vertrauensbereiches (arbeitet mit vertraulichen Daten). Der Server berechnet auf Anfrage den Reputationswert zu einem Inhalt. Dazu muss sich der aktuelle Nutzer per Passwort einloggen. Dieses Passwort gibt der Server an den Ressourcen-Manager aus dem *Data*-Paket weiter, damit die vertraulichen Daten von der Festplatte geladen werden können. Um Wertungsschlüssel zu erstellen, Bekannte zu verwalten und die Wertungen einer Bedeutung in der Metrik zuzuordnen, hat der Server eine GUI, die ihn kontrolliert. Dort werden alle administrativen Arbeiten vorgenommen.

FrontEnd Das *FrontEnd* des Systems befindet sich in einem eigenen Paket. Dieses ist nicht mehr Teil des Vertrauensbereiches und kann auf einem anderen Rechner laufen als die bisherigen Pakete. Das *FrontEnd* kommuniziert per Messages mit dem Server. Es sammelt die Daten, die zur Berechnung des Reputationswertes eines Inhaltes benötigt werden (die Wertungen aus den RDF-Dateien). Danach fragt es ab, welche dieser Wertungen interpretiert werden können, und fordert den berechneten Reputationswert an. Dieser Wert wird dann beim Inhalt angezeigt. In der bisherigen Im-

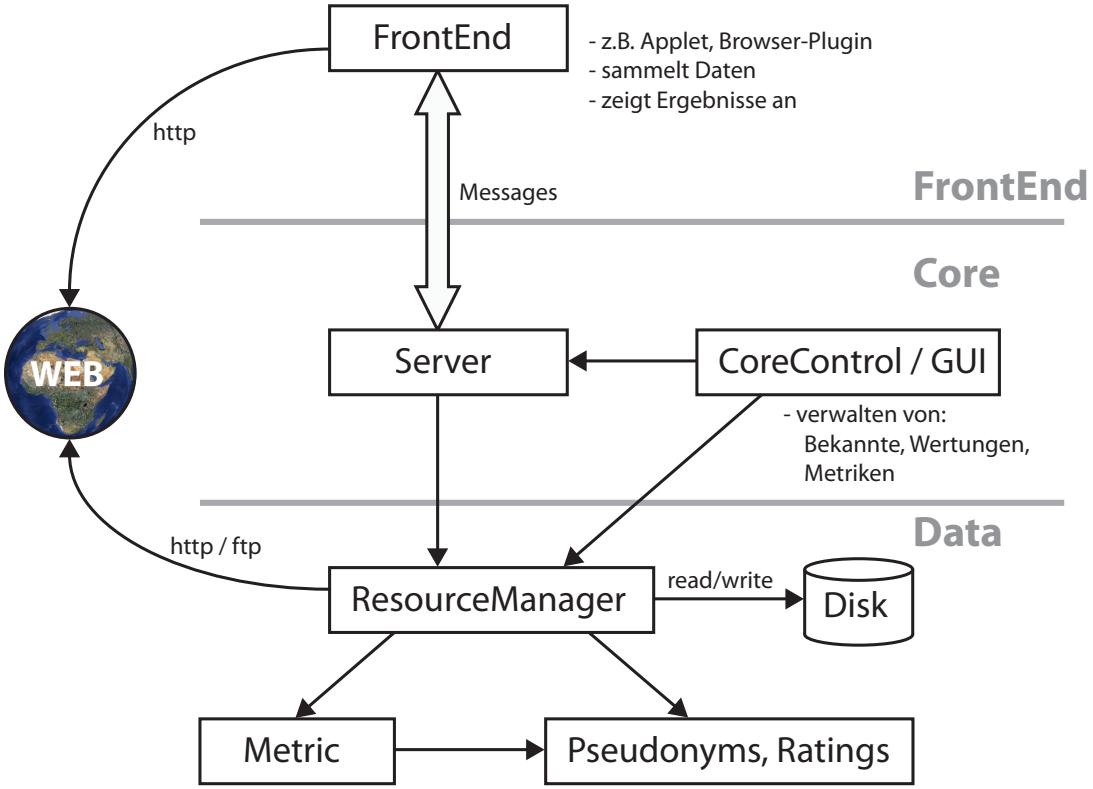

Abbildung 4.2: Aufbau des Systems.

plementierung war das *FrontEnd* ein Applet. Denkbar sind auch ein Browser-Plugin oder andere Rich-Internet-Application-Technologien.

Nachdem die grobe Struktur erläutert wurde, sollen nun die wichtigsten Teile detailliert als UML dargestellt werden. Dabei sind bereits einige Attribute und Methoden eingezeichnet, aber auch viele Details aus Gründen der Übersichtlichkeit ausgelassen. Die erste Abbildung 4.3 zeigt die Datenklassen. Diese Struktur wird vom *ResourceManager* erzeugt und geladen, der hier nicht eingezeichnet ist. Die meisten Daten werden in HashMaps gespeichert und durch ihren Namen oder ihre URL gefunden. Die gestrichelten Linien bedeuten, dass man über diese HashMaps zur entsprechenden Klasse navigieren kann. Zu den Klassen mit einem Schatten gibt es jeweils eine XML-Datei, aus der sie geladen oder gespeichert werden. Die folgenden Absätze erklären die einzelnen Klassen.

Key, CheckKey, SignKey (Schlüssel) Es gibt zwei Sorten von Schlüsseln. Die öffentlichen *CheckKeys* zum Überprüfen einer Signatur und die privaten *SignKeys* zum Erstellen. Beide können aus einer XML-Datei geladen werden und bestehen aus einer eindeutigen ID und einem Schlüsselwert, der als Text codiert in der Variable *encodedKeyValue* gehalten wird. *keyBytes* ist der gleiche Wert als Zahl. Wird der Schlüssel geladen, werden die Bytes aus dem Text erzeugt; wird er neu erstellt, werden die Bytes als Text codiert. *SignKey* enthält private Daten und ist deshalb mit einem Passwort geschützt. Beim Laden oder Speichern wird der Schlüsselwert mit diesem Passwort verschlüsselt.

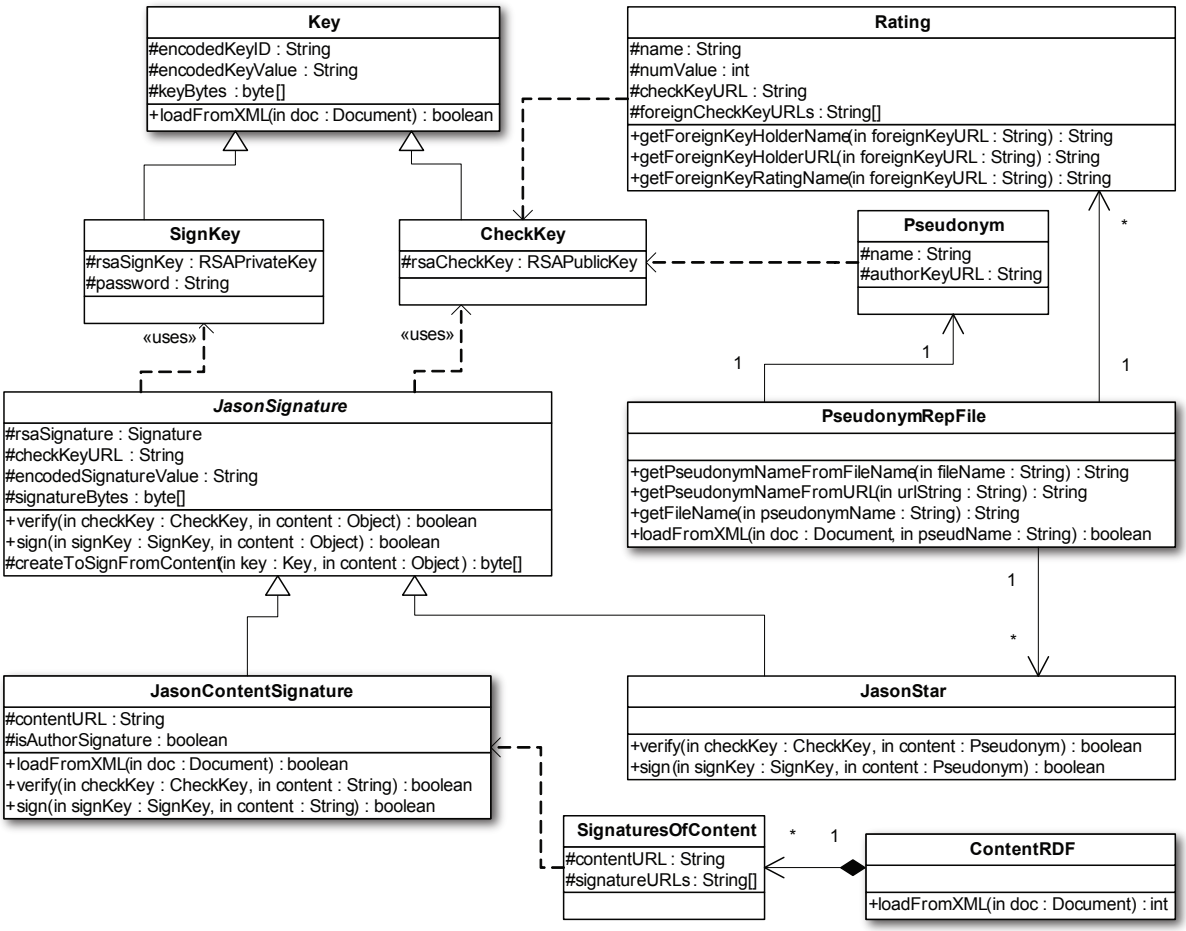

Abbildung 4.3: UML-Klassendiagramm der Datenklassen.

JasonSignature, JasonContentSignature, JasonStar (Signaturen) In JASON sind Signaturen wie gewöhnlich verschlüsselte Hash-Werte, die zusätzlich eine feste Bedeutung haben. Die abstrakte Basisklasse *JasonSignature* enthält alle benötigten Daten für die Signatur und zwei Methoden zum Erstellen und Prüfen der Signatur. Außerdem eine Methode *createToSignFromContent*, die den Hash-Wert erzeugt. Diese Methode muss von den beiden abgeleiteten Klassen implementiert werden. *JasonStar* und *JasonContentSignature* sind auf die Daten, die sie signieren, spezialisiert und enthalten Zusatzinformationen zur Bedeutung. *JasonStar* repräsentiert den in Kapitel 3.3 eingeführten Stern und signiert ein Pseudonym, während *JasonContentSignature* einen Inhalt signiert und entweder eine Wertung oder eine Autorenschaft angibt. Die Signaturen eines Inhaltes werden in extra XML-Dateien gespeichert.

ContentRDF, SignaturesOfContent (RDF) Eine JASON-RDF-Datei listet zu einem oder mehreren Inhalten die zugehörigen Signaturen auf. Für jeden Inhalt wird ein Objekt der Klasse *SignaturesOfContent* erstellt, das eine Liste der URL zu den Signaturen enthält.

PseudonymRepFile, Pseudonym, Rating (RepFile) Die öffentlichen persönlichen Daten eines Pseudonyms werden in XML-Dateien gespeichert, die von der Klasse *PseudonymRepFile* geladen werden. Der Name des Pseudonyms wird in den Dateinamen codiert. Die Klasse enthält Methoden, um den Pseudonym-Namen zu decodieren oder den Dateinamen zu erzeugen. Zu einem Pseudonym

gibt es eine gleichnamige Klasse. Sie enthält den Namen und den Link zum Autorenschlüssel, den jedes Pseudonym haben muss. In der XML-Datei sind außerdem die Wertungen beschrieben, die sich der Nutzer definiert hat (ohne ihre Bedeutung). Jede Wertung wird in ein Objekt der Klasse *Rating* geladen. Es enthält den Namen der Wertungen, einen numerischen Wert, den Link zum Wertungsschlüssel und eine Liste mit Schlüsseln anderer Pseudonyme, die die gleiche Bedeutung haben (nach Meinung des Pseudonym-Besitzers). Der numerische Wert war im ersten JASON-Protoypen die Bedeutung des Schlüssels[3]. Jetzt hat er keine feste Bedeutung, ist aber aus Gründen der Kompatibilität noch vorhanden. Eine Erweiterung zum ersten Protoypen sind die Sterne, welche auch in der XML-Datei gespeichert und in Objekte der Klasse *JasonStar* geladen werden.

Um den Wertungen eine Bedeutung zu geben und einen Reputationswert berechnen zu können, wird eine Metrik benötigt. Abbildung 4.4 zeigt den Aufbau solch einer Metrik. Den eben beschriebenen Wertungen werden hier Bedeutungen zugeordnet und den Pseudonymen Trust-Werte. Die Vergleichsfunktion für das Berechnen der subjektiven Bewerterreputation ist bereits enthalten. Die Trust-Metrik wird möglichst einfach umgesetzt und kann später durch einen komplizierteren Ansatz ersetzt werden.

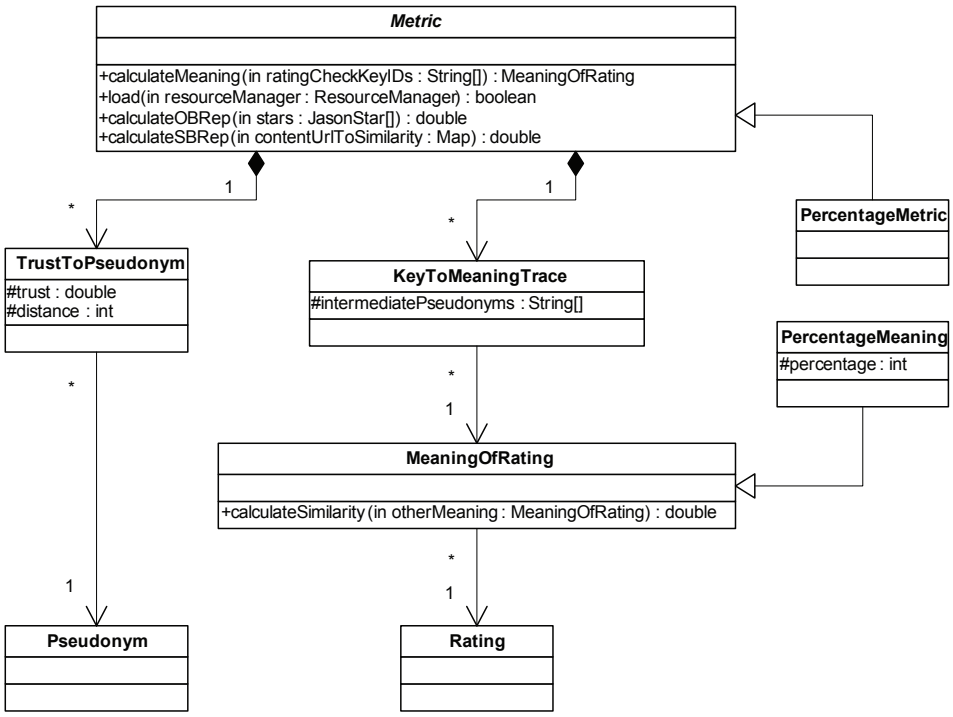

Abbildung 4.4: UML-Klassendiagramm der Metrik.

Metric Basisklasse für jede Metrik. Definiert die Berechnungsfunktionen *calculateMeaning*, *calculateOBrep* und *calculateSBRep*. Die beiden Bewerterreputationen werden auf einen Wert zwischen 0 und 1 abgebildet, wobei 1 der bestmögliche, und 0 der schlechteste Wert ist. Die objektive Bewerterreputation wird aus den Sternen gebildet und die subjektive aus den Vergleichswerten. Wie diese gebildet werden, wird in Kapitel 4.3 und 5.2 beschrieben. In *calculateMeaning* wird der Reputationswerte zu einem Inhalt berechnet. Das heißt, Wertungen, die einem Inhalt gegeben wurden, werden auf eine Bedeutung abgebildet. Als Information zum Berechnen der Bedeutung

[3]Im ersten JASON-Protoyp gab es keine austauschbare vertrauliche Metrik. Die Metrik war fest im FrontEnd implementiert.

stehen folgende Daten zur Verfügung: Die Bedeutung der Wertung, der Trust-Wert zum Erzeuger und, falls vorhanden, der subjektive und objektive Berwerterreputationswert des Erzeugers. Eine weitere Aufgabe der Metrik ist es, alle Bedeutungs- und Trust-Daten zu laden. Dabei können die Bedeutungen und Trust-Werte in extra Dateien gespeichert sein oder aus den vorhandenen Daten erzeugt werden (hängt von der konkreten Metrik ab).

TrustToPseudonym (Trust-Werte) Zu jedem Pseudonym muss es einen Trust-Wert geben, da man die Wertung sonst nicht interpretieren kann (Das ist eines der Grundprinzipien von JASON, siehe Kapitel 2.3). Wie dieser Wert aussieht, wird nicht festgelegt und sollte in dieser Klasse implementiert sein. Zunächst wird hier eine einfache Variante vorgesehen. Der Trust-Wert besteht aus einem einzigen Fließkommawert und der Information, über wie viele Pseudonyme hinweg man denjenigen kennt (0 man selbst, 1 direkt bekannt, 2 Bekannter eines Bekannten, ...).

MeaningOfRating, KeyToMeaningTrace (Wertungsbedeutung) Die Wertungen, die sich ein Nutzer definiert, bekommen eine feste Bedeutung. Dafür gibt es die Klasse *MeaningOfRating*. Diese enthält die Vergleichsfunktion (s_A siehe 3.1) zum Berechnen der subjektiven Bewerterreputation. Die Klasse *KeyToMeaningTrace* ist eine Hilfsklasse. Jeder Nutzer kann seinen Bedeutungen fremde Schlüssel zuordnen. Diese können wiederum anderen Schlüsseln zugeordnet werden (transitive Struktur). Damit die Abbildung, durch welche Pseudonyme ein Schlüssel zu seiner Bedeutung kommt, nicht jedes mal neu berechnet werden muss, wird sie in dieser Klasse gespeichert.[4]

PercentageMetric, PercentageMeaning (Implementierung) Diese beiden Klassen stellen eine einfache Implementierung einer Metrik dar. Die Bedeutung ist dabei ein ganzzahliger Prozentwert zwischen 0 und 100.

Damit sind die Grundlagen des Systems beschrieben. Ein weiterer wichtiger Teil ist die Kommunikation zwischen dem FrontEnd und dem Kern. Der Kern verwaltet die Daten, die Metrik und das soziale Netz (Bekannte, Trust-Werte). Das FrontEnd sammelt die zum Interpretieren benötigten Daten aus dem Internet. Um möglichst flexibel beim FrontEnd und beim Kern zu sein, kommunizieren die beiden über Nachrichten. Dadurch können sie mit unterschiedlichen Technologien oder von unterschiedlichen Anbietern umgesetzt werden. Es ist z. B. denkbar, dass es verschiedene FrontEnd-Versionen für verschiedene Endgeräte von verschiedenen Anbietern gibt. Abbildung 4.5 zeigt die Klassen, die an der Kommunikation beteiligt sind. Sie lassen sich grob in *FrontEnd*, *Core* und *Shared* (von beiden Seiten verwendet) einteilen.

