TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Systemarchitektur Lehrstuhl Rechnernetze

DIPLOMARBEIT

Modularisierung einer wiederverwendbaren Crowdsourcing Infrastruktur am Beispiel der prokurierten Client-Server-Kommunikation des SANE

Name: Duoyi Zhang

Matrikelnummer: 3560153

Institut für Systemarchitektur Lehrstuhl Rechnernetze

Fakultät Informatik an TU Dresden

Betreuer: Dipl.-Inf. Tenshi Hara

Dr.-Ing. Thomas Springer

Termin der Abgabe: Dresden, 02,04,2015

Aufgabenstellung

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Systemarchitektur Lehrstuhl Rechnernetze

AUFGABENSTELLUNG FÜR DIE DIPLOMARBEIT

THEMA: Modularisierung einer wiederverwendbaren Crowdsourcing Infrastruktur am Beispiel der prokurierten Client-Server-Kommunikation des SANE

Name, Vorname:	Zhang, Duoyi	Studiengang:	Informatik (Diplom)
Matrikel-Nummer:	3560153	Projekt/Schwerpunkt:	MapBiquitous/Mobile
Betreuer:	Dipl.-Inf. Tenshi Hara	Zweiter Betreuer:	Dr.-Ing. Thomas Springer
Beginn am:	02.07.2014	Einzureichen bis:	02.01.2015

ZIELSTELLUNG

Im Rahmen des MapBiquitous-Projektes wurde für die Organisation von mobilen Crowdsourcing-Nutzern und den anfallenden Daten eine dezentrale, verteilte Crowdsourcing-Proxy-Struktur entwickelt und implementiert. Aus der vorhandenen Struktur heraus ist eine Wiederverwendbarkeit des Crowdsourcing-Systems über MapBiquitous hinaus erstrebenswert.

Ziel dieser Diplomarbeit ist, die vorhandenen Client- und Serverimplementierungen in der Art zu modularisieren, dass eine einfach wiederverwendbare Crowdsourcing-Infrastruktur entsteht, die das einfach Verteilen und Einbinden neuer Crowdsourcing-Derivate möglichst intuitiv und mit wenig Konfigurations-/Implementierungsaufwand ermöglicht.

Zu untersuchen ist im Rahmen der so abgegrenzten Zielstellung, wie die Modularisierung möglichst optimal vollzogen werden kann, und ob insbesondere auf der Client-Seite den besonderen Anforderungen mobiler Endgeräte, beispielsweise sparsamer Umgang mit Batterieressourcen, gerecht geworden werden kann.

Die Ergebnisse der Untersuchung sollen prototypisch für die vorhandene Infrastruktur umgesetzt werden. Dies schließt insbesondere einen wiederverwendbaren Android-Service und eine Server-Bibliothek ein. Im Anschluss ist durch eine geeignete Evaluation die Validität der Untersuchungsannahmen nachzuweisen.

SCHWERPUNKTE

- Recherche verwandter Arbeiten zu Crowdsourcing und Projekt-Modularisierung,
- Erstellung eines wiederverwendbaren, auf Mobilgeräte skalierten Modularisierungskonzeptes,
- Prototypische Umsetzung des Konzepts und
- Evaluation und Bewertung der Ergebnisse.

Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill
(zuständiger Hochschullehrer)

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Prüfungsamt

Antrag auf Verlängerung der Bearbeitungszeit

☒ **Diplomarbeit** (maximal 3 Monate)

☐ **Bakkalaureatsarbeit** (maximal 2 Monate)

Zutreffendes ankreuzen

Name	Zhang	Vorname	Duoqi
Geburtsdatum	30.08.1984	Fachsemester	14
Matrikelnummer	3560753	Studiengang	Informatik

Betreuender HSL: _____

Beginn: 02.07.2014

Abgabe: 02.01.2015

Dauer der Verlängerung: 3 Monate

Neuer Abgabetermin: 02.04.2015

Begründung der Verlängerung:

Meine Diplomarbeit läuft nicht gut.
Es wird mehr Zeit für die Fertigstellung der Implementierung und die Durchführung der Evaluation benötigt.

Zustimmung betreuender HSL: Schill

Dieser Antrag ist mit einer Kopie der Aufgabenstellung für die Diplomarbeit termingerecht im Prüfungsamt vorzulegen.