SingleClientServerMessage, ClientServerMessage (Nachrichten) Ein Nachricht besteht aus mindestens einem Prefix. Darauf können beliebige viele als Text codierte Werte folgen, zwischen denen jeweils wieder ein Prefix liegen muss. Um diesen Aufbau einer Nachricht möglichst einfach zu beschreiben, gibt es die Klasse *SingleClientServerMessage*. Sie enthält Methoden zum Beschreiben des Aufbaus der Nachricht und zum Dekodieren und Kodieren. Nachrichten werden immer vom Client (FrontEnd) zum Server geschickt und können mehrere mögliche Antwortnachrichten auslösen. Diese Kommunikation wird in der Klasse *ClientServerMessage* zusammengefasst. Wiederum gibt es Methoden zum Codieren und Decodieren der Nachricht sowie zum Zugriff auf die Werte, die in einer Nachricht codiert waren. Die abstrakte Methode *initMessages()* muss von den konkreten Nachrichten-Klassen implementiert werden. In ihr müssen die Objekte für die einzelnen

[4] Schlüssel können auf diese Weise über unterschiedliche Wege zu unterschiedlichen Bedeutungen kommen. Siehe 5.2

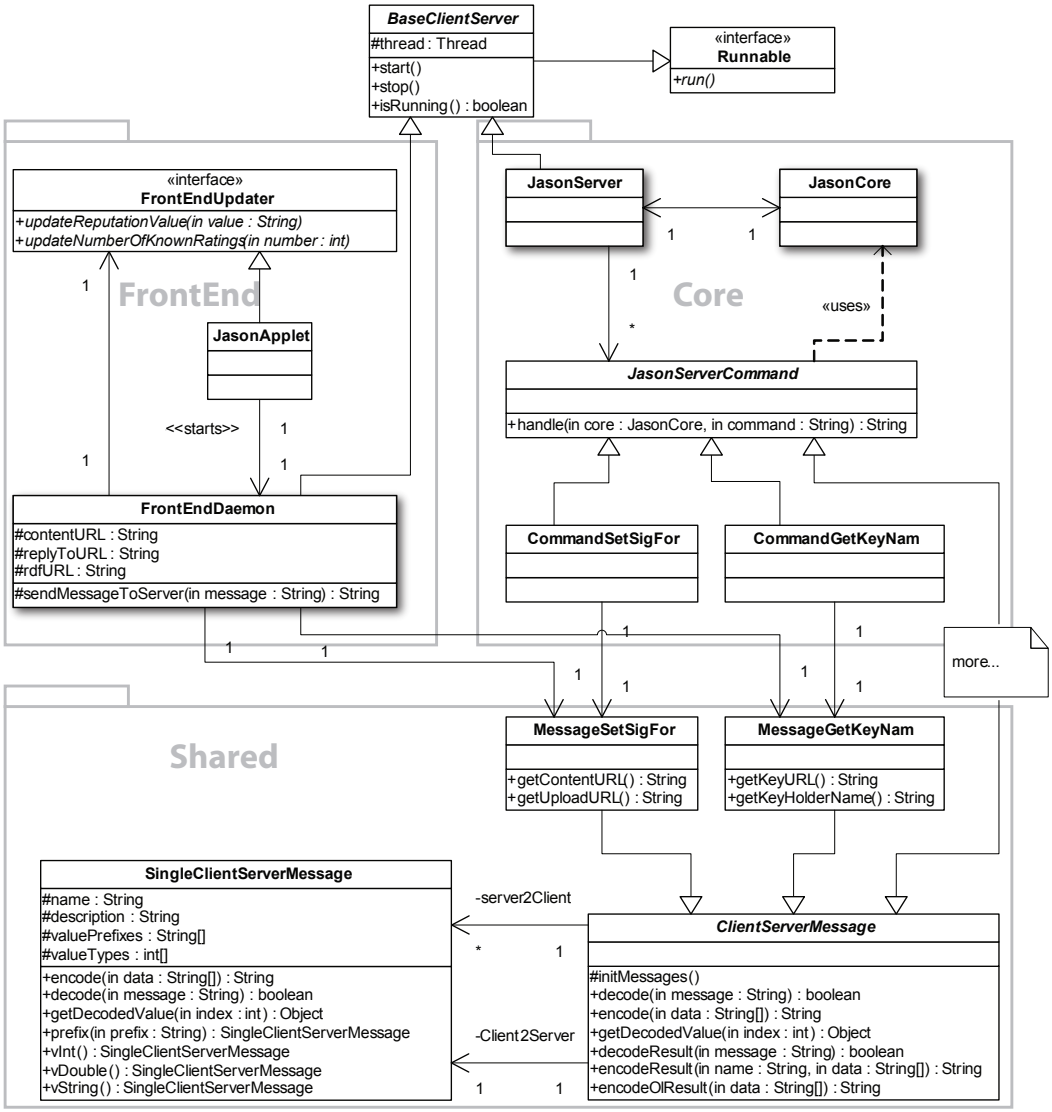

Abbildung 4.5: UML-Klassendiagramm der Client-Server-Strukutr.

Nachrichten angelegt werden. Beispielhaft sind zwei dieser Unterklassen dargestellt: *MessageSetSigFor*, *MessageGetKeyNam*. Welche Nachrichten es noch gibt, wird in Anhang A beschrieben.

BaseClientServer Sowohl der Kern als auch das FrontEnd benötigen einen Dienst zum Verschicken und Empfangen der Nachrichten. Die gemeinsamen Funktionen befinden sich in der Klasse *BaseClientServer*. Der Dienst enthält Methoden zum Starten und Stoppen.

JasonApplet, FrontEndDaemon, FrontEndUpdater (FrontEnd) Im dargestellten UML besteht das FrontEnd aus einem Applet (*JasonApplet*). Dieses Applet startet den *FrontEndDaemon*, der die Wertungsdaten sammelt und die Berechnung des Reputationswertes im Server auslöst. Um den berechneten Wert anzuzeigen, nutzt der Daemon das Interface *FrontEndUpdater*. In diesem Fall implementiert das Applet dieses Interface und zeigt die Werte an.

JasonServer, JasonCore, JasonServerCommand (Server) Jede Nachricht ist einem *JasonServerCommand* zugeordnet. Der *JasonServer* empfängt eine Nachricht, sucht ein passendes Kom-

mando und führt diese auf den *JasonCore* aus (Methode *handle*).

4.3 Systemverhalten

In diesem Kapitel werden zu den Anwendungsfällen aus Abbildung 3.1 Sequenzdiagramme gezeigt und erläutert. Abbildung 4.6 zeigt ein Squenzdiagramm zum Anwendungsfall *Reputation für einen Inhalt errechnen*. Im *FrontEnd* startet der Ablauf. Das *FrontEnd* ist z. B. ein Applet, angezeigt an einem Webinhalt. Es liest zuerst den Inhalt ein. Danach wird das RDF-Dokument, welches zum Inhalt gehört gelesen[5]. In dem RDF-Dokument werden Links zu Signaturen des Inhaltes aufgelistet. Diese werden geladen, womit der Zugriff auf den *ContentProvider* beendet ist. Der *ContentProvider* besteht in der Regel aus einem oder mehreren Webservern, auf denen die Dokumente gespeichert sind. Für jede Signatur fragt das *FrontEnd* beim Kern der Anwendung (*JasonCore*) den Wert des Schlüssels zum Überprüfen der Signatur ab. Wenn der Schlüssel unbekannt ist, kann die Signatur nicht interpretiert werden und wird nicht weiter verwendet. Mit den erhaltenen Prüfschlüsseln werden die Signaturen überprüft. Das *FrontEnd* ist nicht in der Lage, Signaturen zu interpretieren oder Wertungen zu berechnen, es überprüft nur, ob die Signaturen echt sind. Das *FrontEnd* übergibt dem Kern eine Liste der Signaturen, die erfolgreich überprüft wurden. Das bedeutet, die Signatur passt zum unterschriebenen Inhalt. Der Kern sammelt dann zuerst alle Daten, die zur Berechnung des Reputationswerte benötigt werden. Das sind die Bedeutungen der Signaturen (Wertungen) und die Bewerterreputationen der Erzeuger der Signaturen. Danach wird die Berechnungsfunktion der aktuellen Metrik aufgerufen. Der berechnete Wert wird schließlich an das *FrontEnd* zurückgegeben und dort angezeigt.

Für die Aufteilung, welche Aufgaben das *FrontEnd* übernimmt und welche der *JasonCore*, gibt es auch andere Möglichkeiten. Die Idee hinter der vorgestellten Variante ist, zeitaufwändige Vorgänge, die keine vertraulichen Daten benötigen, im *FrontEnd* zu halten. Es ist z. B. denkbar, dass sehr viele Webseiten gleichzeitig offen sind, die alle einen Reputationswert berechnen. Dann müsste der Kern viele parallel laufende Aufgaben bewältigen, um performant zu sein. Um das zu vermeiden, sind alle Zugriffe auf das Internet, die potentiell langsam sind, und die Verifikation der Signaturen im *FrontEnd* vorgesehen. Der Kern bekommt nur Aufgaben, die er unmittelbar und schnell bearbeiten kann. Das vereinfacht den Aufbau des Kernes (kein Multithreading) und bringt auch Vorteile für die Sicherheit. Wenn der Kern möglichst einfach bleibt, ist die Wahrscheinlichkeit für Sicherheitslücken geringer, was wichtig ist, da der Kern mit vertraulichen Daten arbeitet. Ein weiterer Vorteil liegt in der Skalierbarkeit. Es ist denkbar, viele verschiedene *FrontEnds* auf verschiedenen Geräten zu starten, während der Kern nur auf einem Server läuft.

Die Berechnung der beiden Bewerterreputationen wird in den beiden Sequenzdiagrammen 4.7 und 4.8 näher erläutert. Zunächst die Berechnung der subjektiven Bewerterreputation: Wie beim ersten Diagramm sind auch hier mehrere Objekte zusammengefasst unter den Bezeichnungen *JasonCore*, *ContentProvider* und *Metric*. Der *JasonCore* berechnet die subjektive Bewerterreputation eines Nutzers *B*. Das kann vom hier nicht dargestellten *FrontEnd* ausgelöst werden oder direkt im Kern (in der aktuellen Implementierung hat der Kern eine grafische Nutzerschnittstelle). Zunächst müssen alle öffentlichen Prüfschlüssel von *B* geladen werden, damit die Wertungen später überprüft werden können. Danach werden

[5] Der Link zum RDF-Dokument ist in der gegenwärtigen Implementierung ein Parameter des Applet.

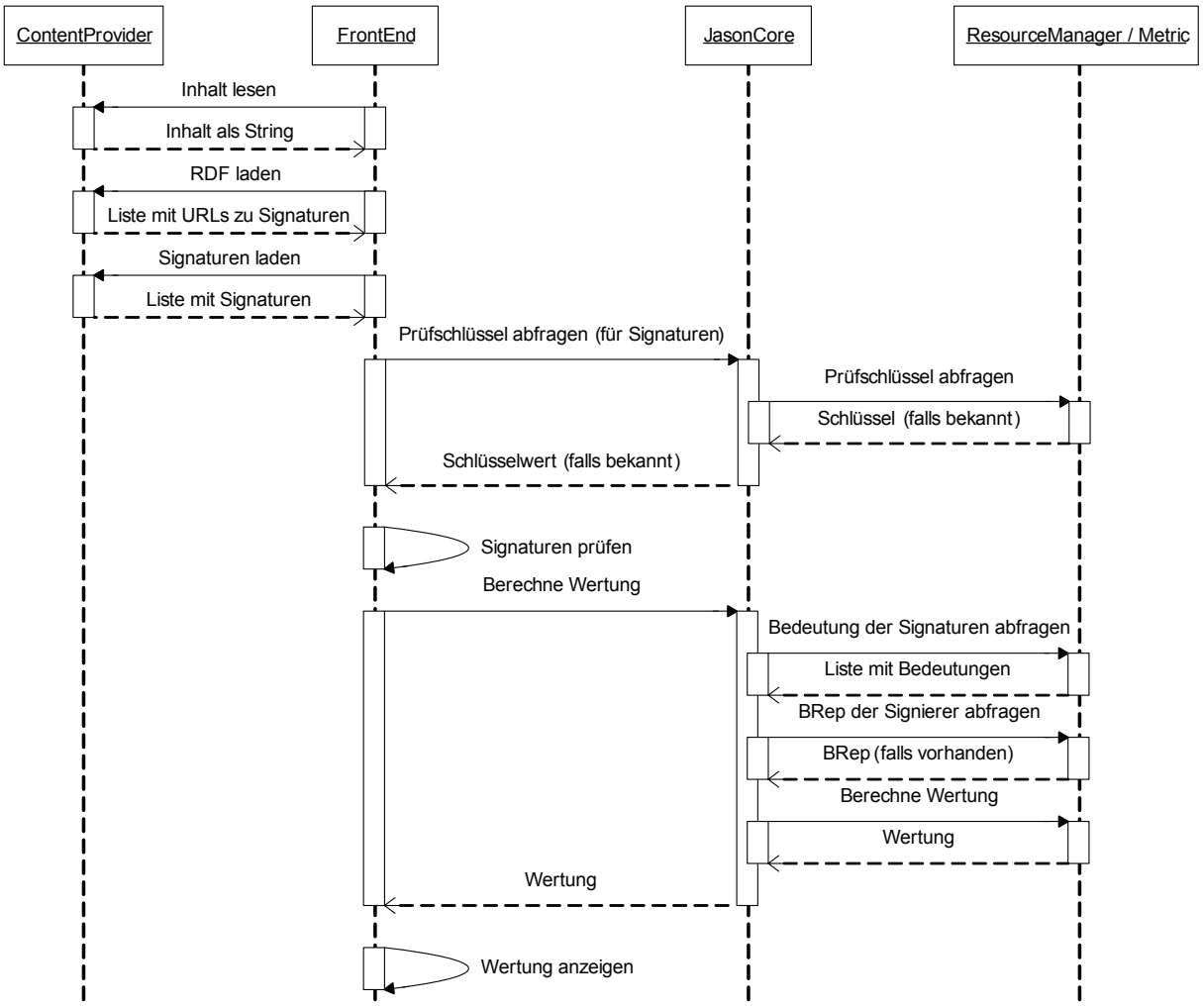

Abbildung 4.6: Berechnen des Reputationswertes von einem Inhalt.

alle bekannten Wertungen von *B* geladen. Da es keine zentrale Instanz gibt, bei der man alle Wertungen von *B* abfragen kann, stellt sich die Frage, wie man die Wertungen von *B* findet. Die aktuelle Implementierung speichert bei jeder Bewertung oder Reputationsberechnung eines Inhaltes, wer diesen bewertet hat. So entsteht eine nicht vollständige Liste, mit den Wertungen anderer Nutzer. Zu allen bekannten Inhalten, die *B* bewertet hat, müssen auch die eigenen Wertungen geladen werden. Das kann zusammen mit dem Laden der Wertungen von *B* geschehen. Da diese Zugriffe auf den *ContentProvider* (verschiedene Webserver) langsam sein können, sollte die Implementierung hier eine Caching-Strategie verwenden. Die geladenen Wertungen müssen mit den Prüfschlüsseln validiert werden. Falls *B* Inhalte bewertet hat, die man selbst noch nicht bewertet hat, könnte man anstatt der eigenen Wertung den berechneten Reputationswert für den Inhalt verwenden. Allerdings ist dieser von der SBRep abhängig, so dass man darauf achten muss, dass keine Endlosschleife entsteht. Jetzt sind alle Rohdaten für die subjektive Bewerterreputation vorhanden und die eigentliche Berechnung kann in zwei Schritten beginnen. Zunächst wird für jede Wertung von *B* ein Ähnlichkeitswert zur eigenen Wertung (oder dem berechneten Reputationswert) bestimmt. Aus dieser Liste von Vergleichswerten wird die subjektive Bewerterreputation berechnet und kann als Information angezeigt werden oder als Basis für die Gewichtung der Wertungen von *B* verwen-

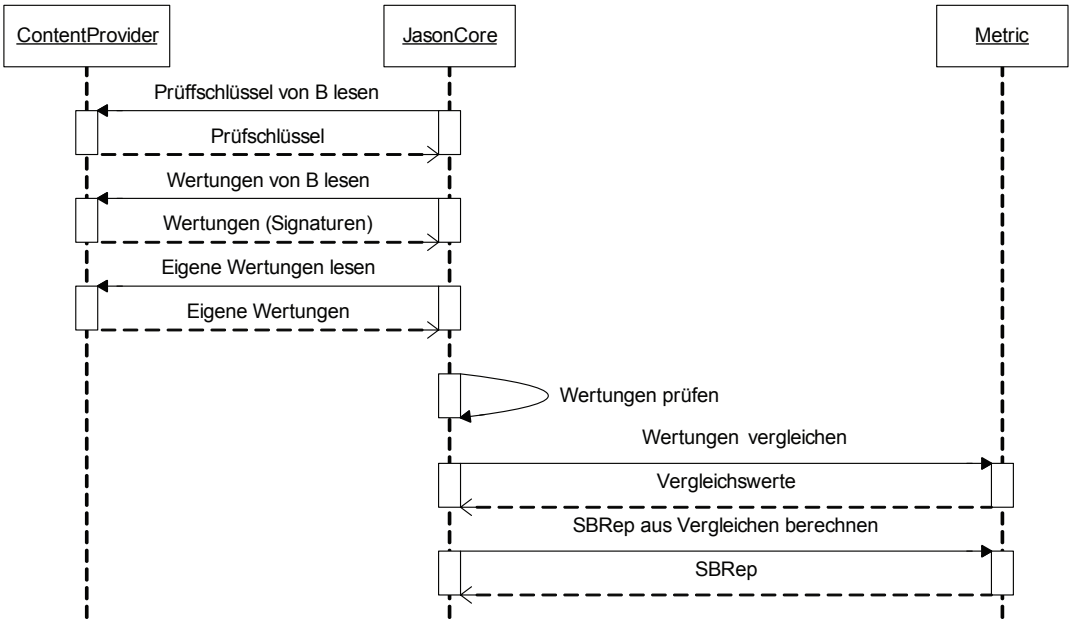

Abbildung 4.7: Berechnen der subjektiven Bewerterreputation eines Werters B.

det werden[6].

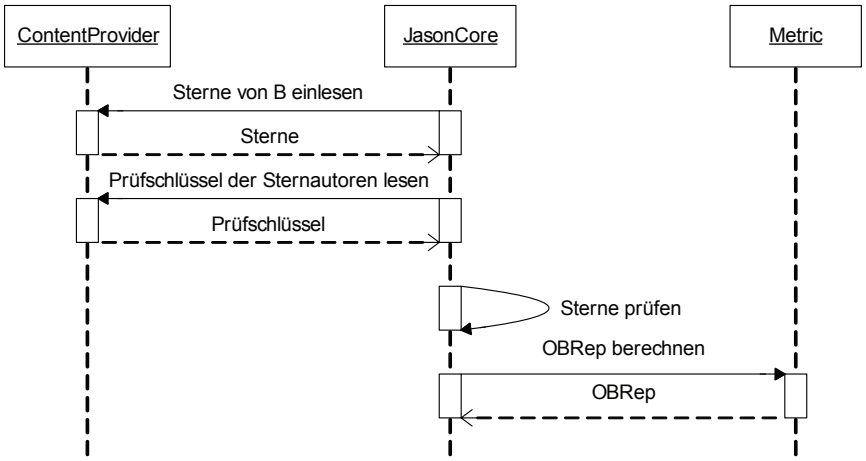

Abbildung 4.8: Berechnen der objektiven Bewerterreputation eines Werters B.

Die Berechnung der objektiven Bewerterreputation läuft ähnlich ab wie die der subjektiven Bewerterreputation, ist aber etwas unkomplizierter aufgrund der in Kapitel 3.3 getroffenen Vereinfachungen. Wiederum wird der Nutzer, für den die Bewerterreputation berechnet werden soll, mit *B* bezeichnet. Zuerst müssen alle Sterne von *B* geladen werden. Die Sterne sind Signaturen anderer Nutzer. Die Prüfschlüssel dieser anderen Nutzer müssen als nächstes geladen werden. Danach können alle Sterne überprüft werden. Zuletzt wird die objektive Bewerterreputation aus den erfolgreich überprüften Sternen berechnet. Da sich die objektive Bewerterreputation nicht sehr oft ändert, aber oft zur Berechnung von Reputationswerten benötigt wird, sollte auch hier eine Caching-Strategie verwendet werden.

[6]Die SBRep könnte man als berechneten Trust-Wert bezeichnen.

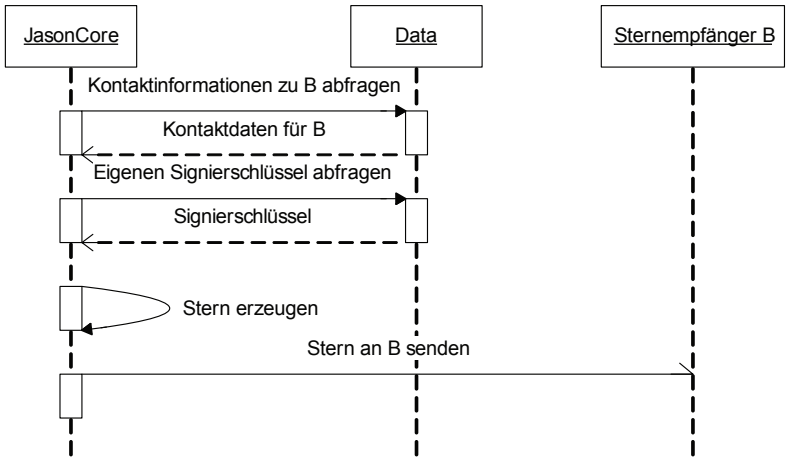

Abbildung 4.9: Erzeugen und Versenden von eines Sterns.

Der letzte Anwendungsfall ist das Erstellen und Vergeben eines Sterns. Abbildung 4.9 zeigt den Ablauf diese Vorgangs. *B* ist der Nutzer, für den ein Stern erzeugt werden soll. Bisher wurde nicht festgelegt, wie ein Stern zu seinem Empfänger gelangt. Deshalb wird im Diagramm nur von Kontaktdaten gesprochen. Diese müssen in den eigenen Daten vorhanden sein. Alternativ könnten sie auch manuell eingegeben werden (nicht dargestellt im Diagramm). Um den Stern zu erzeugen, benötigt man den eigenen Signierschlüssel, welcher als nächstes geladen wird. Danach wird der Stern erstellt, indem die Identitätsdaten von *B* signiert werden. Die entstehende Signatur wird dann an *B* gesendet, was hier als asynchroner Vorgang eingetragen ist. Wie man den Stern genau an *B* übergibt, ist eine Entscheidung der Implementierung. Wenn man den genauen Aufbau des Sterns festlegt, kann jeder den Stern überprüfen, da er sich die Daten, die signiert wurden, selbst erzeugen kann. Es reicht dann, nur die Signatur zu senden. Ansonsten müssen die Daten, die signiert wurden, mit übertragen werden.

Damit man Wertungen berechnen und Sterne erstellen kann, muss die Kern-Anwendung zuerst gestartet werden. Danach kann man einen neuen Nutzer erstellen oder sich als bereits erstellter Nutzer einloggen. Dadurch werden alle persönlichen Daten, die zur Berechnung benötigt werden, geladen. Als nächstes muss der Server gestartet werden. Danach kann der Nutzer nach Belieben eine der beschriebenen Aktionen ausführen, bis er sich am Ende ausloggt und das Programm beendet. Abbildung 4.10 zeigt diesen Ablauf als Aktivitätsdiagramm.

4.4 Sicherheitsanforderungen und Skalierbarkeit

In Kapitel 3.2 wurden die wichtigsten Anforderungen an die Sicherheit des Systems genannt. Hier soll nun erläutert werden, inwieweit die Anforderungen von dem entworfenen System umgesetzt werden.

Verfügbarkeit Die Verfügbarkeit hängt von der Verfügbarkeit der Daten ab, die zur Berechnung der Bewerterreputation benötigt werden. Für die SBRep sind das Wertungen von Inhalten und für die OBRep sind das die Sterne in den RepFiles der anderen Nutzer. In beiden Fällen müssen die öffentlichen Prüfschlüssel, mit denen die Sterne oder Wertungen validiert werden können erreichbar sein. Die Verfügbarkeit hängt also von dem Webserver ab, auf denen die RepFiles, Signaturen und

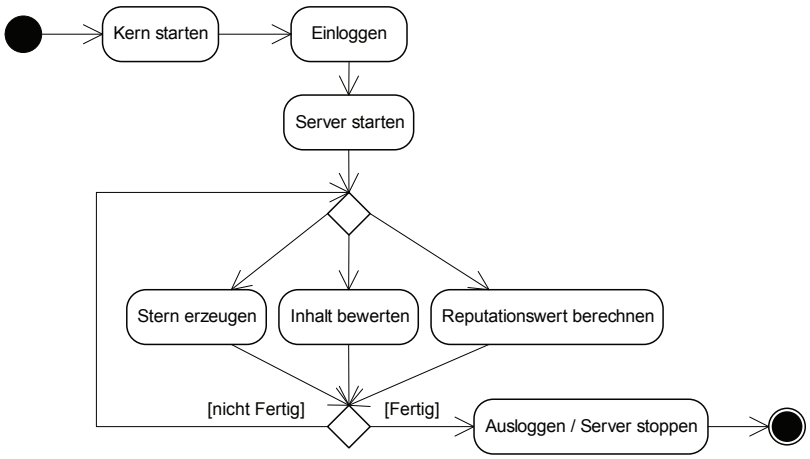

Abbildung 4.10: Ablauf und Zustände der Kernanwendung.

Schlüssel gespeichert werden. Diese Dateien können lokal zwischengespeichert werden. Das verbessert nicht nur die Verfügbarkeit, sondern auch die Performance. Man könnte einen Austausch von Informationen im sozialen Netz vorsehen, um nicht erreichbare Webserver zu kompensieren. Dazu müsste man eine direkte Kommunikation zwischen den JASON-Anwendungen vorsehen. Dadurch ergäben sich eine Reihe neuer Möglichkeiten (direkter Sternaustausch, Schlüsselweitergabe, usw.), aber auch ein erheblicher Implementierungsaufwand. Letztendlich hängt die Verfügbarkeit aber von der Sicherheit der Webserver ab und kann nicht durch das JASON-System alleine gewährleistet werden.

Vollständigkeit Bei der objektiven Bewerterreputation hat jeder Stern eine positive Bedeutung. Der Sternhalter ist also motiviert, alle Sterne anzuzeigen. Um einer Manipulation der Sternliste durch Dritte vorzubeugen, kann die Liste vom Sternhalter signiert werden. Das hilft aber nur, Manipulationen bei der Übertragung über einen unsicheren Kanal zu vermeiden. Bei der subjektiven Bewerterreputation ist Vollständigkeit nur schwer zu erreichen. Es gibt keine einfache Möglichkeit, alle Wertungen eines Nutzers zu finden. Unter Umständen kann es aber auch von Vorteil sein, nur einen Teil der Wertungen eines Nutzers zu verwenden, wenn man mehrere Wertungen zu einem ähnlichen Thema auswertet. So ergibt sich ein spezielles Bild des Wertungsverhalten des Nutzers zu diesem Thema. Wieviele Wertungen für eine aussagekräftige subjektive Bewerterreputation nötig sind, können nur praktische Untersuchungen zeigen.

Integrität Datengrundlage für beide Arten von Bewerterreputation sind digitale Signaturen. Diese sollten bei entsprechender Schlüssellänge hinreichend sicher sein. Ganz gleich ob im Bestand oder bei der Weitergabe von Daten, ist in diesen Fällen die Integrität gewährleistet. Grundlage der gesamten Sicherheit ist das Identitäts-Management-System, welches nicht Teil von JASON ist. Wenn Pseudonyme und ihre Autorenschlüssel sicher sind, dann sollte auch die Integrität der Bewerterreputation gegeben sein.

Pseudonymität Die Bewerterreputation wird für ein Pseudonym berechnet. Pseudonymität ist also gegeben. Allerdings funktioniert die objektive Bewerterreputation wahrscheinlich am besten für sehr bekannte Pseudonyme, die viele Sterne bekommen. Jeder Systemteilnehmer kann sehen, von welchen Pseudonymen die Sterne erzeugt wurden. Die Erzeuger der Sterne sind wiederum wahr-

scheinlich Bekannte des Sternempfängers. Wenn man viele Bekannte eines Pseudonymes kennt, wird es auch leichter, ein Profil über das Pseudonym zu erstellen und so den Nutzer dahinter zu entdecken. Durch Anonymisierung der Sterne wäre dieser Angriff deutlich schwerer. Dadurch bekommt man aber Probleme bei der Integrität der Sterne. Hier muss also ein Kompromiss gefunden werden. Der aktuelle Entwurf sieht Sterne mit sichtbaren Erzeugern vor. Dadurch umgeht man das Problem, wie man die Echtheit von Sternen überprüft, ohne ihren Erzeuger zu kennen.

Anonymität Anonymität für Nutzer ist durch Pseudonymität unterstützt. Pseudonyme können beim entworfenen System nicht anonym bleiben, da die Wertungen ihnen zugeordnet werden können. Die Bedeutung ist geheim, nicht aber, welches Pseudonym die Wertung erzeugt hat. Die subjektive Bewerterreputation berechnet sich jeder Nutzer innerhalb seines Vertrauensbereiches aus Daten, die bereits im ersten JASON-System vorhanden waren. Man kann also sagen, die subjektive Bewerterreputation ist anonym berechenbar. Bei der objektiven Bewerterreputation ist das anders. Der Sternempfänger kann nicht anonym bleiben, da sich dies nicht mit dem Ziel der objektiven Bewerterreputation vereinbaren lässt (sie soll für jeden sichtbar sein). Interessant ist die Frage, ob der Sternerzeuger anonym bleiben kann. Mit dem Erzeugen eines Sterns gibt er seine Ansichten preis. Wenn man viele Sterne kennt, die ein Pseudonyme erzeugt hat, kann man ein Profil über seine Ansichten erstellen. Anonymität wäre also wünschenswert. Allerdings bekommt man dadurch Probleme mit der Integrität. Wie will man den Wert und die Richtigkeit eines Sterns sicherstellen, ohne seinen Autor zu kennen? Dadurch könnten Sybil-Angriffe (siehe auch Kapitel 5.3) erleichtert werden, bei denen man viele Sterne erzeugt, ohne dass ein Außenstehender erkennen kann, dass diese Sterne alle von derselben Person stammen. Eine denkbare Lösung ist eine Art Währung, die nur begrenzt vorhanden ist und die man sich gegenseitig als Zeichen von Qualifikation übergibt. Ein solches System wäre genug Stoff für eine eigene Arbeit und wird hier deshalb nicht verwendet. Der Sternerzeuger ist sichtbar, gibt aber durch die Vereinfachung auf nur positive, binäre Sternbedeutung zumindest nicht preis, wen er schlecht findet. Anonymität des Bewerters bei der objektiven Bewerterreputation lässt sich also mit dem entworfenen System nicht erreichen.

Vertraulichkeit Vertraulichkeit der Wertungen wird durch die geheime Bedeutung der Signierschlüssel erreicht. Da die subjektive Bewerterreputation aus den Wertungen gebildet wird, ist hier das gleiche Maß an Vertraulichkeit gegeben. Die objektive Bewerterreputation ist aus Sicht der Vertraulichkeit ein Problem. Sie ist für jeden sichtbar und verrät viel über das Pseudonym, da die objektive Bewerterreputation per Definition etwas über die Kompetenz des Pseudonymes aussagt. Ob Bewertungen dieser Art öffentlich sichtbar sein dürfen, ist umstritten, wie zum Beispiel in [Hip09] beschrieben. Diese Diskussion wird hier nicht weiter geführt. Ein Implementierung und Evaluierung des Systems kann aber eine Diskussionsgrundlage bieten. Um potentiellen Schaden für die Sternempfänger zu begrenzen, wurden nur positive Sterne eingeführt. Es ist also nur sichtbar, wenn jemand besonders gut ist. Des weiteren muss der Sternempfänger seine Sterne nicht veröffentlichen. Vertraulichkeit ist hier also nicht gegeben. Dafür hat jeder die Kontrolle darüber, wie viel Information (Sterne) er über sich preisgibt, und das Schadenspotential dieser Information ist begrenzt.

Neben den Sicherheitsanforderungen soll auch kurz die Skalierbarkeit des Systems diskutiert werden. Der im nächsten Kapitel vorgestellte Prototyp dient der Validierung des Entwurfes und wird wahrschein-

lich nicht in einem Szenario eingesetzt werden, bei dem Performance ein Problem wird. Die Diskussion bleibt also theoretisch. Zunächst soll untersucht werden, welche Objekte theoretisch unbegrenzt vorkommen können und welche Auswirkungen dies hätte.

Nutzer Die Anzahl der Systemnutzer kann beliebig steigen. Das es sich bei JASON um ein verteiltes System handelt, hat dies keine direkten negativen Auswirkungen. Jeder Nutzer verwendet für die JASON-Core-Anwendung und das FrontEnd seinen eigenen Rechner.

Inhalte Die Inhalte die durch das JASON-System bewertet werden, existieren bereits im Web. Es werden lediglich einige Meta-Informationen angehängt. Es müssen keine Informationen lokal pro Inhalt gespeichert werden. Die Anzahl der Inhalte, die man bewerten und zu denen man eine Wertung berechnen kann spielt also keine Rolle bei der Performance.

Wertung für einen Inhalt Jede Wertung besteht aus einer XML-Datei, in der die Signatur enthalten ist. Diese Datei wird in den Meta-Daten des Inhaltes verlinkt (RDF-Dokument). Die Signatur-Dateien sind beim entwickelten Prototyp etwa 600 Byte groß und jeder Eintrag in der RDF-Datei etwa 80 Zeichen lang (abhängig von der Link-Länge). Das ist nicht sehr viel und könnte noch optimiert werden. Wird die Anzahl der Wertungen allerdings sehr groß, müssen Lösungen gefunden werden. Es könnten z. B. alte Wertungen gelöscht und so die maximale Anzahl begrenzt werden.

Wertungsschlüssel eines Pseudonymes Theoretisch kann sich ein Nutzer unendlich viele eigene Wertungen und Bedeutungen definieren. Praktisch scheint dies aber nicht sinnvoll, da man den Wert eines Inhaltes nicht beliebig genau bestimmen kann. Die Anzahl der zugelassenen Wertungen sollte also begrenzt werden.

Sterne Wenn man eine gute objektive Bewerterreputation erhalten möchte, muss man möglichst viele Sterne haben. Diese werden in dem RepFile eines Pseudonymes gespeichert. Dadurch kann dieses beliebig groß werden. Das wird erst bei sehr vielen Nutzern ein Problem und im Moment ist noch keine Lösung für dieses Problem vorgesehen. Eine Möglichkeit, Sterne zusammenzufassen oder alte Sterne zu löschen, ist denkbar.

Bekannte Jeder Nutzer speichert Daten über seine Bekannten. Dies geschieht hauptsächlich in der Rep-Datei. Zu den eigenen Wertungen werden fremde Wertungen angegeben, die eine gleiche Bedeutung haben. Die Rep-Datei kann dadurch groß werden. Es ist aber unwahrscheinlich dass eine einzelne Person, so viele fremde Wertungen kennt, dass die Anzahl für einen modernen Rechner zu groß wird. Im implementierten Prototypen werden zu jedem Bekannten einige Daten gecached. Da auch Bekannte von Bekannten betrachtet werden, muss die Suchtiefe begrenzt werden, da die Anzahl exponential wächst.

Bei den in Kapitel 4.3 angegebenen Sequenzdiagrammen gibt es einige Stellen, die problematisch werden könnten. Viele Informationen werden aus dem Internet geladen. Das kann langsam sein und die Anzahl der zu ladenden Information darf nicht beliebig ansteigen (z. B. mit der Gesamt-Nutzeranzahl). Außerdem sollten alle Daten aus dem Web gecached werden. Schlüssel ändern sich z. B. nicht und müssen nicht mehrfach geladen werden. Im Folgenden wird jedes der angegebenen Sequenzdiagramme kurz besprochen.

Reputationswert berechnen Hier gibt es drei Zugriffe auf das Web. Das Laden des Inhaltes ist immer der gleiche Zeitaufwand. Die Liste mit den URL zu den Signaturen kann jedoch sehr groß

werden. Wie bereits erwähnt, sollte die maximale Anzahl Wertungen pro Inhalt beschränkt werden. Eine andere Möglichkeit ist es, die Wertungen nach Alter zu sortieren und die Liste zu teilen. Dann kann das FrontEnd nur die gewünschte Anzahl aktueller Wertungen laden. Das gleiche gilt für das Laden der Signaturen. Wenn man es hier ermöglicht, anhand eines Eintrages in der Liste mit den Signaturen bereits zu erkennen, ob man die Signatur interpretieren kann, muss nur eine begrenzte Anzahl geladen werden[7].

SBRep berechnen Es müssen möglichst viele Wertungen eines anderen Nutzers geladen werden. Eine Möglichkeit ist es, immer dann wenn man selbst eine Wertung abgibt oder eine Reputation für einen Inhalt berechnet, lokal zu speichern, welcher Bekannte diesen Inhalt bewertet hat. Dann fällt das Suchen und Laden der Wertungen aus dem Web weg.

OBRep berechnen Die Sternanzahl kann zwar sehr groß werden, aber die objektive Bewerterreputation muss nicht sehr oft berechnet werden. Die Anwendungen sollte diese Berechnung im Hintergrund vornehmen und das Ergebnis speichern. Dann sollte es hier keine Probleme geben.

Stern vergeben Hier gibt es keine Probleme mit der Skalierbarkeit, da nur einzelne Objekte an dem Vorgang beteiligt sind.

[7] Der aktuelle Prototyp ermöglicht dies nicht.

5 Prototypische Validierung

Wie bereits in der ersten Arbeit zum JASON-System [Mei08], soll das entworfene System durch einen Prototypen validiert werden. Dabei wird ein neues System entworfen, welches gewonnene Erfahrungen des ersten Systems nutzt. Warum das erste System nicht einfach erweitert wurde, wird im ersten Unterkapitel 5.1 erklärt.

5.1 Der erste Prototyp

Der erste JASON-Prototyp entstand im Rahmen einer großen Belegarbeit. Anschließend wurde er für eine Diplomarbeit weiterentwickelt. Er zeigt, dass das Konzept funktioniert, setzt aber einige Teile des Konzeptes noch nicht um. Es gibt nur eine Anwenderdokumentation, aber keine Dokumentation für Entwickler. Deshalb soll hier ein kurzer Überblick zu diesem ersten Prototypen gegeben werden.

Die Software besteht aus vier Verzeichnissen, in denen sich jeweils ein Applet oder eine Java-Anwendung befinden. Hinzu kommen ein PHP-Upload-Script und die Schema-Datei für die JASON-XML-Dateien. Das PHP-Upload-Script wird verwendet, um Wertungen an einen Inhalt anzuhängen, genauer gesagt, um eine Signatur-XML-Datei zu erzeugen und diese in einem RDF-Dokument zu verlinken. Diese PHP-Datei muss auf den Webserver kopiert werden, auf dem bewertbare Inhalte liegen. Die vier Verzeichnisse enthalten drei Applets und die Kern-Anwendung. Dabei sind die meisten Klassen in jedem der vier Verzeichnisse vorhanden, also 4fach-redundant. Wenn man an einer Klasse etwas ändern möchte, muss man es in den vier Verzeichnissen ändern. Die vier Verzeichnisse sind:

jr	Kernanwendung, Verwalten von Bekannten und Wertungen.
jrApplet	Das Applet zum Anzeigen der Wertung eines Inhaltes.
jrAuthorCheckBoxApplet	Ein Applet um sich als Autor eines Inhaltes einzutragen.
jrAuthorIndicationApplet	Ein Applet zum Anzeigen des Autors eines Inhaltes.

jr entspricht dem, was bisher als JasonCore oder Kern bezeichnet wurde. Die anderen drei Verzeichnisse gehören alle zum FrontEnd. Allerdings enthalten sie auch vertrauliche Funktionen. Die Aufteilung entspricht also nicht ganz der im Entwurfskapitel (Kapitel 4) getroffenen Aufteilung.

Jedes Applet bekommt als Parameter drei URL: Eine URL verweist auf das RDF-Dokumente in dem zu einem oder mehreren Inhalten die Wertungen verlinkt werden. Eine zweite URL verlinkt den Inhalt, für den die Wertung berechnet werden soll. Dieser Inhalt muss im gegebenen RDF-Dokument enthalten sein. Als dritte URL wird ein Link zum PHP-Upload-Script übergeben, mit der neue Wertungen in das RDF-Dokument eingetragen werden können.

Will man die Autorenschaft zu einem Inhalt angeben, geschieht dies über ein eigenes Applet, welches man extra dafür starten muss. Das bedeutet auch, dass jeder Nutzer sich als Autor eintragen kann. Wie man das Angeben der Autorenschaft organisiert, wurde bis jetzt nicht diskutiert. Die Frage ist, wie man sicherstellt, dass die Autorensignaturen auch von den echten Autoren stammen. Im ersten Prototypen wurden dafür keine besonderen Sicherheitsvorkehrungen vorgesehen. Für den Prototypen sind die Sicherheitsvorkehrungen des Web-Servers ausreichend. Der Bereitsteller des Inhaltes (der die Kontrolle über den Webserver hat) muss darauf achten, dass die Autorensignaturen stimmen. Die Implementierung dieser Arbeit ändert daran nichts. Eine parallel entwickelte Erweiterung von JASON um Autorenreputation beschäftigt sich mit diesem Thema genauer [Bar10].

Im alten JASON-System wird jeder Wertung ein numerischer Wert beigefügt. Dieser wird verwendet, um fremde Wertungen automatisch den eigenen zuzuordnen, aber auch zum Berechnen der Reputation. Hier werden Bedeutung und Wertung also nicht getrennt. Weiterhin wird der Reputationswert im Applet (FrontEnd) berechnet. Es gibt keine Möglichkeit, die verwendete Metrik auszutauschen und die Bedeutung einer Wertung ist, über den numerischen Wert, für jeden sichtbar. Für eine erste Validierung war diese Vereinfachung ausreichend. Im neuen Prototypen sollen Bedeutung und Wertung getrennt werden.

Jedes der vier Verzeichnisse verwendet die gleiche Paketstruktur. Die meisten Klassen gibt es genau identisch in jedem Verzeichnis. Einige Klassen kommen nur in einigen Verzeichnisse vor. Die Pakete befinden sich alle unter *org.cubeweb.jason*. Dort gibt es fünf Pakete mit folgendem Funktionsumfang:

crypt	IDs generieren
	PBE, Hash-Funktionen
	Schlüssel erzeugen
	Signaturen erzeugen
data	Cache
	Request-Handler (für Server: Empfangene Nachrichten abarbeiten)
	Background-Daemon (für Applet: Nachrichten an Server schicken)
	Icons und Logo
	Preferences (Programmeinstellungen)
	User-Data (Name und Passwort)
	Modelle für Swing-Komponenten
div	Base64-Encoding
	Hilfsklasse für Threads
GUI	Komponenten und Dialoge
IO	XML-Hilfsklassen
	Cache laden und speichern
	FTP-Funktionen
	Server

Die beiden wichtigsten Klassen sind *JasonServerRequestHandlerRunnable* und *BackgroundDaemonRunnable*. In *JasonServerRequestHandlerRunnable* werden die Nachrichten von einem oder mehreren laufenden Applets verarbeitet. Das geschieht in einer knapp 300 Zeilen langen Methode, die etwas unübersichtlich ist. Die meiste Funktionalität steckt direkt in dieser Methode. Das Gegenstück im Applet

ist die Klasse *BackgroundDaemonRunnable*. Hier werden Schlüssel abgefragt, Wertungen berechnet, angezeigt und zum Cachen an den Kern gesendet.