08. DEZ. 2014 — Datum

technische Universität Dresden
Fakultät Informatik
Prüfungsamt
01062 DRESDEN

Albrecht — Unterschrift Prüfungsamt

Entscheidung des Prüfungsausschusses:

Dem Antrag auf Verlängerung wird stattgegeben / ~~nicht stattgegeben~~

11. DEZ. 2014 Ch. Baier

Datum/Unterschrift Prüfungsausschuss

Prof. Dr. Christel Baier
Vorsitzende des Prüfungsausschusses
Studiengang Informatik
Technische Universität Dresden
Fakultät Informatik

ERKLÄRUNG

Hiermit erkläre ich, ...(Name, Vorname), geboren in ....., dass die vorgelegte Diplomarbeit mit dem Titel .....
.....
..... durch mich selbstständig verfasst wurde. Ich habe keine anderen als die angegebenen Quellen sowie Hilfsmittel benutzt und die Diplomarbeit nicht bereits in derselben oder einer ähnlichen Fassung an einer anderen Fakultät oder einem anderen Fachbereich zur Erlangung eines akademischen Grades eingereicht.

Ort, Datum	Unterschrift

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

1 Einführung

Crowdsourcing ist ein verteiltes Problemlösung-Model. Durch Sammlung und Analysierung von den „*Wisdom*“ aus einer flexiblen Crowd kann eine komplexe Aufgabe erledigt werden. Am Häufigsten ist diese Aufgabe in mehrere kleine Teile geteilt. Ein Crowdsourcer kann sich auf nur einen kleinen Teil einer Aufgabe konzentrieren.

Ein Crowd mit ausreichenden Crowdsourcer spielt eine wichtige Rolle in Crowdsourcing. Mit der Entwicklung der mobilen Kommunikation und dem Erhöhen der Anzahl von Smartphones Benutzern wird eine riesige potenzielle Menge von Crowdsourcer für Crowdsourcing vergrößert. In den letzten Jahren waren einige Anwendungen in Smartphones für Crowdsourcing auftreten, z.B. Gigwalk [1], OpenStreetMap [2]. Durch diese Anwendungen werden die Crowdsourced-Daten aus Crowdsourcer an den Crowdfounder gesandt und nach Verarbeitung mitbenutzt. Ein Szenario ist, dass mehrere Crowdsourcing-Anwendungen gleichzeitig in einem Smartphone laufen. Daraus gibt es ein Problem, dass einige Komponenten mit gleichen Funktionen in diesen Anwendungen wiederholt werden, z.B. Web-Kommunikation. Die Abbildung 1-1 beschreibt diese Situation. In diesem Fall existieren mehrere Komponenten mit gleichen Funktionen in einem Mobilen Gerät. Außerdem, wenn eine neue Crowdsourcing-Anwendung entwickelt wird, wird eine Komponente wie eine Web-Kommunikation-Komponente widerholt. Ein Ergebnis davon ist, nicht nur die System Ressourcen in einem mobilen Gerät zu verschwenden, sondern auch die Entwicklung neuer Crowdsourcing-Anwendungen zu komplizieren.

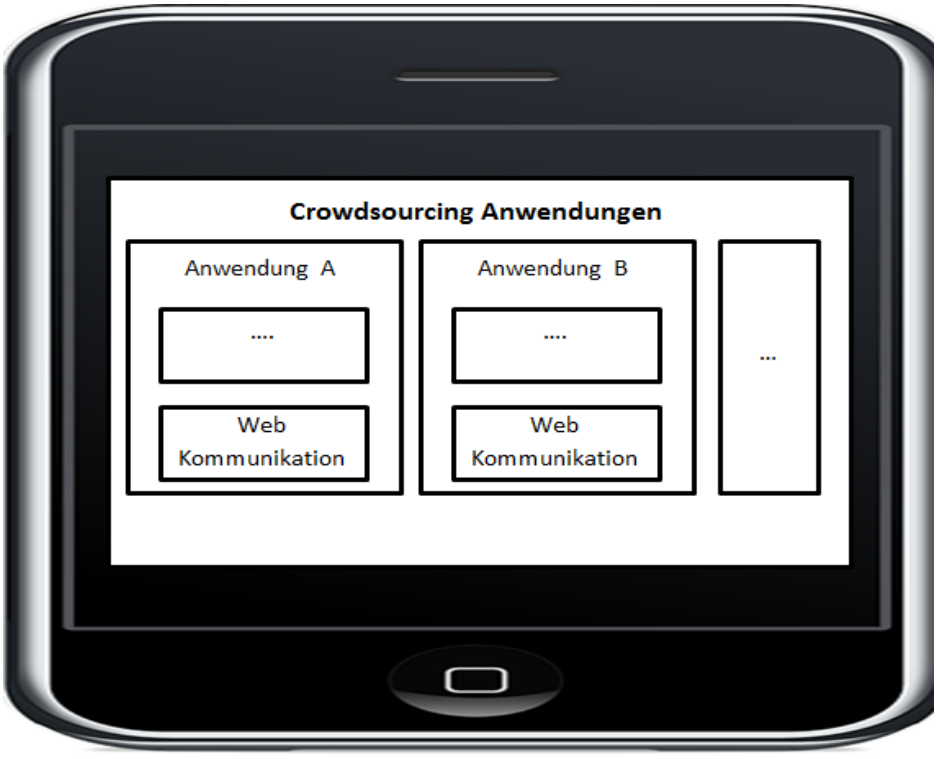


Abbildung 1-1: Crowdsourcing-Anwendungsszenario in Mobilen Geräten

Im Vergleichen zum Desktop haben Mobile-Geräte begrenzte Ressourcen, z.B. begrenzte RAM und Akkulaufzeit. Die begrenzten Ressourcen behindern die Anwendung von

Crowdsourcing in mobilen Geräten. Systemressourcen-Sparen in Mobilen Geräten sollte in jeder Crowdsourcing-Anwendung überlegt werden. Eine einfache Lösung davon ist, gleiche Funktionen in eine wiederverwendbare Komponente, sowie eine Crowdsourcing-Infrastruktur zu integrieren, sodass die mobilen clientseitigen Systeme Ressourcen gespart werden könnten. Natürlich sollte diese Crowdsourcing-Infrastruktur die Eigenschaften von Wiederverwendbarkeit und Erweiterbarkeit besetzen. Auf Serverseite gibt es fast gleiche Probleme wie auf Clientseite, sowie einige Komponenten mit gleichen Funktionen wieder zu implementieren, z.B. Benutzerverwaltung.

Außerdem gibt es auch insbesondere Funktionen in Crowdsourcing, z.B. Benutzer-Anonym und Crowdsourcer-Anreizen. Benutzer-Anonym bedeutet, dass ein Benutzer einige Crowdsourced-Daten anonym abgeben kann. Crowdsourcer-Anreizen bedeutet, dass ein Crowdsourcer motiviert werden sollte. Diese Funktionen werden häufig von Crowdsourcing-Anwendungen angefordert. Eine generelle Crowdsourcing-Infrastruktur sollte diese Funktionen unterstützen. Eine Middleware könnte zwischen Client und Server in einer Crowdsourcing-Infrastruktur hinzugefügt werden. Diese Middleware kann diese Funktionen erledigen. Die Abbildung 1-2 beschreibt diese Möglichkeit einer Crowdsourcing-Infrastruktur. In diesem Fall könnte die Entwicklung und das Hinzufügen neuer Crowdsourcing-Anwendungen vereinfacht werden.

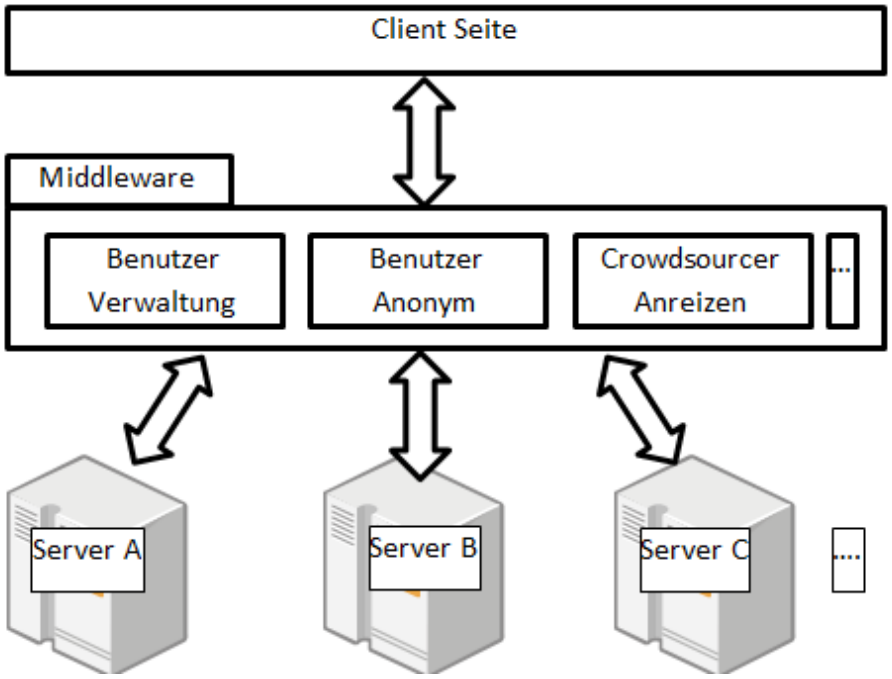


Abbildung 1-2: Generelle Middleware für Crowdsourcing Server

Die Motivationen einer Crowdsourcing-Infrastruktur in Mobile Geräten sind, die Ressourcen von Mobilen Geräten auf Clientseite zu sparen, gleichzeitig die Entwicklung und Realisierung einer Crowdsourcing-Anwendung auf beiden Seiten sowie auf Mobilen Geräten und Crowdsourcing-Servern zu vereinfachen. Außerdem sollte eine Crowdsourcing-Infrastruktur die spezifischen Funktionen, z.B. Crowdsourced-Daten anonym abgeben und Crowdsourcer motivieren, unterstützen.