Der erste Prototyp ist vollständig funktionsfähig und enthält einige gute Cachingstrategien. Außerdem wurde er bereits einmal überarbeitet und vor allem auf dem Gebiet Performance und Plattformunabhängigkeit verbessert. Eine Integration der entworfenen Erweiterung wäre möglich. Es gibt aber einige Gründe, die für ein komplettes Redesign sprechen. Der Arbeitsaufwand ist für diese Arbeit zwar höher als er sein müsste, aber nachfolgende Arbeiten werden davon sehr profitieren. Folgende Gründe sprechen für einen neuen Prototypen:

1. Wertung und Bedeutung nicht getrennt
2. Definition von Metriken nicht möglich
3. Berechnung erfolgt im Applet in einer 500 Zeilen langen Methode (sollte im Kern sein)
4. Objektorientierte Basisstrukturen kaum vorhanden
5. unübersichtlicher reduntanter Code, schwer erweiterbar und wartbar
6. keine Entwicklerdokumentation

Aus diesen Gründen wurde ein komplettes Redesign, vor allem an der Basis, durchgeführt. An den Datenstrukturen wurden nur Erweiterungen vorgenommen, und es wurden Java-Klassen für jede Struktur eingeführt. So ist der neue Prototyp weitgehend kompatibel zum alten. Die Struktur der Signaturen und XML-Dateien wurde nicht verändert. Auch die Nachrichten zwischen FrontEnd und Server wurden übernommen. Nach außen hin sichtbar hat sich nur die GUI des Kernes verändert. Der Code darunter hat sich aber bei ähnlicher Funktionalität stark verändert.

5.2 Der neue Prototyp: JASON-RaterReputation

5.2.1 Paketstruktur und Grundlagen

Der neue Prototyp ist, genau wie der alte, eine Java-Anwendung. Er besteht aus zwei Teilen. Dem FrontEnd, welches als Applet implementiert wird, und dem Kern, als normale Java-Anwendung mit GUI. Beide Teile sind jeweils in einem Paket, *frontend* und *core*, untergebracht. In einem dritten Paket *shared* befinden sich alle Klassen, die von beiden Anwendungen benötigt werden. Um das System zu nutzen, benötigt man außerdem einen Web-Space samt FTP-Zugang. Dorthin werden das eigene RepFile und die öffentlichen Prüfschlüssel hochgeladen. Um das System zu nutzen, startet man die Kern-Anwendung und meldet sich an oder erstellt einen neuen Nutzer. Während die Kern-Anwendung läuft, können Web-Seiten, die das FrontEnd-Applet eingebunden haben, Wertungen berechnen und das Erzeugen von Wertungen auslösen.

Folgende Liste erklärt den Inhalt der wichtigsten Pakete:

core	enthält *JasonCore*, stellt API für GUI bereit
gui	GUI zu *JasonCore*
metrics	Metriken des Nutzers

server	*JasonServer*, empfängt und führt Nachrichten aus
commands	Jeder Nachrichtentyp hat eine Kommandoklasse mit der Funktionalität
frontend	Enthält den *FrontEndDaemon*, Gegenstück zu *JasonServer*, sendet Nachrichten an diesen
applet	Applet zum Anzeigen der Wertung
shared	Alle Klassen die vom FrontEnd und Core benötigt werden
baseobject	Java-Spezifische Hilfsklassen
crypt	PBE, Base64-Encoding, Hash-Funktion, ID-Generator
data	Alle Datenklassen
key	Signier- und Prüfschlüsselklassen sowie eine Factory-Klasse
logo	Hilfsklasse zum Icons laden und Klasse für das Jason-Logo
metric	Basisklassen für jede Metrik
preferences	Konstanten und Einstellungen für die Kernanwendung
rdf	RDF-Dokument
repfile	Die Rep-Datei
signature	Jason-Signaturen, 2 Klassen für Wertungen und Sterne
io	Alle Klassen für Laden (lokal und Web), Speichern und Hochladen von Daten
xmltool	Hilfsklassen zum Einlesen und speichern von XML-Dateien
server	Basisklassen für das Empfangen und Verschicken von Nachrichten

Alle Klassen werden von einer Basisklasse *BaseObject* abgeleitet. Diese Klasse enthält Java-spezifische Hilfsfunktionen, die der Übersichtlichkeit und Vereinfachung des Codes dienen. *BaseObject* erbt von der Klasse *Logger*, die Methoden zum Logging enthält. Dadurch kann die Art und Weise des Logging sehr einfach verändert werden.

Grundlegende Kryptografie-Funktionen befinden sich im Paket *crypt*. Zum Speichern von Zahlenwerten, wie Schlüssel und Signaturen, wird eine Base64-Kodierung verwendet. Die privaten Schlüssel und das Passwort zum FTP-Zugang werden mit dem Nutzer-Passwort, das man zum Einloggen benötigt, PBE[1]-verschlüsselt. Ein *IDGenerator* erzeugt aus Schlüsseln oder Signaturen IDs, die zum Erzeugen der Dateinamen verwendet werden.

5.2.2 Signaturen

Es gibt zweit Arten von Signaturen im System. Signierte Identitätsdaten bilden einen Stern und signierte Inhaltsdaten eine Wertung, oder sie geben die Autorenschaft an. Die Klasse *JasonSignature* bildet die Basisklasse für beide Arten und stellt Methoden zum Erzeugen (sign) und Prüfen (verify) der Signatur bereit. Listing 5.1 zeigt den Ablauf dieser Methoden. Zunächst wird in Zeile 3 und 12 ein Byte-Array aus dem zu signierenden Inhalt erzeugt. Danach werden Java-RSA-Funktionen genutzt, um die Signatur zu erzeugen oder zu prüfen. Die Methode *createToSignFromContent* ist abstrakt und muss von den erbenden Klassen *JasonContentSignature* und *JasonStar* implementiert werden. In dieser Methode wird

[1] Password-based encryption. Eine Funktionalität aus dem *javax.crypto*-Paket zum Verschlüsseln von Daten mit Hilfe eines Passworts.

genau festgelegt, wie ein Stern aussieht und wie ein Inhalt signiert wird. Listing 5.2 zeigt die beiden Implementierungen dieser Methode. Bei der Inhalts-Signatur wurde der Aufbau von der ersten JASON-Implementierung übernommen, um kompatibel zu bleiben. An den eigentlichen Inhalt werden noch die ID des Prüfschlüssels und die Aussage, ob es sich um eine Autorensignatur handelt, angehangen. Für den Aufbau des Sterns gibt es, wie in Kapitel 3.3 beschrieben, verschiedene Möglichkeiten. Hier wird die einfachste gewählt. Pseudonymname und öffentlicher Autorenschlüssel werden zusammen signiert. Da der Sterntext (*contentString* Zeile 20) immer gleich aufgebaut ist, muss nur der Signatur-Wert weitergegeben werden.

```
1  public boolean verify( CheckKey checkKey, Object content )
2  {
3      byte[] toSign = createToSignFromContent( checkKey, content ); // daten erzeugen
4      rsaPublicKey = checkKey.getRsaCheckKey();
5      rsaSignature.initVerify( rsaPublicKey ); // Java-RSA-Funktionalität
6      rsaSignature.update( toSign, 0, toSign.length );
7      return rsaSignature.verify( signatureBytes ); // verify
8  }
9
10 public void sign( SignKey signKey, Object content )
11 {
12     byte[] toSign = createToSignFromContent( signKey, content ); // daten erzeugen
13     rsaPrivateKey = signKey.getRsaSignKey();
14     rsaSignature.initSign(rsaPrivateKey);
15     rsaSignature.update( toSign, 0, toSign.length );
16     signatureBytes = rsaSignature.sign(); // sign
17     encodedSignatureValue = Base64.encodeBytes( signatureBytes, Base64.URL_SAFE );
18     encodedID = makeSignatureID( signatureBytes );
19 }
```

Abbildung 5.1: Erzeugen und Überprüfen einer Signatur in der Klasse *JasonSignature*.

5.2.3 Datenverwaltung

Um möglichst einfach auf die Daten (Schlüssel, RepFile, ...) zugreifen zu können, werden diese von der Klasse *ResourceManager* verwaltet. Die Klasse übernimmt das Laden und Speichern von der Festplatte sowie das Herunterladen und Hochladen aus dem Web. Auch die Bekannten von Bekannten werden hier bis zu einer eingestellten Tiefe geladen. Gespeichert werden die Daten in HashMaps. Welche Schlüssel dabei auf welche Werte verweisen, zeigt Abbildung 5.3. Schlüssel und Pseudonyme werden immer über ihre URL angesprochen. Das wurde von der bisherigen Implementierung übernommen, hat sich aber als ungünstig erwiesen und sollte bei der nächsten Überarbeitung des Systems geändert werden. Schlüssel-IDs und Pseudonyme allein sind bereits eindeutige Identifikatoren und sollten nicht mit URL gemischt werden. Die geladenen RepFiles enthalten keinen direkten Link auf ihre Wertungsschlüssel, sondern nur die URL der Schlüssel, so wie sie in den XML-RepFiles stehen. Der *ResourceManager* lädt auch die Metriken. Dies geschieht über eine Hilfsklasse *MetricManager*. Die geladenen Metriken enthalten

```
1  public class JasonContentSignature extends JasonSignature
2  ...
3      protected byte[] createToSignFromContent( Key key, String contentString )
4      {
5          String ending;
6          ending = key.getEncodedKeyID() + Boolean.toString( isAuthorSignature );
7          byte[] contentBytes = contentString.getBytes("UTF-8");
8          // UTF-16LE um kompatibel mit bisheriger Impl. zu sein
9          byte[] endingBytes = ending.getBytes("UTF-16LE");
10         return Utils.concatArray( contentBytes, endingBytes );
11     }
12
13 public class JasonStar extends JasonSignature
14 ...
15     protected byte[] createToSignFromContent(Key key, Pseudonym pseudonym )
16     {
17         CheckKey authorCheckKey;
18         String   contentString;
19         authorCheckKey = resourceManager.getCheckKey(pseudonym.getAuthorKeyURL());
20         contentString  = pseudonym.getName() + authorCheckKey.getEncodedKeyValue();
21         return contentString.getBytes("UTF-8");
22     }
```

Abbildung 5.2: Erzeugen der zu signierenden Daten für eine Wertung und für einen Stern.

zwei weitere HashMaps, die in der Abbildung grau unterlegt dargestellt sind. Zu jedem Pseudonym wird ein Trust-Wert gespeichert, und zu jeder Wertung, identifiziert durch ihre Prüfschlüssel-URL, eine Bedeutung.

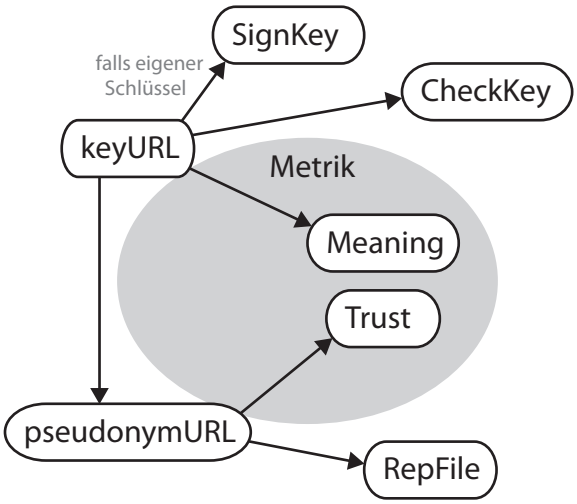

Abbildung 5.3: Verlinkung von Daten im *ResourceManager*.

Alle Daten werden lokal in einem Archiv gespeichert. Der Name setzt sich aus dem Pseudonymnamen und der Endung *.jason* zusammen. Der *ResourceManager* nutzt die Klasse *IOManager*, um aus diesen Archiven zu laden und zu speichern. Außerdem enthält der *IOManager* Methoden zum Laden von URL aus dem Web und zum Hochladen per FTP. Das Jason-Archiv ist eine ZIP-Datei und enthält folgende Verzeichnisse mit dem beschriebenen Inhalt:

keys Alle eigenen öffentlichen und privaten Schlüssel. (Autorenschlüssel und Wertungsschlüssel)

userData RepFile, Einstellungen zum JASON-System, Hash-Wert des Nutzerpassworts, sonstige Daten der Kern-Anwendung[2]

metrics Enthält für jede Metrik ein Verzeichnis. Darin wird gespeichert, welcher Wertungsschlüssel zu welcher Metrik gehört, welche Bedeutung diese haben, und alle sonstigen Daten, die eine Metrik benötigt (z. B. Trust-Werte, Cache für SBRep und OBRep, usw.).

acquaintances Die RepFiles der Bekannten (und deren Bekannten) sowie ihre öffentlichen Schlüssel werden hier gecached. Für jeden Bekannten wird ein Verzeichnis angelegt. Der Verzeichnisname ist der Pseudonymname. Zusätzlich wird in einer XML-Datei gespeichert, von welcher URL die RepFiles geladen wurden.

Alle Daten werden in XML-Dateien gespeichert. Die Struktur dieser Dateien wurde nicht verändert. Die einzige Erweiterung ist das Speichern der Sterne in den RepFiles. Außerdem sind für die Metriken einige Dateien hinzugekommen. Anhang E zeigt, wie diese XML-Dateien beispielsweise aussehen.

[2]Die grafische Oberfläche des Kerns muss in der Lage sein, Daten zu speichern, die zum Funktionieren des Systems nicht notwendig sind, aber die Benutzbarkeit stark verbessern.

5.2.4 FrontEnd und JasonCore

Das *FrontEnd* besteht, wie schon bei der ersten JASON-Implementierung, aus einem Java-Applet. Dieses Applet wurde kaum verändert und nur überarbeitet, um den Code übersichtlicher zu machen. Das Applet startet den *FrontEndDaemon* in einem eigenen Thread. Dieser startet die Berechnung des Reputationswertes, wie sie in Kapitel 4.3 und Abbildung 4.6 beschrieben wurde. Außerdem löst er das Erzeugen einer Wertung aus, indem er eine Nachricht an den Server schickt. Damit der *FrontEndDaemon* die Ergebnisse anzeigen kann, greift er über das Interface *FrontEndUpdater* auf das Applet zu. Jedes FrontEnd sollte dieses Interface implementieren. Es enthält im Moment zwei Methoden, *updateReputationValue* und *updateNumberOfKnownRatings*. Die möglichen Anzeigen sollten in zukünftigen Arbeiten erweitert werden. Das hat die Evaluierung in der Vorgängerarbeit gezeigt. Aus Zeitgründen wurde aber kein komplett neues FrontEnd entwickelt.

Die Nachrichten zwischen *FontEndDaemon* und *JasonServer* sind in Anhang A beschrieben. Beide Klassen erben von *BaseClientServer*. Diese Klasse enthält Funktionen zum Starten und Stoppen des Threads sowie zum Verschicken und Empfangen von Nachrichten. Nicht mehr benötigte Nachrichten, aus der ersten Implementierung, werden immer noch unterstützt. Dadurch funktionieren die alten Applets auch mit dem neuen Kern. Eine neue Nachricht ist hinzugekommen: das Berechnen des Reputationswertes. Die alte Implementierung hat diese Berechnung im FrontEnd vorgenommen und deshalb keine Nachricht dieser Art benötigt. Jetzt wird die Berechnung von den Metriken im Kern erledigt.

Für jede Nachricht, die der *JasonServer* empfängt, gibt es eine von *JasonServerCommand* abgeleitete Klasse. Diese enthält die Methode *handle*, in der die Nachricht dekodiert und die gewünschte Funktionalität ausgeführt wird. Als Ergebnis liefert die Methode die Antwortnachricht zurück. Die Kommandos werden auf dem Kern der Anwendung ausgeführt. Dieser wird durch die Klasse *JasonCore* repräsentiert. *JasonCore* stellt eine API bereit, um alle nötigen Funktionen einfach auszuführen. Diese API wird von der grafischen Oberfläche des Kerns verwendet, könnte aber auch von einer Kommandozeile aus verwendet werden. Die folgende Liste zeigt, welche Funktionen von *JasonCore* zur Verfügung gestellt werden. Anhang B beschreibt die entsprechenden Methoden-Aufrufe genauer. Listing 5.4 zeigt ein Beispiel für eine API-Methode.

- login / logout
- start / stop Server
- Bekannten hinzufügen
- neue Metrik anlegen / Wertung hinzufügen / Bedeutung fremder Wertung setzen
- neue RDF-Datei erzeugen / Inhalt hinzufügen
- Inhalt bewerten / Wertung ins Web laden / Autorenschaft angeben
- Einstellungen speichern / RepFile speichern / ins Web laden

```
1  public Rating newRating( String name, String metricName )
2  {
3      Rating result = null;
4      repFile = resourceManager.getCurrentRepFile(); // get repfile
5      preferences = resourceManager.getPreferences();
6      // gibt es schon ein Rating mit diesem Namen?
7      if( isNull( repFile.getRatingByName(name) ) )
8      {
9          ratings = repFile.getRatings();
10         // Schlüssel erzeugen
11         keyFactory = resourceManager.getKeyFactory();
12         signKey = keyFactory.newSignKey();
13         signKey.setPassword( password );
14         checkKey = keyFactory.newCheckKey();
15         keyFactory.createKeyPair( signKey, checkKey, preferences.getKeyLength() );
16         checkKeyURL = makeCheckKeyURL( checkKey );
17         // Wertung erzeugen
18         result = new Rating();
19         result.setName( name );
20         result.setNumValue(0);
21         result.setCheckKeyURL( checkKeyURL );
22         ratings.add( result );
23         // Schlüssel cachen und speichern
24         resourceManager.addKeyPair( checkKeyURL, checkKey, signKey );
25         resourceManager.saveRepFile();
26         // Bedeutung setzen und speichern
27         metricManager = resourceManager.getMetricManager();
28         metric = metricManager.getMetrics().get( metricName );
29         meaningOfRating = metric.newMeaning( result );
30         metric.addMeaning( meaningOfRating );
31         metricManager.saveMetric( resourceManager, metric );
32     }
33     return result;
34 }
```

Abbildung 5.4: API-Methode zum Erzeugen einer neuen Wertung.

5.2.5 Metriken

Bisher wurde nicht gesagt, wie man die Reputation eines Inhaltes oder die Bewerterreputation genau berechnet. Diese Berechnung gehört nicht zu den Kernpunkten der Arbeit. Die Frage war, was benötigt man, um diese Werte zu berechnen, und wie organisiert man es, diese Werte zur Verfügung zu haben. Bei der Implementierung muss es die entsprechenden Funktionen trotzdem geben. Wie sie genau implementiert wurde und welche Probleme es dabei gibt, zeigen die nächsten Absätze.

Als Beispielmetrik ist eine sehr einfach Variante implementiert. Die Bedeutung einer Wertung ist eine Prozentzahl zwischen 0 und 100, wobei 100 der beste Wert ist und 0 der schlechteste. Die entsprechende Klasse heißt *PercentageMetric*, die Bedeutungsklasse heißt *PercentageMeaningOfRating* und enthält den Zahlenwert zwischen 0 und 100. Zu der Metrik gibt es einen View, der die Wertungen auf einem Zahlenstrahl darstellt. Auch für die subjektive Bewerterreputation gibt es einen metrikspezifischen View, der grafisch darstellt, wie ein Bekannter im Vergleich zu den eigenen Wertungen wertet. Die Kurzanleitung in Anhang C enthält einige Screenshots, die das zeigen. Andere Metriken sind denkbar, z. B. könnte man eine zweidimensionale Bedeutung definieren, die beurteilt, wie hoch der Informationsgehalt eines Inhaltes ist und wie gut diese Information dargestellt wird. Eine entsprechende Metrik wurde unter dem Titel *StyleAndTechnicMetric* entwickelt. Weitere Metriken könnten einfach aus JAR-Dateien geladen werden.

Jede Metrik muss nicht nur für direkte Bekannte Trust-Werte und Schlüsselbedeutungen definieren, sondern auch für die Bekannten der Bekannten, bis zu einer einstellbaren Tiefe. Dabei muss zu jedem Pseudonym ein Trust-Wert berechnet werden. Eine mögliche Implementierung wurde bereits in der Vorgängerarbeit [Mei08] erläutert. Dabei werden Wertungen nach Alter und Entfernung gewichtet. Die Entfernung gibt an, wie lang die Kette ist, über die man den Bekannten kennt, der die Wertung abgegeben hat. Daran wurde bei der Implementierung dieser Arbeit nichts geändert. Allerdings stehen der Berechnung jetzt auch die Bewerterreputationen zur Verfügung, welche eine weitere Grundlage für die Gewichtung darstellen.

Wie die angesprochene *PercentageMetric* den Reputationswert berechnet, zeigt Listing 5.5. Die Prozentwerte der Wertungen werden gewichtet addiert (Zeile 14) und am Ende durch die Summe der Gewichte geteilt (Zeile 20). Die Liste der Wertungen ist nach Alter sortiert, wobei die neuesten Wertungen zuerst kommen. Wie man die Gewichte in Zeile 12 berechnet, ist eine schwierige Frage. Die aktuelle Implementierung verwendet einfach die subjektive Bewerterreputation, falls vorhanden. Ansonsten wird der Trust-Wert verwendet, oder wenn dieser kleiner als die objektive Bewerterreputation ist, der OBRep-Wert[3].

Eine weitere Aufgabe der Metrik ist es, die subjektive und objektive Bewerterreputation zu berechnen. Ebenso wie die Trust-Werte wird auch die Bewerterreputation als Fließkommazahl zwischen 0 und 1 gespeichert, wobei 0 der schlechteste und 1 der beste Wert ist. Kompliziertere Werte und eigene Klassen für den Bewerterreputationswert sind denkbar. Zunächst soll aber eine möglichst einfache Variante implementiert werden. Die subjektive Bewerterreputation wird als Mittelwert der Vergleichswerte gebildet. Der Vergleichswert wird in der Methode *calculateSimilarity* der Klasse *PercentageMeaning* als Wert zwischen -1 und +1 berechnet. Das ist die Differenz der Prozentwerte der eigenen und anderen

[3]Die OBRep ist für den Fall gedacht, dass man den Bekannten gar nicht oder nur weit entfernt kennt und der Trust-Wert deshalb sehr klein oder null ist.

```
1  ageIndex = 0;
2  // für alle Wertungen
3  for( String keyURL : ratingCheckKeyURLs )
4  {
5      // Daten sammeln
6      meaning        = getMeaningOfRating( keyURL );
7      keyHolderURL   = resourceManager.getKeyHolderURL( keyURL );
8      sbrep          = calculateSBRep( keyHolderURL );
9      obrep          = calculateOBRep( keyHolderURL );
10     trust          = pseudonymUrlToTrust.get( keyHolderURL );
11     // Wichtung berechnen
12     weightOfRating = calculateWeight( trust, sbrep, obrep, ageIndex );
13     // Summieren
14     up   += (weightOfRating * meaning.getPercentage());
15     down += weightOfRating;
16     ageIndex++;
17 }
18 // Ergebnis erzeugen
19 result = new PercentageMeaning();
20 result.setPercentage( (int)Math.round( up / down) );
21 result.setRating( getClosestRating( result.getPercentage() ) );
```

Abbildung 5.5: Berechnung des Reputationswertes für einen Inhalt in der *PercentageMetric*.

Wertung, geteilt durch 100. Da die subjektive Bewerterreputation erst bei mehreren Vergleichswerten aussagekräftig ist, wird der berechnete Wert bei weniger als 10 Vergleichen abgeschwächt (Zeile 12).

Die objektive Bewerterreputation wird aus einer Liste von Sternen berechnet. Wiederum soll ein Wert zwischen 0 und 1 herauskommen. Dazu wird jedem Stern ein Wert zugeordnet. Diese Werte werden addiert und durch eine Funktion auf einen Wert zwischen 0 und 1 gebracht. Die Funktion soll bei (0,0) beginnen (0 Sterne -> 0 OBRep) und für unendlich viele Sterne gegen 1 gehen. Eine Funktion mit dieser Eigenschaft ist der Tangens Hyperbolicus, der z. B. auch als Aktivierungsfunktion in neuronalen Netzen verwendet wird. Die Frage ist, wie man den Wert eines Sterns festlegt. Kann man dem Erzeuger des Sterns einen Trust-Wert zuordnen, kann dieser Grundlage für den Wert des Sterns sein. Sterne von unbekannten Erzeugern gibt man entweder einen Minimalwert oder beachtet sie gar nicht. Möglich ist es auch, für den Unbekannten eine OBRep zu berechnen und darüber den Stern zu bewerten. Das sollte davon abhängen, wie schwer es ist, ein Pseudonym beim System anzumelden. Da beim Prototypen noch kein echtes Identitäts-Management-System verwendet wird, kann der Wert eines unbekannten Sternes nur willkürlich festgelegt werden.

Listing 5.6 zeigt die beiden Methoden für die subjektive und objektive Bewerterreputation aus der Klasse *PercentageMetric*. Die Wert-Berechnung der Sterne wurde so modelliert, dass jemand mit 5 Sternen von direkten Bekannten genauso stark gewichtet wird wie ein direkter Bekannter. Um diese OBRep mit Sternen von Unbekannten zu erreichen, muss er etwa 440 Sterne besitzen.

Jeder Nutzer bildet die Wertungen von Bekannten auf eigene Wertungen ab. Dadurch kann man auch die

```
1  public double calculateSBRep( Map<String, Double> contentUrlToSimilarity )
2  {
3      double result, sum = 0;
4      for( Double similarity : contentUrlToSimilarity.values() )
5      {
6          sum += Math.abs( similarity );
7      }
8      result = 1.0 - (sum / contentUrlToSimilarity.size());
9      if( contentUrlToSimilarity.size() < 10 )
10     {
11         // abschwächen bei weniger als 10 Vergleichen
12         result *= contentUrlToSimilarity.size() * 0.1;
13     }
14     return result;
15 }
16
17 public double calculateOBRep( ResourceManager resourceManager, List<JasonStar> stars )
18 {
19     double sum = 0;
20     for( JasonStar star : stars )
21     {
22         // Gibt es einen Trust-Wert für den Sternerzeuger?
23         keyHolderURL = resourceManager.getKeyHolderURL( star.getCheckKeyURL() );
24         trust        = pseudonymUrlToTrust.get( keyHolderURL );
25         // Nein
26         if( isNull( trust ) ) sum += 0.005;
27         // Ja
28         else                  sum += trust.getTrust() * 0.55;
29     }
30     // 0 Stern -> 0, unendlich viel Sterne -> 1
31     retrun Math.tanh( sum * 0.5 );
32 }
```

Abbildung 5.6: Berechnung der Bewerterreputationen für ein Pseudonym in der *PercentageMetric*.

Wertungen der Bekannten der Bekannten auf eigene abbilden. Hierbei können jedoch Konflikte entstehen. Ein und dieselbe Wertung kann zu verschiedenen Bedeutungen kommen. Wenn dieser Fall eintritt, wird die weniger weit entfernte Bedeutung verwendet. Das bedeutet z. B., die Angaben eines direkten Bekannten wiegen schwerer als die eines Bekannten eines Bekannten. Wenn beide Entfernungen gleich sind, werden beide Möglichkeiten in einer Liste gespeichert, und der Nutzer muss entscheiden, welche Bedeutung er der Wertung geben will.

Wie bereits in Kapitel 4.3 erwähnt, kann man nicht so einfach alle Wertungen eines Bekannten finden, um die subjektive Bewerterreputation zu berechnen. Deshalb verwendet die Klasse *Metric* einen Cache, der durch die Klasse *MetricRatingsCache* beschrieben wird. Der Name soll bedeuten, hier werden Wertungen gespeichert, die zu dieser Metrik gehören. Der Cache speichert, welche Pseudonyme welchen Inhalt bewertetet haben, wie die Wartung war und wie man den Inhalt selbst bewertet hat. Gefüllt wird der Cache beim Berechnen der Reputation für einen Inhalt. Alle Wertungen, die man dabei sieht, werden in den Cache gespeichert. Dadurch kann man die subjektive Bewerterreputation für ein Pseudonym leicht berechnen. Listing 5.7 zeigt die entsprechende Methode aus der Klasse *Metric*. Aus den gecacheden Daten wird die Liste der Vergleiche erzeugt und dann die eigentliche Berechnungsfunktion, die von jeder Metrik definiert werden muss, aufgerufen.

```
1  public double calculateSBRep( String pseudonymURL )
2  {
3      // bekannte Inhaltswertungen vom gegebenen Pseudonym
4      List<MetricRatingsCacheEntry> entries = cache.getContentUrlsForPseudonymUrl( pseudonymURL );
5      // Eingabedaten für die SBRep-Berechnung
6      Map<String, Double> contentUrlToSimilarity = new HashMap<String, Double>();
7      for( MetricRatingsCacheEntry entry : entries )
8      {
9          // wie hat das Pseudonym den Inhalt bewertet
10         rating     = entry.getRating();
11         // eigne Wertung
12         ownRating  = entry.getOwnRating();
13         // wenn keine eigene Wertung, berechneten Reputationswert nehmen
14         if( isNull( ownRating ) ) ownRating = entry.getOwnReputationValue();
15         // vergleichen
16         similarity = rating.calculateSimilarity( ownRating );
17         // in Map speichern
18         contentUrlToSimilarity.put( entry.getContentURL(), similarity );
19     }
20     // eigentliche Berechnung starten
21     return calculateSBRep( contentUrlToSimilarity );
22 }
```

Abbildung 5.7: Verwendung des Cache zum Berechnen der subjektiven Bewerterreputation.

Der entwickelte Prototyp enthält eine Visualisierung für die beiden Bewerterreputationen. Die Abbildungen 5.8 und 5.9 zeigen je einen Screenshot. Der Aufbau des Programmfensters wird in Anhang C erläutert. Die Visualisierung der Bewerterreputation zeigt die Anzahl bekannten Wertungen oder Ster-

ne, die Metrik nach der die BRep berechnet wurde, den Wert selber und eine grafische Darstellung der Daten. Abbildung 5.8 zeigt einen Bekannten der die meisten Inhalte eine Stufe besser bewertet hat als man selbst. Einen Inhalt hat er sogar mit null Prozent bewertet, welches man selber mit hundert Prozent bewertet hat. Die Daten stammen aus der Evaluierung, die in Kapitel 5.4 beschrieben wird. Abbildung 5.9 zeigt, wie die Visualisierung der objektiven Bewerterreputation aussehen könnte. Da es hier keine echten Daten gibt, wurden ein Beispiel erzeugt. Gezeigt wird die Anzahl der Sterne, sortiert nach der Entfernung des Sternerzeugers. Die Entfernung ist die Anzahl, über wie viele Personen man mit dem Erzeuger bekannt ist.

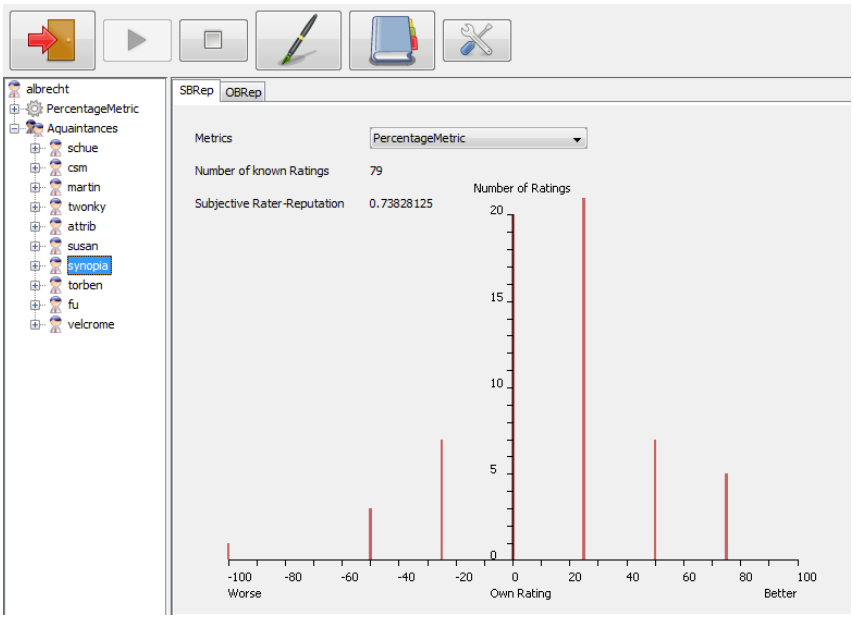

Abbildung 5.8: Darstellung subjektiven Bewerterreputation im Prototypen. Die gezeigten Daten stammen aus der Evaluierung des Systems.

5.3 Sicherheitsanalyse

In Kapitel 4.4 wurden Sicherheitsanforderungen diskutiert, die man beim Entwurf eines Reputationssystems beachten sollte. Eine zweite Möglichkeit, die Sicherheit eines Systems zu diskutieren, ist es, konkrete Angriffe zu besprechen. [CHE] listet eine Reihe von möglichen Angriffen auf Reputationssysteme auf. Von diesen Angriffen werden hier die für Bewerterreputation relevanten diskutiert.

Reputationssysteme sollen das Verhalten von Nutzern beeinflussen. Das bedeutet, wenn man das Reputationssystem manipulieren kann, kann man indirekt das Verhalten der Nutzer beeinflussen. Daraus entsteht ein starkes Motiv für Angriffe auf das Reputationssystem. Meist stehen wahrscheinlich kommerzielle Interessen, die ein bestimmtes Ziel verfolgen, hinter den Angriffen. Man kann der Konkurrenz schaden und sich selbst einen Vorteil schaffen. Denkbar sind auch rein destruktive Angriffe, die das System einfach nur schädigen wollen. Kein richtiger Angriff, aber eventuell auch ein Problem, ist die wissentliche oder unwissentliche falsche Benutzung des Systems. Unfaire Wertungen werden in Arbeiten wie [ZC08] oder [VHP08] untersucht. Motivationen privater Anwender zur falschen Benutzung müssen im Bereich der

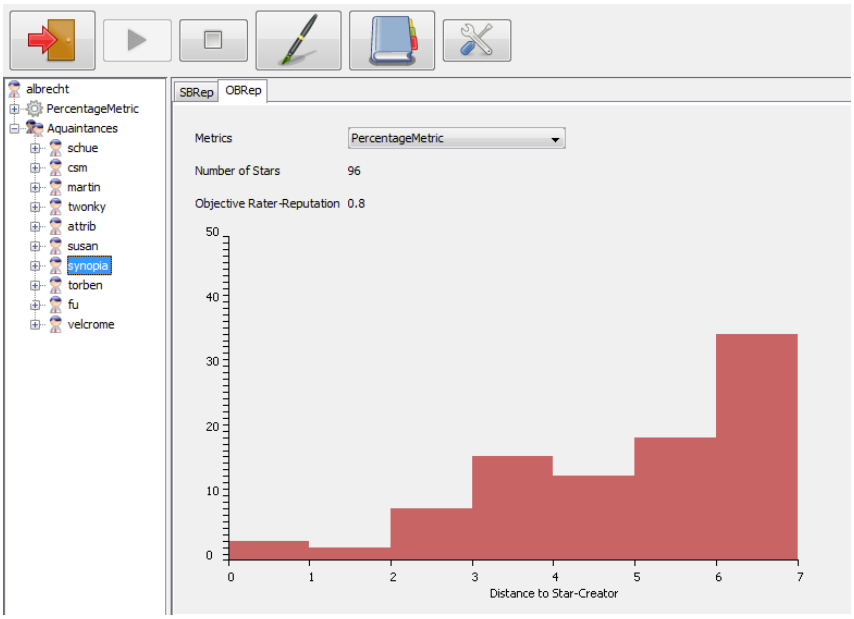

Abbildung 5.9: So könnte man die objektive Bewerterreputation visualisieren. Die gezeigten Daten wurden Beispielhaft erzeugt und nicht aus der Evaluierung.

Psychologie untersucht werden und führen hier zu weit (Einen Einstieg in diese Richtung bietet [bia]).

Folgende Angriffe werden in den weiteren Absätzen diskutiert[4]:

- **Whitewashing**: Zurücksetzen schlechter Bewerterreputation durch neue Identität.
- **Sybil-Attacke**: Erzeugen vieler Pseudonyme und Missbrauch dieser.
- **Ballot Stuffing / Bad Mouthing**: Falsche Wertungen abgeben, um Reputation zu erhöhen oder zu senken.
- **Collusion**: mehrere Nutzer arbeiten zusammen um einen Reputationswert zu beeinflussen.
- **Recommender dishonesty**: Ein Werter ist nicht vertrauenswürdig in seinen Wertungen.
- **Social threats**: Diskriminierung, Herdenverhalten, Unterdrückung von Minderheiten usw.

Da es sich bei der Umsetzung des Systems um einen Prototypen handelt, können einige Angriffe nicht vollständig diskutiert werden. Vor allem das Identitäts-Management-System ist entscheidend für die Sicherheit des Systems. Im Moment wird aber noch kein solches Identitäts-System verwendet und man kann beliebig viele Pseudoyme erzeugen. Sybil-Attacken sind deshalb einfach ausführbar, allerdings durch die Verwendung des sozialen Netzes erschwert, da unbekannte Pseudonyme weniger Einfluss haben. Auf die subjektive Bewerterreputation hat diese Attacke sogar gar keine negativen Auswirkungen, da bei der SBRep nur genau zwei Pseudonyme in die Berechnung einfließen.

Bei der objektiven Bewerterreputation hängt es davon ab, wie die Metrik mit Sternen umgeht, die von Unbekannten erzeugt wurden. Wenn diese nicht beachtet werden, ist der Sybil-Angriff wirkungslos,

[4]Die Namen der Angriffe wurde nicht ins Deutsche übersetzt, da dies keinen Vorteil für die Verständlichkeit bringt, dafür aber Ungenauigkeit hinzufügt.

die OBRep verliert aber wahrscheinlich auch an Aussagekraft. Eine Möglichkeit ist es, die Sterne, die von Unbekannten erzeugt wurden, nur dann zu beachten, wenn der Erzeuger selbst eine Mindestanzahl an Sternen besitzt. Das macht allerdings nur Sinn, wenn man über das Identitäts-Management-System verhindern kann, dass ein Nutzer Sterne an seine eigenen Pseudonyme vergibt[5]. Um dann noch einen erfolgreichen Angriff auf die OBRep auszuführen, muss man neben der Sybil-Attacke auch noch mit anderen Nutzern zusammenarbeiten (Collusion).

Die Möglichkeit einer Whitewashing-Attacke ergibt sich aus der Bewerterreputation die ein neu erstelltes Pseudonym bekommt. Da man bei der objektiven Bewerterreputation bei Null anfängt und sich von dort aus nur verbessern kann, ist diese Angriffsart hier wirkungslos. Das gilt nur, solange man nur positive Bedeutungen von Sternen zulässt. Bei der subjektiven Bewerterreputation hängt es davon ab, ob die Metrik eine gewisse Mindestanzahl an Vergleichsdaten fordert. Die in Listing 5.6 gezeigte Funktion schwächt die SBRep z. B. ab, wenn weniger als 10 Vergleiche bekannt sind. Ein weiteres Erschwernis für diesen Angriff ist es, dass man die eigene SBRep nicht kennt.

Eine Attacke auf die subjektive Bewerterreputation, die man dem Ballot Stuffing zuordnen könnte, sieht folgendermaßen aus: Das entworfene System macht zwar keine Bedeutungen von Wertungen öffentlich, aber die Abbildungen fremder Wertungen auf eigene. Ein Angreifer sieht also, wie sein Opfer andere Wertungen abbildet. Außerdem kann er sehen, mit welchem Schlüssel das Opfer einen Inhalt bewertet. Ohne die Bedeutung kennen zu müssen, kann der Angreifer immer mit dem eigenen Schlüssel werten, der die gleiche Bedeutung hat. Dadurch bekommt er eine hohe subjektive Bewerterreputation beim Opfer. Die Metrik sollte deshalb die Auswirkungen einer hohen SBRep beschränken. Der Angreifer muss Teil des sozialen Netzes sein, was den Aufwand des Angriffs erhöht. Weiterhin muss man diskutieren, ob die von der ersten JASON-Implementierung eingeführte öffentliche Abbildung fremder Wertungen auf eigene notwendig ist. Denkbar ist es auch, diese Abbildung nur an gute Freunde weiterzugeben.

Recommender dishonesty lässt sich von den bisherigen Attacken nicht sauber trennen. Bei der objektiven Bewerterreputation geht der nicht vertrauenswürdige Werter hoffentlich in der Menge der Sterne unter. Bei der SBRep müsste der Angreifer mit System arbeiten, wie es im vorhergehenden Absatz beschrieben wurde. Das soziale Netz und die beiden Bewerterreputationen ermutigen Nutzer hoffentlich zu vertrauenswürdigem Verhalten[6].