1.1 Aufgabenstellung

Im letzten Abschnitt wurden die Motivationen einer generellen Crowdsourcing-Infrastruktur vorgestellt. Nach der Aufgabenstellung dieser Diplomarbeit gibt es schon eine Crowdsourcing-Infrastruktur in Projekt MapBiquitous. Diese Infrastruktur kann von anderen Crowdsourcing-Anwendungen nicht verwendet, aber auf Basis von dieser Infrastruktur könnte eine neue generelle Crowdsourcing-Infrastruktur entstehen, die die Motivationen im letzten Abschnitt erfüllen könnten.

Es gibt hauptsächlich zwei Aufgaben in dieser Diplomarbeit. Die erste Aufgabe ist, eine generelle Crowdsourcing-Infrastruktur zu entwickeln. Diese Crowdsourcing-Infrastruktur könnte nicht nur von unterschiedlichen Crowdsourcing-Anwendungen wiederverwendet werden, sondern auch die Anforderungen von Mobilen Geräten erfüllen, z.B. RAM sparen. Die zweite Aufgabe ist, die neue generelle Crowdsourcing-Infrastruktur zu evaluieren, z.B. wie viele Ressourcen in einem mobilen Gerät gespart werden kann? Ob die Crowdsourcing-Infrastruktur optimal modularisiert werden kann.

1.2 Struktur dieser Diplomarbeit

Nach der Vorstellung von Motivation soll kurz auf den Aufbau und die Gliederung der Arbeit eingegangen werden. Die ganze Arbeit gliedert sich in acht Kapitel.

- Im Kapitel 2 werden die Grundlagen und aktuellen Stände der Forschung in Crowdsourcing dargestellt, darin wird die Definition von Mobile Crowdsourcing und ihre Anwendungen benotet. Außerdem werden die Erkenntnisse von Software-Infrastruktur und -Modularisierung gesprochen. Weil die Implementierung sich auf Android OS basiert, wird Android OS kurz vorgestellt.
- Im Kapitel 3 wird eine Maßnahme sowie Anforderung Analyse getroffen. Durch Anforderungsanalyse werden die Zielstellungen dieser Diplomarbeit vertieft. Darin werden die konkreten Anforderungen aufgezählt.
- Im Kapitel 4 wird ein Konzept einer generellen Crowdsourcing-Infrastruktur ausführlich vorgestellt.
- Im Kapitel 5 wird eine von dem Konzept angesetzte Crowdsourcing-Infrastruktur realisiert.
- Im Kapitel 6 werden die Anforderungen der generellen Crowdsourcing-Infrastruktur überprüft. Nach Motivationen einer generellen Crowdsourcing-Infrastruktur wird ihre Leistung untersucht.
- Im Kapitel 7 wird die ganze Diplomarbeit zusammengefasst und werden weitere Arbeiten vorgestellt.

2 Grundlagen

Im diesem Kapitel werden die Erkenntnisse behandelt, die für die folgende Arbeit relevant sind. Bis jetzt gab es noch kein konkretes Konzept von Crowdsourcing-Infrastruktur, deshalb werden die Grundlagen von Crowdsourcing und ihre wichtige Entwicklung sowie Mobile Crowdsourcing vorgestellt. Darin werden nicht nur die Begriffe diskutiert, sondern auch einige typische Crowdsourcing-Anwendungen gezeigt. Eine generelle Infrastruktur sollte gut modularisiert werden, z.B. die Komponenten in dieser Infrastruktur los sind. Eine Infrastruktur mit einer optimalen modularisierten Architektur kann ihre Wiederverwendbarkeit erhöhen. Deshalb wird die Erkenntnisse und Prinzipien der Modularisierung vorgestellt. Zum Ende wird Android OS kurz vorgestellt, weil die neue generelle Crowdsourcing-Infrastruktur in Android OS implementiert wird.