Social Threats sind sehr schwierig zu diskutieren. Sie entstehen durch komplexes menschliches Verhalten in Gruppen und müssen eher im Fachbereich Psychologie untersucht werden. Reputationssysteme sollen ihre Nutzer zu besserem Verhalten motivieren. Der implementierte Prototyp hat aber zuerst das Ziel, das technische Konzept, welches in dieser Arbeit entwickelt wurde, zu validieren. Deshalb eignet er sich nur bedingt, um soziale Gefahren zu untersuchen. Der Aufwand übersteigt den Rahmen dieser Arbeit. Fragen dieser Art müssen in zukünftigen interdisziplinären Arbeiten untersucht werden.

Neben den möglichen Angriffen, die sich aus dem System-Entwurf ergeben, hat der Prototyp einige Sicherheitslücken. Das aus der Vorgängerarbeit übernommene PHP-Upload-Script zum Hochladen einer Wertung z. B. enthält keinerlei Sicherheitsprüfungen. Öffentliche Schlüssel werden nicht signiert und könnten deshalb manipuliert werden. Die grösste Sicherheitslücke ist das fehlende Identitäts-Management-System. Der Prototyp zeigt die Funktionsfähigkeit des entworfenen Systems, ist aber auch nur als Proto-

[5] Diese Einschränkung könnte aber dazu führen, dass man Pseudonyme ein und demselben Nutzer zuordnen kann.

[6] Eine gute Erklärung, was man unter vertrauenswürdigem Verhalten verstehen kann, findet man in [MMH02].

typ zu betrachten.

5.4 Diskussion des Ergebnisses und Validierung

Die Erweiterung von JASON um Bewerterreputationen konnte erfolgreich als Prototyp umgesetzt werden. Um die Qualität des Konzeptes und der Umsetzung zu testen, wurde eine Evaluation durchgeführt. Dazu ist ein spezielles Testszenario entwickelt worden. Zunächst soll beschrieben werden, welche Fragen die Evaluation beantworten soll. Danach wird das Testszenario beschrieben.

Die wichtigste Frage ist, ob die Bewerterreputation die eigentliche Reputationsberechnung verbessern konnte. Weitere interessante Fragen sind, ob das Konzept des Systems verständlich ist, es als nützlich eingestuft wird und ob der Befragte das System nutzen würde. Mangelndes Verständnis deutet auf eine schlechte Anleitung, eine schlechte Benutzerführung in der Anwendung oder auf ein schlechte Konzept hin. Wird das System nicht als nützlich eingestuft, muss untersucht werden, ob die verwendete Metrik ungeeignet war oder das Testszenario ungünstig gewählt wurde. Schlechte Akzeptanz und fehlende Bereitschaft, das System zu nutzen, deuten auf zuviel Aufwand bei der Bedienung hin. Das Aufwand-Nutzen-Verhältnis muss dann verbessert werden.

Folgende Hypothesen soll die Validierung festigen:

1. JASON ist ein einfaches, gut bedienbares Reputationssystem mit hohen Sicherheitsansprüchen.
2. Die subjektive Bewerterreputation kann den Reputationswert für Inhalte, bei denen Geschmack eine Rolle spielt, erheblich verbessern.
3. Die objektive Bewerterreputation ermöglicht die Interpretation von Wertungen unbekannter oder wenig bekannter Pseudonyme.

Die subjektive Bewerterreputation wurde für Gruppen entworfen, die häufig gleiche Inhalte bewerten (siehe Beispiel Kinofreunde aus Kapitel 3.1). Für ein Testszenario benötigt man also nur wenige Teilnehmer, aber eine größere Menge an bewertbaren Inhalten. Um es möglichst einfach zu halten, sollen die Inhalte alle aus einem ähnlichen Kontext kommen. Dazu wurden reichlich 100 Bilder auf einer Webseite mit dem JASON-Applet bereit gestellt. Die Bilder werden in einer zufälligen Reihenfolge angezeigt. Wenn alle Testteilnehmer genügend Wertungen abgegeben haben, können für die noch nicht bewerteten Bilder Reputationswerte berechnet werden. Hier kann direkt die Qualität der Reputationswerte, mit Einbeziehung der SBRep und ohne, verglichen werden. Abbildung 5.10 zeigt eines der Testbilder mit dem FrontEnd-Applet darunter. Zum Werten klickt man auf das Logo des Applets. Ein Mausklick auf das Bild hat das nächste Bild zum Bewerten angezeigt.

Für die objektive Bewerterreputation lässt sich nicht so einfach ein Testszenario erzeugen. Die OBRep ist für Pseudonyme entworfen, die im sozialen Netz weit entfernt sind. Es werden also sehr viele Nutzer aus verschiedenen Bereichen benötigt, um ein weit zerstreutes soziales Netz aufzubauen. Der Aufwand ist zu hoch für diese Arbeit. Diese Untersuchung muss entweder anhand eines bestehenden Netzes durchgeführt werden oder schrittweise bei der Weiterentwicklung des Prototypen. Die Evaluierung der OBRep muss sich demnach mit Fragen zu Verständnis und Akzeptanz begnügen.

Der komplette Evaluierungsbogen befindet sich in Anhang D. Er ist in fünf Teile unterteilt:

Abbildung 5.10: Das am besten bewertete Testbild mit angezeigter Bewertung.

1. Allgemeine Angaben zum Befragten
2. Allgemeine Angaben zum JASON-System
3. Fragen zur subjektiven Bewerterreputation
4. Fragen zur objektiven Bewerterreputation
5. Fragen zum Prototypen

Teil 2, 3 und 4 erfragen jeweils: Verständnis, Akzeptanz (Würden sie das System verwenden?) und Nützlichkeit. Teil 5 stellt vor allem Fragen zur Benutzbarkeit des Prototpyen. Die Nützlichkeit der SBRep wurde auch objektiv ausgewertet, indem die Resultate der alten Reputationsberechnung, mit denen der neuen Berechnung verglichen wurden. Dazu wurde der Reputationswert einmal mit Verwendung der SBRep und einmal ohne berechnet. Der Reputationswert wird dann jeweils mit der abgegebenen eigenen Wertung verglichen. Ein besserer Reputationswert hat dabei eine höhere Ähnlichkeit.

Bei der Evaluierung haben zehn Testpersonen teilgenommen. Diese haben insgesamt über 700 Wertungen an reichlich 100 Testinhalten (Bilder) abgegeben. Alle Testteilnehmer kamen aus technischen Studienrichtungen. Vier davon studieren Informatik oder haben ein Informatik-Studium abgeschlossen. Diese Vorbildung der Teilnehmer hat sich positiv auf das Verständnis des Systemes ausgewirkt. Nur wenige hatten Probleme den Prototypen zu starten und zu benutzen. Bei technisch nicht vorgebildeten Nutzern sind hier deutlich größere Probleme zu erwarten.

Abbildung 5.11 zeigt die Ergebnisse der Befragung, die durchgeführt wurde, nachdem die Testteilnehmer den Prototypen verwendet haben. Die Fragen lassen immer drei verschiedene Antwortmöglichkeiten

zu (es konnten aber auch Text-Kommentare abgegeben werden). Grün steht für die bestmögliche Antwort, rot für schlecht und gelb für eine mittlere Einschätzung (die jeweilige genaue Bedeutung der drei möglichen Antworten kann den Fragen in Anhang D entnommen werden). Die Zahlen in den Balken, geben die Anzahl der Antworten dieser Art wieder.

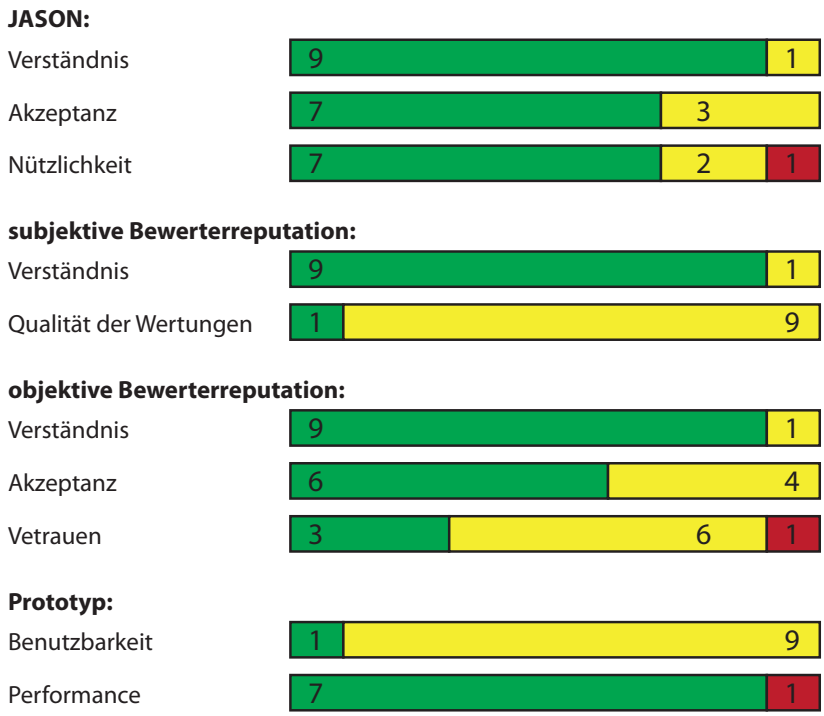

Abbildung 5.11: Ergebnisse der Befragung nach der Evaluierung.

Die allgemeine Befragung zum JASON-System hat sehr positive Ergebnisse gebracht. Das System wurde gut verstanden, was sicherlich an der Vorbildung der Testeilnehmer liegt. Sieben von zehn Befragten halten das System außerdem für nützlich und würden es in einer produktiven Version verwenden. Die eine negative Wertung bei der Nützlichkeit bezog sich auf das verwendete Testszenario. Der Befragte fühlte sich bei der Bewertung der Bilder sehr stark beeinflusst durch die bereits angezeigte Wertung. Diese Problem wird im nächsten Absatz besprochen. Weitere Kommentare sagten z. B., dass man das System in bestehende Plattformen integrieren sollte und dass es nur dann nützlich ist, wenn es von vielen benutzt wird.

Bei der subjektiven Bewerterreputation gibt es keine Frage, ob man sie verwenden würde, da sie automatisch vom System verwendet wird. Die entscheidende Frage ist, ob die berechneten Reputationswerte durch die Verwendung der SBRep genauer wurden. Im nachhinein hat sich gezeigt, dass die Evaluierung diese Frage nicht genau beantworten konnte. Die meisten Befragten gaben an, dass die angezeigten Wertungen nur manchmal ihrer Meinung entsprach. Das Testszenario hat den entworfen Prototypen verwendet. Für eine reine Untersuchung der SBRep hätte ein eigenes Testsystem entwickelt werden müssen und auch bei der Testdurchführung mehr Aufwand betrieben werden (z. B. ein Training mit den Testpersonen durchführen). Die ersten Werter haben noch keine Wertungen der anderen gesehen, während die letzten bereits bis zu zehn Wertungen ihrer Vorgänger gesehen haben. Die ersten Werter konnten also noch keine SBRep verwenden, da es keine anderen Wertungen gab. Ein weiteres Problem waren die Testinhalte.

Sie boten nicht genug Gestaltungstiefe, so dass sich die Werter stark von der angezeigten berechneten Wertungen beeinflusst sahen. Außerdem lies sich bei keinem Werter ein bestimmtes Profil erkennen, welche Bilder er gut bewertet. Das hat sich z. B. an zwei Bildern gezeigt, die fast identisch sind, aber unterschiedlich bewertet wurden. Die Wertung hing wahrscheinlich stark von den vorher gesehen Bildern, der aktuellen Gemütslage und der angezeigten Bild-Reputation ab. Für zukünftige Untersuchungen der subjektiven Bewerterreputation sollten Testinhalte gewählt werden, die eine tiefgründigere Meinung erfordern. Beispiele sind Bilder von zubereiteten Speisen oder Filmplakate. Weiterhin muss genau festgelegt werden, welcher Aspekt bewertet werden soll. Bei Speisen z. B. wie gerne man sie isst und bei Filmplakaten, wie gut man den Film fand.

Die Funktionsweise der SBRep wurde nicht nur per Fragebogen untersucht, sondern auch rechnerisch. Zu jedem Bild, dass ein Testeilnehmer bewertet, kann man die Reputation des Bildes einmal mit Verwendung der SBRep und einmal ohne berechnen. Der berechnete Wert mit Verwendung der SBRep sollte im Mittel näher an der Wertung des Testteilnehmers liegen, als der Wert ohne SBRep. Tatsächlich haben die Daten diese Tendenz gezeigt, allerdings schwächer als erwartet. Das hat möglicherweise zwei Ursachen: Die verwendeten Berechnungsalgorithmen waren willkürlich gewählt und die verwendeten Testinhalte waren, wie bereits angesprochen, ungeeignet. Entscheidend für die Berechnung der SBRep ist die Ähnlichkeitsfunktion zwischen zwei Wertungen. Wahrscheinlich sollte man die Ähnlichkeit besonders hoch ansetzen, wenn in Extrembereichen gleich gewertet wurde. Also z. B. zwei mal sehr gut oder zwei mal sehr schlecht. Leider waren die evaluierten Daten nicht signifikant genug, um diese Hypothese zu festigen. Weiterhin gibt es Hinweise, dass lineare Metriken, bei denen arithmetische Mittelwerte verwendet werden, nicht die besten sind. Bilder mit mehr Wertungen tendieren immer in Richtung einer durchschnittlichen Reputation. Besonders gut bewertete Bilder waren nicht die mit den meisten Wertungen.

Es hat sich gezeigt, dass die Untersuchung über geeignete Wertungsalgorithmen mehr als genug Stoff für eine eigene Arbeit ist. Genauso muss bei dem Testszenario mehr Zeit investiert werden. Für zukünftige Untersuchung des Konzeptes der subjektiven Bewerterreputation bleiben folgende Hinweise festzuhalten:

- Die Testinhalte müssen eine tiefergehende oder gefestigte Meinung abfragen, sonst gibt es zu viele nicht beeinflussbare Störfaktoren. Einfach Beispiele könnten sind Bilder von Speisen oder Filmplakate.
- Es muss genau festgelegt werden, welcher Aspekt bewertet werden soll. JASON lässt eigene Metriken und damit selbst definierte Aspekte zu. Für eine reine Untersuchung der SBRep ist das ungeeignet.
- Es muss ein eigenes Testsystem verwendet werden. Die Testpersonen müssen mit diesem System ein Training durchlaufen, vor den eigentlichen Wertungen.
- Anhand grösserer Mengen solcher Daten sollte es möglich sein, Untersuchungen durchzuführen, wie eine gute Bewertungsmetrik beschaffen sein muss. Durch Optimierungsalgorithmen kann außerdem die Form der Ähnlichkeitsfunktion bestimmt werden.

Die objektive Bewerterreputation wurde nur anhand der Befragung untersucht. Der Prototyp unterstützt bereits das Erstellen und Einfügen von Sternen. Allerdings sind sehr viele Sterne nötig, um eine aussagekräftige OBRep zu berechnen (siehe Kapitel 3.1). Diese Menge an Testpersonen stand zur Evaluierung

leider nicht bereit. Die Befragung konnte jedoch positive Ergebnisse beim Verständnis und Akzeptanz bringen. Fast alle Befragten haben das Konzept der Sterne gut verstanden und würden das System verwenden. Wichtig scheint ein erkennbarer Nutzen bei der Erstellung von Sternen. Vier von zehn Testpersonen gaben an, sie würden das System verwenden, wenn sie einen Nutzen davon haben. Der Aufwand beim Erzeugen und Vergeben von Sternen muss so gering wie möglich sein, da der Nutzen eines einzelnen Sternes für den Stern-Erzeuger nicht spürbar ist. Bei der Frage, ob sie dem berechneten OBRep-Wert vertrauen würden, gaben drei Testpersonen ”Ja” an. Sechs mal wurde angegeben: ”Ja, wenn ich weiß, woher die Sterne kommen”. Eine Testperson hätte kein Vertrauen in diese Art der Berechnung. Hier zeigt sich, dass ein transparenter Vorgang bei der Berechnung der OBRep sehr wichtig ist. Die Nutzer müssen wissen, woher die Sterne kommen und wie diese zur OBRep umgerechnet werden. Die Kommentare zu den OBRep-Fragen hat gezeigt, dass das Vertrauen von den bisherigen Erfahrungen mit Reputationssystemen abhängt. Durch Systeme wie EBay, kennen Internet-Nutzer bereits Systeme, bei denen sie sich gegenseitig bewerten und einen Nutzen davon haben.

Der letzte Abschnitt im Fragebogen enthielt zwei Fragen zur Qualität des Prototypes. Die Benutzbarkeit erhielt eine überwiegend mittlere Bewertung. Besonders die Aufteilung in eine Java-Anwendung und ein Applet haben für Verwirrung gesorgt. Die Performance-Bewertung schnitt hingegen sehr gut ab. Nur eine Testperson hat die Berechnungszeit als zu lang empfunden. Dabei muss erwähnt werden, dass die Zeit der Berechnung sehr stark von der Anzahl der Wertungen abhängt, die geladen werden. Bei steigender Anzahl von Wertungen muss der Ladevorgang optimiert werden (z. B. keine eigene XML-Datei pro Wertung). Weiterhin wurde das fehlende Feedback nach einer abgegebenen Wertung bemängelt. Die wichtigste Schlussfolgerung aus diesem Teil der Befragung ist, dass eine richtig Browserintegration zwingend ist für einen produktiven Einsatz.

Die drei oben genannten Hypothesen konnte die Evaluierung untermauern. Sie können aber noch nicht als gesichert betrachtet werden:

1. Die Befragung konnte die erste Hypothese gut festigen. Das JASON-System wurde von den Testpersonen verstanden und akzeptiert. Bei der Benutzbarkeit gibt es noch Verbesserungspotential.
2. Die subjektive Bewerterreputation verbessert die Berechnung der Reputationswerte. Das Potential dieser Verbesserung konnte nicht ausgelotet werden. Dazu sind tiefergehende Untersuchungen zu den verwendeten Algorithmen zum Vergleichen von Wertungen und zur Berechnung von Reputationen nötig.
3. Die Evaluierung deutet darauf hin, dass die objektive Bewerterreputation im produktiven Einsatz funktionieren würde. Verständnis, Akzeptanz und Vertrauen sind hinreichend gegeben, wenn die Umsetzung transparent und die Benutzung einfach ist.

Die wichtigste Schlussfolgerung für den Prototypen ist, dass eine richtige Browserintegration notwendig ist. Das Applet kann das Konzept validieren, hat aber Schwächen in der Benutzbarkeit (Ladezeiten, Fokuswechsel zwischen Applet und Kern-Anwendung usw.). Weiterhin lassen sich auch unabhängig von der Evaluierung einige Bereiche nennen, in denen der Prototyp und Entwurf noch einmal überarbeitet werden muss. Folgende Punkte sollten von weiterführenden Arbeiten aufgegriffen werden:

- Nutzbarkeit der Anwendung auch ohne FTP-Zugang.
- Keine Vermischung von URL und ID bei Schlüsseln und Pseudonymen.

- Mehr Kontrolle bei der Veröffentlichung der Abbildungen der Wertung.
- Kommunikationsmöglichkeit zwischen verschiedenen Pseudonymen, z. B. um Sterne zu vergeben.
- Applet sollte durch ein Browser-Plugin ersetzt werden.

Viele Verbesserungsmöglichkeiten gibt es bei der grafischen Nutzeroberfläche des Kerns. Auch die Anzeigemöglichkeiten des FrontEnds sollten erweitert werden. Ein Browser-Plugin bietet hier mehr Möglichkeiten und bessere Akzeptanz. Applets werden nur noch wenig verwendet im Web, aber fasst jeder Nutzer kennt Browser-Plugins. Dadurch verliert man etwas von der Plattformunabhängigkeit, gewinnt aber auch sehr viele neue Möglichkeiten.

Abschließend kann gesagt werden, dass das Konzept erfolgreich durch einen Prototypen validiert werden konnte. Dieser Prototyp bietet noch viele Verbesserungsmöglichkeiten, aber auch eine solide Grundlage für weiterführende Arbeiten.

6 Zusammenfassung

Die Bewerterreputation, oder kurz BRep, eines Pseudonyms soll helfen, dessen Wertungen besser einzuschätzen. Dabei wurden zwei Arten von Bewerterreputation eingeführt. Die erste soll aussagen, wie nützlich die Wertungen aus Sicht eines bestimmten Pseudonyms sind. Die andere soll etwas über die allgemeine Qualität der Wertungen aussagen. Erstere BRep ist also die subjektive Sicht einer Person und wird deshalb subjektive Bewerterreputation oder kurz SBRep genannt. Die zweite Variante versucht möglichst objektiv einzuschätzen, wie gut die Wertungen eines Pseudonyms sind. Sie wird dementsprechend objektive Bewerterreputation, oder OBRep genannt.

Die SBRep wird immer aus Sicht einer Person auf ein anderes Pseudonym berechnet. Dazu vergleicht man die Wertungen des Pseudonyms mit den eigenen. Das Ergebnis sagt aus, wie ähnlich das andere Pseudonym wertet. Wertungen von Pseudonymen mit hoher SBRep sollten dann stärker gewichtet werden. Dieses Konzept eignet sich gut für Gruppen die viele gemeinsame Inhalte bewertet haben, da man viele Wertungen vergleichen muss, um eine aussagekräftige SBRep zu erhalten. Weiterhin ist die SBRep gut geeignet für Inhalte, bei denen Geschmack und persönliche Vorlieben eine Rolle spielen.

Die OBRep soll den Fall abdecken, dass man das betreffende Pseudonym nicht oder wenig kennt. In diesem Fall ist es schwer abzuschätzen, wie kompetent und vertrauenswürdig das Pseudonym wertet. Die OBRep soll genau das ausdrücken. Eine objektive Aussage über die Qualität der Wertungen des Pseudonyms. Dazu können sich Pseudonyme gegenseitig bewerten. Die grundlegende Idee dabei ist es, dass unter bestimmten Bedingungen, viele subjektive Meinungen zu einer objektiven Meinung zusammengefasst werden können.

Auf Grundlage das JASON-Konzeptes wurde ein System entworfen, dass beide Arten von Bewerterreputation umsetzt. Dieses System ist als Prototyp implementiert und validiert worden. Die Evaluierung konnte zeigen, dass beide Arten von Bewerterreputation funktionieren. Das gesamte Potential der OBRep und SBRep konnte aber noch nicht ergründet werden. Dazu sind weitere Schritte nötig, die in Kapitel 5.4 beschrieben werden.

Zukünftige Arbeiten sollten folgende Themen bearbeiten:

1. Verbesserung des Prototypen durch Browserintegration.
2. Weitere Evaluierung der Bewerterreputationskonzepte durch konkrete Testszenarien und Wertungsalgorithmen.
3. Erweiterung des JASON-System für dynamische Inhalte.

Der erste Punkt hat sich aus den Nutzertests ergeben. Fast alle Nutzer empfanden die Trennung in Applet und Java-Anwendung als störend. Heutige Internet-Browser können Erweiterungen in Form von Add-ons oder Plugins hinzufügen. Dadurch kann die Usability des Systemes stark verbessert werden. Der zweite Punkt ist ebenfalls eine Schlussfolgerung der Evaluierung. Die entwickelten Konzepte zur Be-

werterreputation müssen an eigenen Testsystemen untersucht werden, da es beim Nutzen des gesamten JASON-Systems zu viele schwer beeinflussbare Faktoren gibt. Im gleichen Zug sollten konkrete Bewertungsmetriken untersucht werden. JASON selbst legt sich nicht auf einen Algorithmus fest. Um aber in Richtung eines produktiven Einsatzes des Systems zu kommen, sind die verwendeteten Algorithmen sehr wichtig. Der letzte Punkt eignet sich für eine zukünftige Diplomarbeit. Die bei JASON verwendeten digitalen Signaturen verlieren ihre Gültigkeit, sobald der entsprechende Inhalt geändert wird. Selbst kleine Änderungen, die die Bedeutung des Inhalts nicht ändern, lassen alle Wertungen ungültig werden. Lösungen für dieses Problem müssen in einer zukünftigen Arbeit gefunden werden.

Literaturverzeichnis

[Bar10] BARTHEL, Stefan: *Erweiterung des Reputationssystems JASON um Autorenreputation*, Technische Universität Dresden, Diplomarbeit, 2010

[bia] *Wikipedia - List of cognitive biases*. http://en.wikipedia.org/wiki/List_of_cognitive_biases

[Boe08] BOENISCH, Vasco: *Krise der Kritik? : was Theaterkritiker denken - und ihre Leser erwarten*. Theater der Zeit, 2008

[CHE] CARRARA, Elisabetta ; HOGBEN, Giles ; ENISA: Reputation-based Systems: a security analysis.

[CKP09] CHO, Jinhyung ; KWON, Kwiseok ; PARK, Yongtae: Q-rater: A collaborative reputation system based on source credibility theory. In: *EXPERT SYSTEMS WITH APPLICATIONS* 36 (2009), MAR, Nr. 2, Part 2, S. 3751–3760. – ISSN 0957–4174

[CPHH02] CLAUSS , Sebastian ; PFITZMANN, Andreas ; HANSEN, Marit ; HERREWEGHEN, Els V.: Privacy-Enhancing Identity Management. In: *The IPTS Report* 67 (2002), September, S. 8–16

[CS01] CHEN, Mao ; SINGH, Jaswinder P.: Computing and using reputations for internet ratings. In: *EC '01: Proceedings of the 3rd ACM conference on Electronic Commerce*. New York, NY, USA : ACM, 2001. – ISBN 1–58113–387–1, S. 154–162

[elo] *Wikipedia - Elo-Zahl*. http://de.wikipedia.org/wiki/Elo-Zahl

[FN03] FUJIMURA, Ko ; NISHIHARA, Takuo: Reputation rating system based on past behavior of evaluators. In: *EC '03: Proceedings of the 4th ACM conference on Electronic commerce*. New York, NY, USA : ACM, 2003. – ISBN 1–58113–679–X, S. 246–247

[GTW09] GIBBS, Michael ; TAPIA, Mikel ; WARZYNSKI, Frederic: Globalization, Superstars, and the Importance of Reputation: Theory & Evidence from the Wine Industry. In: *SSRN eLibrary* (2009)

[Hip09] HIPP, Dietmar: Empfindliche Seele. In: *Spiegel* 26 (2009), S. 51

[JIB07] JOSANG, Audun ; ISMAIL, Roslan ; BOYD, Colin: A survey of trust and reputation systems for online service provision. In: *DECISION SUPPORT SYSTEMS* 43 (2007), MAR, Nr. 2, S. 618–644. – ISSN 0167–9236

[klaa] *Dewey Decimal Classification*. http://www.oclc.org/dewey/

[klab] *Universelle Dezimalklassifikation*. http://www.udcc.org/

[Mal01] MALAGA, Ross A.: Web-Based Reputation Management Systems: Problems and Suggested Solutions. In: *Electronic Commerce Research* 1 (2001), S. 403–417

[MB09] MALIK, Zaki ; BOUGUETTAYA, Athman: RATEWeb: Reputation Assessment for Trust Establishment among Web services. In: *The VLDB Journal* 18 (2009), Nr. 4, S. 885–911. – ISSN 1066–8888

[Mei08] MEICHAU, Markus: *Ein mehrseitig sicheres Reputationssystem im Web 2.0*, Technische Universität Dresden, Diplomarbeit, 2008

[MMH02] MUI, L. ; MOHTASHEMI, M. ; HALBERSTADT, A.: A Computational Model of Trust and Reputation. In: *HICSS '02: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 7*. Washington, DC, USA : IEEE Computer Society, 2002. – ISBN 0–7695–1435–9, S. 188

[PH09] PFITZMANN, Andreas ; HANSEN, Marit: *A terminology for talking about privacy by data minimization:Anonymity, Unlinkability, Undetectability, Unobservability,Pseudonymity, and Identity Management*. http://dud.inf.tu-dresden.de/Anon_Terminology.shtml. Dezember 2009. – v0.32

[SGM09] STEINBRECHER, Sandra ; GROSS, Stephan ; MEICHAU, Markus: Jason: A Scalable Reputation System for the Semantic Web. In: GRITZALIS, Dimitris (Hrsg.) ; LOPEZ, Javier (Hrsg.): *Emerging Challenges for Security, Privacy and Trust. Proceedings of the 24th IFIP International Information Security Conference (IFIP Sec2009)* Bd. 297/2009. Pafos, Cyprus : Springer Boston, Mai 2009, S. 421–431

[SP04] SONG, WH ; PHOHA, VV: Neural network-based reputation model in a distributed system. In: BICHLER, M AND CHUNG, JY (Hrsg.): *CEC 2004: IEEE INTERNATIONAL CONFERENCE ON E-COMMERCE TECHNOLOGY, PROCEEDINGS*, 2004. – IEEE International Conference on E-Commerce Technology (CEC 2004), San Diego, CA, JUL 06-09, 2004. – ISBN 0–7695–2098–7, S. 321–324

[Sur05] SUROWIECKI, James: *The Wisdom of Crowds*. Anchor, August 2005. – ISBN 0385721706

[VHM05] VOSS, Marco ; HEINEMANN, Andreas ; MÜHLHÄUSER, Max: A Privacy Preserving Reputation System for Mobile Information Dissemination Networks. In: *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM'05)*, IEEE, 2005, S. 171–181

[VHP08] VASALOU, Asimina ; HOPFENSITZ, Astrid ; PITT, Jeremy V.: In praise of forgiveness: Ways for repairing trust breakdowns in one-off online interactions. In: *Int. J. Hum.-Comput. Stud.* 66 (2008), Nr. 6, S. 466–480. – ISSN 1071–5819

[Vos04] VOSS, Marco: Privacy Preserving Online Reputation Systems. In: *International Information Security Workshops*, 2004, S. 245–260

[XL04] XIONG, L ; LIU, L: PeerTrust: Supporting reputation-based trust for peer-to-peer electronic

communities. In: *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING* 16 (2004), JUL, Nr. 7, S. 843–857. – ISSN 1041–4347

[ZC08] ZHANG, Jie ; COHEN, Robin: Evaluating the trustworthiness of advice about seller agents in e-marketplaces: A personalized approach. In: *ELECTRONIC COMMERCE RESEARCH AND APPLICATIONS* 7 (2008), FAL, Nr. 3, S. 330–340. – 8th International Conference on Electronic Commerce (ICEC 2006), Fredericton, CANADA, AUG 13-16, 2006. – ISSN 1567–4223

[ZG07] ZIEGLER, Cai-Nicolas ; GOLBECK, Jennifer: Investigating interactions of trust and interest similarity. In: *Decis. Support Syst.* 43 (2007), Nr. 2, S. 460–475. – ISSN 0167–9236

A Nachrichten zwischen Client und Server

Nachrichten werden vom *FrontEnd* zum *JasonCore* geschickt, woraufhin dieser eine Antwort sendet. Sie bestehen aus einem Prefix, gefolgt von Parametern, die wiederum durch festgelegte Zeichenketten getrennt werden. Der Kern kodiert auf die gleiche Art eine Antwortnachricht. Diese beginnt entweder mit **OK** oder **NO**, gefolgt von den Antwort-Parametern. Viele der Nachrichten werden vom aktuellen Prototypen nicht mehr verwendet. Um kompatibel zum ersten Prototypen zu bleiben, werden sie aber trotzdem unterstützt. Im Folgenden sind alle Nachrichten aufgelistet und kurz beschrieben:

GETFORURL*contentUrl*_RDF_*rdfUrl* Abfrage des Reputationswertes für den Inhalt unter der *contentUrl*, dessen RDF-Dokument unter der *rdfUrl* zu finden ist.

Der erste Prototyp hat die berechneten Reputationswerte gespeichert. Mit dieser Nachricht wurden diese gespeicherten Werte abgefragt. Aktuell wird die Reputation nicht gespeichert, sondern immer neu berechnet, da die Berechnung nicht sehr lange dauert.

NO*timeWeigthTillFactor* Rückgabe des maximalen Alterindex timeWeigthTillFactor (int)

GETKEYB64*keyUrl* Abfrage des öffentlichen Prüfschlüssels, der unter der URL *keyUrl* zugänglich ist. Mit dieser Nachricht fragt das *FrontEnd* die Schlüssel ab, um zu überprüfen, ob die Wertungen echt sind.

OK*keyBase64Value* Rückgabe der Schlüsseldaten keyBase64Value (Base64-String).

NO Schlüssel nicht vorhanden oder nicht erreichbar.

GETKEYFAK*keyUrl* Abfrage des Transitionsfaktors des Schlüssel der unter der URL *keyUrl* zugänglich ist.

OK*transitionLengthFactor* Rückgabe des Transitionsfaktors *transitionLengthFactor* (int).

NO Schlüssel nicht vorhanden oder nicht erreichbar.

GETKEYNAM*keyUrl* Abfrage des Schlüsselinhabers des Schlüssel der unter der URL *keyUrl* zugänglich ist.

OK*keyHolderName* Rückgabe des Namens des Schlüsselinhabers *keyHolderName* (String).

NO Schlüssel nicht vorhanden oder nicht erreichbar.

GETKEYNTV*keyUrl* Abfrage des Schlüsselinhabers, des Transitionsfaktors und des numerischen Wertes des Schlüssel der unter der URL *keyUrl* zugänglich ist.

Der numerische Wert wurde im ersten Prototypen zum Berechnen der Wertung verwendet. Im aktuellen Prototypen hat er keine festgelegte Bedeutung mehr. Dadurch verliert diese Nachricht an Bedeutung.

OK*keyHolderName*_FAK_*transitionLengthFactor*_VAL_*keyValue* Rückgabe des Namens des Schlüsselinhabers *keyHolderName* (String), des Transitionsfaktors *transitionLengthFactor* (int) und des numerischen Schlüsselwertes *keyValue* (int).

NOIS AUTHOR Schlüssel ist ein Autorenschlüssel und deshalb der numerische Schlüsselwert *keyValue* nicht bestimmbar.

NO Schlüssel nicht vorhanden oder nicht erreichbar.

GETKEYVAL*keyUrl* Abfrage des numerischen Wertes des Schlüssel der unter der URL *keyUrl* zugänglich ist.

Der numerische Wert wurde im ersten Prototypen zum Berechnen der Wertung verwendet. Im aktuellen Prototypen hat er keine festgelegte Bedeutung mehr. Dadurch verliert diese Nachricht an Bedeutung.

OK*keyValue* Rückgabe des numerischen Schlüsselwertes keyValue (int).

NOIS AUTHOR Schlüssel ist ein Autorenschlüssel und deshalb der numerische Schlüsselwert *keyValue* nicht bestimmbar.

NO Schlüssel nicht vorhanden oder nicht erreichbar.

GETOWNNAM Abfrage des eigenen Pseudonym-Namens des Nutzers.

OK*name* Rückgabe des Pseudonym-Namens name (String)

NO Nutzer ist nicht eingeloggt.

SETAUTFOR*contentUrl*_RPL_*uploadUrl* Auslösen des Erstellens einer Autorensignatur für den unter *contentUrl* erreichbaren Inhaltes und Upload der Signatur mittels des unter *uploadUrl* erreichbaren PHP-Skriptes.

OK Signaturerstellung eingeleitet.

SETFORURL*contentUrl*_RDF_*rdfUrl*_NUM_*value* Setzen des Reputationswertes für den Inhalt *contentUrl*, dessen RDF-Dokument unter *rdfUrl* zu finden ist auf den numerischen Wert *value* (long).

Wird im aktuellen Prototypen nicht mehr verwendet, da die Berechnung der Reputationswertes nicht mehr im *FrontEnd* erfolg.

OK*closestValueName* Rückgabe des Namens des *value* nächstgelegenen Reputationswertes *clostestValueName* (String)

NO Kein Name für einen Reputationswert und damit auch kein nächstgelegener vorhanden.

SETSIGFOR*contentUrl*_RPL_*uploadUrl* Auslösen des Erstellens einer Wertung für den unter *contentUrl* erreichbaren Inhaltes und Upload der Signatur mittels des unter *uploadUrl* erreichbaren PHP-Skriptes.

OK Wertungserstellung eingeleitet.

GETFORRAT*contentUrl*_RDF_*rdfUrl*_IDS_*keyIDs* Auslösen des Berechnens eines Reputationswertes für den unter *contentUrl* erreichbaren Inhaltes, dessen RDF-Dokument unter *rdfUrl* zu finden ist. *keyIDs* besteht aus den IDs der Schlüssel der Wertungen des Inhaltes.

OK*clostestValueName* Rückgabe des Namens des *value* nächstgelegenen Reputationswertes *clostestValueName* (String)

NO Kein Name für einen Reputationswert und damit auch kein nächstgelegener vorhanden.

B JasonCore API

Die Klasse *JasonCore* stellt ein API zur Verfügung, die von *JasonCoreGUI* genutzt wird. Im Folgenden werden alle Methoden dieser API aufgelistet:

`boolean setWorkingDirectory( String dirName )`

Setzt das *Working-Directory* aus dem die .jason-Dateien geladen werden.

Parameters:

`dirName` - Name des Verzeichnis.

Returns:

Das Verzeichnis kann nur geändert werden, wenn kein Nutzer eingeloggt ist. In diesem Fall wird *false* zurückgegeben. Ansonsten *true*.

`boolean newUser( String pseudonymName, ... )`

Legt einen neuen Nutzer an. Dabei wird auch eine neue .jason-Datei erzeugt.

Parameters:

`pseudonymName` - Name des Pseudonyms des neuen Nutzers.

`password` - Passwort

`preferences` - Einstellungen. `null` wenn keine Einstellungen gesetzt werden sollen.

`ftp` - FTP-Verbindungsdaten. `null` wenn keine Einstellungen gesetzt werden sollen.

Returns:

`true` wenn Nutzer angelegt werden konnte, ansonsten `false`.

`boolean login( String pseudonymName, String password )`

Einloggen mit den gegeben Daten.

Parameters:

`pseudonymName` - Pseudonym-Name. `password` - Nutzerpasswort.

Returns:

`true` wenn Nutzer existiert und Passwort richtig.

`boolean logout()`

Ausloggen. Server muss gestoppt sein.

Returns:

`true` wenn ein Nutzer eingeloggt war, und der Server nicht lief.

`boolean startServer()`

Server starten. Der Server empfängt Nachrichten vom *FrontEnd*, führt diese auf dem *JasonCore*

aus und sendet eine Antwort an das *FronEnd*

Returns:

`true` wenn der Server gestartet werden konnte. (Läuft noch nicht und ein Nutzer ist eingeloggt.)

`boolean stopServer()`

Server stoppen. Der Server empfängt Nachrichten vom *FrontEnd*, führt diese auf dem *JasonCore* aus und sendet eine Antwort an das *FronEnd*

Returns:

`true` wenn der Server gestoppt werden konnte. (Er lief vorher.)

`boolean addAcquaintance( String url, LoadingProgressTracker tracker )`

Lädt das RepFile eines Bekannten und seine Wertungsschlüssel aus dem Web und fügt es in eigene Liste der Bekannten ein.

Parameters:

`url` - URL zum RepFile.

`tracker` - Eine Klasse zum Loggen des Lade-Fortschritts oder `null`.

Returns:

`true` wenn der das RepFile des Bekannten geladen werden konnte.

`String newMetric( String name )`

Erzeugt eine neue Metrik. Der übergebene Name gibt an, welche Art von Metrik erzeugt werden soll. Zurückgegeben wird der Name der erzeugten Metrik (Name der Klasse, Name der Instanz).

Parameters:

`name` - Name der Metrik wie er in der *MetricDescription* steht.

Returns:

Name der erzeugten Metrik. Ist meist der Name der übergeben wurde, mit einem angehangenen Index, falls es mehrere Metriken des gleichen Typs gibt. Dieser Name ist gleichzeitig der Verzeichnis-Name in dem die Daten der Metrik gespeichert werden.

`boolean deleteMetric( String metricDirName )`

Löscht die angegebene Metrik und alle Wertungen die zu ihr gehören.

Parameters:

`metricDirName` - Name der Metrik wie er beim Erzeugen durch `newMetric` zurückgegeben wurde.

Returns:

`true` wenn erfolgreich.

`MeaningOfRating newRating( String name, String metricName )`

Erzeugt eine neue Wertung mit dem gegebenen Namen innerhalb der gegenen Metrik. Es wird die Wertung mit ihren Schlüsseln und die Bedeutung erzeugt.

Parameters:

`name` - Name der Wertung.

`metricName` - Name der Metrik wie er beim Erzeugen durch `newMetric` zurückgegeben wurde.

Returns:

Die erzeugte Bedeutung und Wertung.

`void deleteRating( String name )`

Löscht die gegeben Wertung und ihre Bedeutung.

Parameters:

`name` - Name der Wertung die gelöscht werden soll.

Returns:

`true` wenn es die Wertung gab und sie gelöscht wurde.

`boolean setMeaning( Rating own, String foreignRatingURL, ... )`

Setzt die Bedeutung einer fremden Wertung auf die einer eigenen. Falls die fremde Wertung noch nicht bekannt ist, wird sie aus dem Web geladen.

Parameters:

`own` - Eigene Wertung.

`foreignRatingURL` - URL zur Wertung des anderen. Besteht aus der RepFile-URL und dem Namen der Wertung.

`tracker` - Eine Klasse zum Loggen des Lade-Fortschritts oder `null`.

Returns:

`true` wenn die Bedeutung der Wertung gesetzt werden konnte.

`ContentRDF newRDF()`

Erzeugt ein neues RDF-Dokument. Im Moment wird das Dokument nicht automatisch ins Web hochgeladen.

Returns:

Das erzeugte RDF-Dokument.

`void addContentToRDF( ContentRDF rdf, String url )`

Fügt einen Inhalts-Eintrag in ein RDF-Dokument hinzu.

Parameters:

`rdf` - Das RDF-Dokument.

`url` - Inhalts-URL die in das RDF-Dokument eingefügt werden soll.

`JasonContentSignature rateContent( String contentURL, String ratingName )`

Erstellt eine Wertung für den gegebenen Inhalt.

Parameters:

`contentURL` - Inhalt, der bewertet werden soll.

`ratingName` - Name der Wertung, mit der er bewertet werden soll.

Returns:

Die Wertung.

`boolean uploadSignature( String replyToURL, JasonContentSignature signature )`

Läd eine Wertung ins Web.

Parameters:

`replyToURL` - URL zum PHP-Skript, durch das die Wertung in ein RDF-Dokument eingetragen wird und die Signatur-Datei erzeugt wird.

`signature` - Die Wertung.

Returns:

`true` wenn die Wertung erfolgreich hochgeladen wurde.

`JasonContentSignature stateAuthorship( String contentURL )`

Erzeugt eine Autorenschaft-Signature für den gegeben Inhalt. Das ist der gleiche Vorgang wie das Bewerten eines Inhalts, nur dass der Autorenschlüssel anstatt eines Wertungsschlüssels verwendet wird.

Parameters:

`contentURL` - Der Inhalt, für den die Signatur erzeugt werden soll.

Returns:

Die Signatur.

`boolean uploadUserData()`

Lädt das eigene RepFile ins Web.

Returns:

`true` wenn erfolgreich.

`boolean savePreferences()`

Speichert die Einstellungen in die .jason-Datei.

Returns:

`true` wenn erfolgreich.

`boolean saveUserData()`

Speichert das eigene RepFile und die Einstellungen in die .jason-Datei.

Returns:

`true` wenn erfolgreich.

`String getCurrentUserName()`

Liefert den Pseudonym-Namen des aktuell eingeloggten Nutzers.

Returns:

Der Name oder `null` wenn kein Nutzer eingeloggt.

C Kurzanleitung zum Prototypen

Der Prototyp besteht aus einer Java-Anwendung und einem Applet zum Anzeigen der Reputationswerte. Um ihn zu benutzen, muss man zunächst die Java-Anwendung starten. Die Klasse *JasonCoreGUI* enthält die *main*-Methode. Abbildung C.1 zeigt das Programmfenster. Oben befindet sich der Tool-Bar. Links ein Baum, der alle wichtigen Objekte enthält, und rechts der Arbeitsbereich, auf dem das aktuell selektierte Objekt angezeigt wird und bearbeitet werden kann. Der Tool-Bar enthält von links nach rechts folgende Buttons:

- **Login / Logout**, hier kann auch ein neuer Nutzer angelegt werden.
- **Server starten**.
- **Server stoppen**.
- **Inhalt bewerten**, wird normalerweise durch das Applet ausgelöst, kann aber durch diesen Button auch manuel geschehen.
- **Log-Fenster anzeigen**.
- **Einstellungen anzeigen**.

Da es sich um einen Prototypen handelt, ist das Log-Fenster besonders wichtig. Viele Informationen werden nur dort angezeigt, die in einer produktiven Version in die GUI integriert werden sollten.

Wenn man das Programm-Fenster schließt, wird es nicht beendet, sondern läuft im Hintergrund weiter. Es wird ein Tray-Icon angezeigt. Durch einen Doppelklick kann man das Programmfenster wieder anzeigen. Durch einen Rechtsklick wird ein Popup-Menu angezeigt, mit dem man das Programm beenden kann.

Der Angezeigte Baum enthält folgende Objekte:

Pseudonym	Pseudonym-Name des aktuell eingeloggten Nutzers.
Metriken	Alle Metriken die sich der Nutzer anlegt.
Wertungen	Wertungen die zu einer Metrik gehören.
Fremdwertungen	Fremde Wertungen mit der gleichen Bedeutung.
Acquaintances	Bekannte des Nutzers.
Pseudonyme	Pseudonym-Namen der Bekannten.
Wertungen	Namen der Wertungen des Bekannten.
Fremdwertungen	Wertungen anderer Pseudonyme mit gleicher Bedeutung.
Acquaintances	Bekannte der Bekannten.

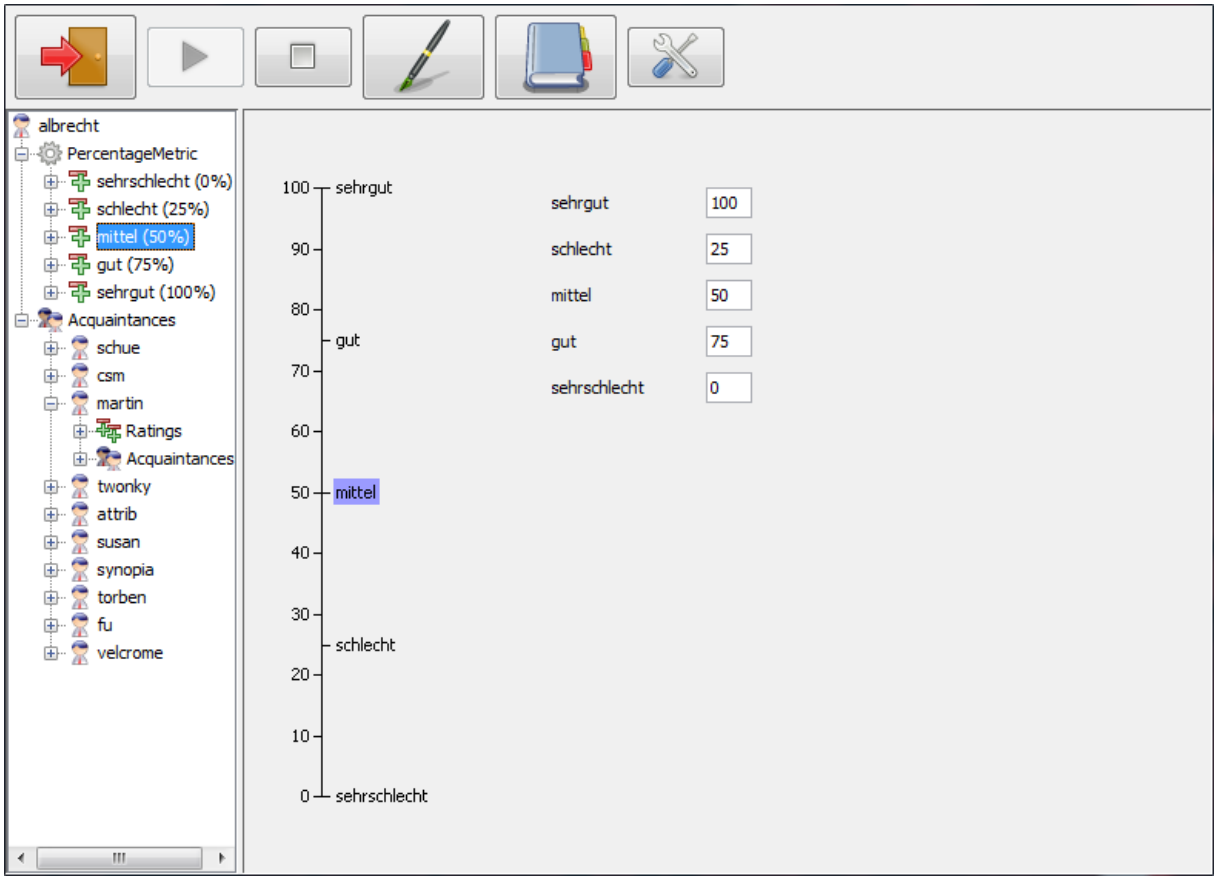

Abbildung C.1: GUI des Kerns. Zu sehen ist der ToolBar, Baum und Arbeitsbereich. Die im Baum selektierte Wertung ”mittel” wird im Arbeitsbereich angezeigt, und kann dort bearbeitet werden.

Durch einen Rechtsklick auf ein Objekt im Baum werden, in einem Popup-Menu, die möglichen Aktionen auf diesem Objekt angezeigt. Tabelle C.1 listet diese Aktionen auf.

Um das System nutzen zu können, benötigt man einen Webspace mit FTP-Zugang. Die öffentlichen eigenen Daten werden über den FTP-Zugang hochgeladen. Danach müssen sie über eine URL erreichbar sein, damit andere Nutzer die öffentlichen Schlüssel sehen können. Die Notwendigkeit eines FTP-Zugangs soll in zukünftigen Versionen abgeschafft werden. Im Moment muss man beim Anlegen eines neuen Nutzers die Zugangsdaten angeben. Abbildung C.2 zeigt den entsprechenden Dialog. Beispielhaft sind die Daten eingetragen, die man verwenden kann, wenn man einen lokalen FTP- und Web-Server zum testen verwendet (z. B. XAMPP). Die Server-URL wird verwendet, um die URL zu erzeugen, nachdem die Anwendung Daten hochgeladen hat. Wird also z. B. ein Pseudonym mit dem Namen ”Bob” erzeugt, ist seine URL dann *http://127.0.0.1/Bob.rep.xml*, wenn man die Daten aus Abbildung C.2 verwendet.

Um das Reputationssytem für einen Inhalt zu nutzen, muss das FrontEnd-Applet auf einer Web-Seite eingebunden werden. Dazu befindet sich im Release des Programms eine signiert .jar-Datei. Abbildung

Objekt	**Aktion**	**Beschreibung**
eigenes Pseudonym	New Metric	Neue Metrik anlegen
	New Acquaintance	Bekannten hinzufügen
	Add Star	Stern in die eigene Sternliste einfügen
Metrik	New Rating	Neue Wertung innherhalb der Metrik anlegen
	Delete	Metrik und alle enthaltenen Wertungen löschen
Wertung	Delete	Wertung löschen
Acquaintances	New Acquaintance	Neuen Bekannten hinzufügen. Laden seiner Daten aus dem Web.
Fremdes Pseudnoym	Create Star	Erzeugt einen Stern für dieses Pseudonym. Der Stern wird im Log-Fenster ausgegeben.
Fremde Wertung	Set Meaning	Die fremde Wertung einer eigenen Zuordnen.

Tabelle C.1: Mögliche Aktionen auf Objekten im Baum.

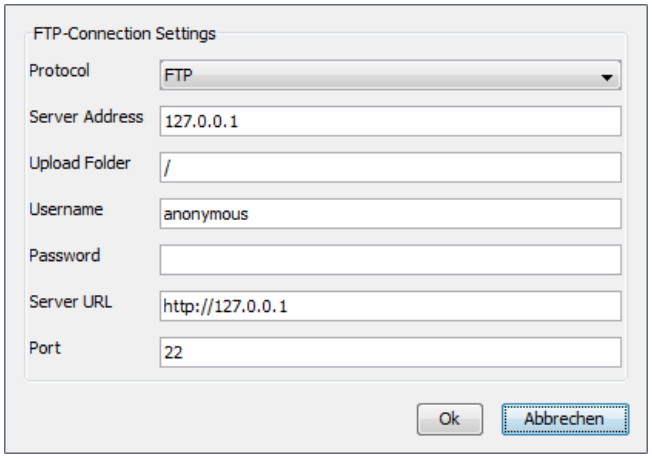

Abbildung C.2: Dialog für die Einstellungen des FTP-Zugangs.

C.3 zeigt eine HTML-Beispiel, wie man das Applet einbinden kann. Die Nutzung eines Applets wurde im ersten Prototypen eingeführt und ist für Zwecke des Prototypen ausreichend. Zukünftige Versionen sollten eine direkte Browserintegration über Plugins entwickeln. Abbildung C.4 zeigt ein Beispiel aus der Evaluierung. Das Applet wird unter einem Bild angezeigt. Zum Abgeben einer Wertung klickt man auf das Logo links im Applet. Daraufhin öffnet sich ein Wertungsdialog. Das ist der gleiche Dialog, der geöffnet wird, wenn man auf den vierten Button von links im Toolbar der Kern-GUI klickt. Abbildung C.5 zeigt diesen Dialog. Der Wert für den Inhalt und der Link, unter dem man die Wertung hochladen kann, sind bereits ausgefüllt. Diese Information schickt das Applet an den Kern, zusammen mit der Aufforderung, eine Wertung zu erstellen.

```
<html><head><title>Fotobewertung</title></head><body>
  <a href=""><img id="pic" src="http://127.0.0.1/images/129086694450884733.jpg" /></a><br />
  <APPLET archive="jason2.jar" code="tud.diplom.albrecht.jason2.frontend.applet.JasonApplet"
          width=160 height=50>
   <PARAM NAME="content" VALUE="http://127.0.0.1/images/129086694450884733.jpg">
   <PARAM NAME="RDF" VALUE="http://127.0.0.1/images/129086694450884733.jpg.rdf.xml">
   <PARAM NAME="replyTo" VALUE="http://127.0.0.1/jason_upload.php">
  </APPLET>
  <p>Click on image for next one.</p>
</body></html>
```

Abbildung C.3: Einbindung des FrontEnd-Applets auf einer Web-Seite.

Abbildung C.4: FrontEnd-Applet mit angezeigter Reputation. Der Bewertete Inhalt ist in diesem Fall ein Bild.

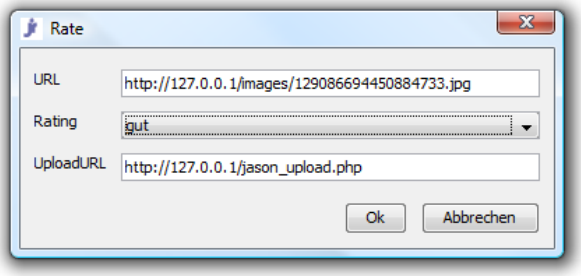

Abbildung C.5: Dialog zum Werten eines Inhaltes.

D Evaluierungsbogen

Folgender Fragebogen wurde bei der Evaluierung verwendet:

Allgemeine Angaben zur Person

Geschlecht:

- □ Weiblich
- □ Männlich

Tätigkeit / Studiengang:

Fragen zu JASON

Ist das Prinzip von JASON verständlich? (Wertung = Signatur, Bedeutungsweitergabe)

- □ ja
- □ weiß nicht
- □ nein

Würden sie das System bei vertretbaren Aufwand nutzen?

- □ ja
- □ vielleicht
- □ nein

Halten sie das System für Nützlich?

- □ ja
- □ teilweise
- □ nein

Fragen zur subjektiven Bewerterreputation

Haben sie das Konzept der subjektiven Bewerterreputation verstanden (ähnliche Wertungen ↔ hohe SBRep)?

- □ ja
- □ weiß nicht
- □ nein

Entsprachen die gezeigten Wertungen der Bilder ungefähr ihrer Meinung?

- □ oft
- □ manchmal
- □ selten

Fragen zur objektiven Bewerterreputation

Haben sie das Konzept der objektiven Bewerterreputation verstanden (gegenseitiges Bewerten durch Sterne)?

- □ ja
- □ weiß nicht
- □ nein

Würden sie das System nutzen, indem sie Sterne an andere Nutzer vergeben, die sie für Kompetent halten?

- □ ja
- □ nur wenn ich was davon habe
- □ nein

Würden sie einer aus den Sternen berechneten Bewerterreputation vertrauen?

- □ ja
- □ nur wenn ich weiß, woher die Sterne kommen
- □ nein

Fragen zum Prototypen

Bewerten sie die Benutzbarkeit des Prototype:

- □ gut
- □ Konzept erkennbar, muss aber noch verbessert werden
- □ schlecht

Die Zeit zum Berechnen der Wertungen war:

- □ kein Problem
- □ erträglich
- □ zu lang

Sonstige Kommentare

E Beispiele für Daten als XML-Datei

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns"
         xmlns:jason="http://jason.cubeweb.org/JasonSchema">
  <rdf:Description rdf:about="http://127.0.0.1/wtfm.jpg">
    <jason:isAuthorSignature>false</jason:isAuthorSignature>
    <jason:checkWith>http://127.0.0.1/oPjrNA.key.xml</jason:checkWith>
    <jason:signatureValue>YWZGZ4q_FFuK2wUJbmY2q58Xq2PUmeQaeO0Zusdx8100YCCJMuWHLbSNfk1HbuEHd5OQ5
    6CEhjzRmTIBg285wyZa5dyOGdyfqSvy4thgRn_baH0aRbrDHYIbkITBFDYtsVHnV2Am7zwc_r-0p8WK32zke9zOaWJ2
    N8RgX5tlE9o=</jason:signatureValue>
  </rdf:Description>
</rdf:RDF>
```

Abbildung E.1: Wertung eines Inhaltes.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns"
         xmlns:jason="http://jason.cubeweb.org/JasonSchema">
  <rdf:Description rdf:about="http://127.0.0.1/wtfm.jpg">
    <jason:bSignatures>
      <rdf:Alt>
        <jason:signature>http://127.0.0.1/VUcC9g.signature.xml</jason:signature>
        <jason:signature>http://127.0.0.1/3PGc-A.signature.xml</jason:signature>
        <jason:signature>http://127.0.0.1/ItrxWA.signature.xml</jason:signature>
        <jason:signature>http://127.0.0.1/SKnrZw.signature.xml</jason:signature>
        <jason:signature>http://127.0.0.1/hGsckg.signature.xml</jason:signature>
      </rdf:Alt>
    </jason:bSignatures>
  </rdf:Description>
</rdf:RDF>
```

Abbildung E.2: RDF-Dokument eines Inhaltes mit 5 eingetragenen Wertungen.

[]

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns"
         xmlns:jason="http://jason.cubeweb.org/JasonSchema">
  <rdf:Description rdf:ID="author">
    <jason:statedBy>
      <rdf:Alt>
        <rdf:li>http://127.0.0.1/reBfJw.key.xml</rdf:li>
      </rdf:Alt>
    </jason:statedBy>
  </rdf:Description>
  <rdf:Description rdf:ID="schlecht">
    <jason:numValue>0</jason:numValue>
    <jason:statedBy>
      <rdf:Alt>
        <rdf:li>http://127.0.0.1/S-jW9A.key.xml</rdf:li>
        <rdf:li>http://127.0.0.1/csm.rep.xml#mittel</rdf:li>
      </rdf:Alt>
    </jason:statedBy>
  </rdf:Description>
  <rdf:Description rdf:ID="ok">
    <jason:numValue>50</jason:numValue>
      <jason:statedBy>
        <rdf:Alt>
          <rdf:li>http://127.0.0.1/6-VtmQ.key.xml</rdf:li>
          <rdf:li>http://127.0.0.1/csm.rep.xml#gut</rdf:li>
        </rdf:Alt>
      </jason:statedBy>
    </rdf:Description>
  <rdf:Description rdf:ID="super">
    <jason:numValue>100</jason:numValue>
      <jason:statedBy>
        <rdf:Alt>
          <rdf:li>http://127.0.0.1/p8p8cw.key.xml</rdf:li>
          <rdf:li>http://127.0.0.1/csm.rep.xml#sehrgut</rdf:li>
        </rdf:Alt>
      </jason:statedBy>
    </rdf:Description>
  <rdf:Description rdf:ID="__Star__">
    <jason:star jason:origin="http://127.0.0.1/MSoMeA.key.xml">
      qggTwliIsclT0Zf5L8syqeNQo8eCe91deUu5ZBruA6MEqHPGpqiawxbyJlw82XnD-LwDEP18f8LCoSa5c4myYlfZ
      wx0AwE_eZrd7mb4ZsqIKPQFhi9dVz9_6EDzW4jr1kr5xvRS-ABBO2y-lVO0dK7kDkkeBt7ZLJhMH_PLfo9E=
    </jason:star>
  </rdf:Description>
</rdf:RDF>
```

Abbildung E.3: RepFile eines Nutzers mit 3 Wertungen und einem Stern.

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns"
3           xmlns:jason="http://jason.cubeweb.org/JasonSchema">
4    <rdf:Description rdf:ID="checkKey">
5      <jason:keyID>1R51xA</jason:keyID>
6      <jason:keyValue>MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCejf-PruynNBe_i12u82s334XX-WdaL_-gNVT
7      MMz-rEZtGx7tDUvgJBud_uQNgIayslYVuCUtEInlIAf0vQ-_4iue_BOetURfaXeEQA-0XcAOPOF94WDofpxFb0KM2gL
8      8uVYPoAHJ4L4hdwnEAPQVT4h3ZMwa1Ui4JKXaHT2h6CwIDAQAB</jason:keyValue>
9    </rdf:Description>
10 </rdf:RDF>
```

Abbildung E.4: Ein öffentlicher Prüfschlüssel einer Wertung.