2.1 Crowdsourcing

Crowdsourcing ist ein verteiltes Problemlösung-Model. Durch Sammlung und Analysierung von „*wisdom*“ aus einer flexiblen Crowd kann eine komplizierte Aufgabe einfach erledigt werden. Es gibt unterschiedliche Definitionen von Crowdsourcing. Zuerst wird die ursprüngliche Definition vorgestellt.

2.1.1 Definition von Crowdsourcing

Crowdsourcing ist ein Neologismus des „Wired Autors“ Jeff Howe und erschien erstmal in *Wired Story* „The Rise of Crowdsourcing[1]“. Die ursprüngliche White Paper Definition von Jeff Howe ist:

> **Definition 1** Crowdsourcing
>
> *Crowdsourcing is the act of taking a job traditionally performed by a designated agent (usually an employee) and outsourcing it to an undefined, generally large group of people in the form of an open call* [3].

Danach wurden viele unterschiedliche Crowdsourcing Definitionen daraus abgeleitet. Warum gibt es so viele unterschiedlichen Crowdsourcing-Definitionen? Die Gründe dafür sind folgende:

- Eine neue Definition wird nicht in einer Domain spezifiziert. Ein Anwendungsfachgebiet kann die Konzepte von Crowdsourcing stark beeinflussen.
- Die zentralen Eigenschaften oder Begriffe von Crowdsourcing wurden noch nicht standardisiert. Das bedeutet, dass immer mehr Definitionen von Crowdsourcing entstehen.
- Andere Eigenschaften, z.B. Anonymous, sollte in Crowdsourcing-Plattform sowie Crowdsourcing-System gelegt werden, weil Anonymous nicht eine notwendige Eigenschaft in Crowdsourcing, sondern eine zusätzliche Aufgabe nur für spezifische Anforderungen ist.

Trotzdem können daraus einige Eigenschaften oder Begriffe extrahiert werden. Sie sind Crowdsourcer, Crowdsourced-Task, Crowdsourcing-Platform, Erstellung von Crowdsourcing usw. Diese Eigenschaften entwickeln sich in andere Begriffe weiter, z.B. Crowdfounder, Crowdsourced-Daten.

[1] http://archive.wired.com/wired/archive/14.06/crowds.html [Zugriffszeit: 01.03.2015]

Trotz vieler unterschiedlicher Crowdsourcing-Definitionen gibt es eine Aufteilung von Crowdsourcing-System in der Literatur [4], sowie *Crowd Rating, Crowd Creation, Crowd Processing, Crowd Solving.* Das erste Aufteilung-Prinzip ist, ob die Teilnehmer in Crowdsourcing gleich sind. Das zweite Aufteilung-Prinzip ist, ob es einige Beziehung zwischen aus Crowdsourcer abgegebenen Daten gibt. Die Abbildung 2-1 zeigt die vier Typen von Crowdsourcing-Systemen.

Die Crowdsourcer in *Crowd Rating* und *Crowd Processing* sind identisch miteinander. Die Crowdsourced-Daten aus Crowdsourcern in *Crowd Rating* haben einige Beziehungen miteinander, z.B. alle Crowdsourced-Daten eine Crowdsourced-Aufgabe zusammen erledigen können. Wie die Crowdsourced-Daten gesammelt und kombiniert werden, spielt eine wichtige Rolle in *Crowd Rating*. Im Vergleichen dazu können die Crowdsourcer in *Crowd Processing* parallel und direkt eigene Daten hochladen. Diese Daten sind unterschiedlich miteinander.

Die Crowdsourcer in *Crowd Creation* und *Crowd Solving* sind unterschiedlich miteinander. Weil die Crowdsourcer in *Crowd Creation* miteinander kommunizieren können, bestehen sie aus einem „Netz“ oder einer „Community“. In *Crowd Creation* können Crowdsourcer zusammenarbeiten und ein Crowdsourcing-Problem zusammen lösen. *Crowd Solving* ist einfacher als *Crowd Creation*. In *Crowd Solving* gibt es keine Relation zwischen unterschiedlichen Crowdsourcer. Aber die Daten aus Crowdsourcer sind miteinander unterschiedlich.

Aus dieser Abbildung hat *Crowd Processing* die einfachste Struktur, *Crowd Creation* hat die komplexeste Struktur. Dieses Aufteilung-Prinzip konzentriert sich auf Crowdsourcer und Crowdsourced-Daten. Offensichtlich spielen die Crowdsourcer und Crowdsourced-Daten die wichtigsten Rollen in Crowdsourcing. Aber gibt es noch andere Faktoren für Crowdsourcing-Aufteilung, z.B. die Anzahl von Crowdsourcing-Problemen, wie es gibt nur einziges festgelegtes Problem oder mehrere Problemen in Crowdsourcing.

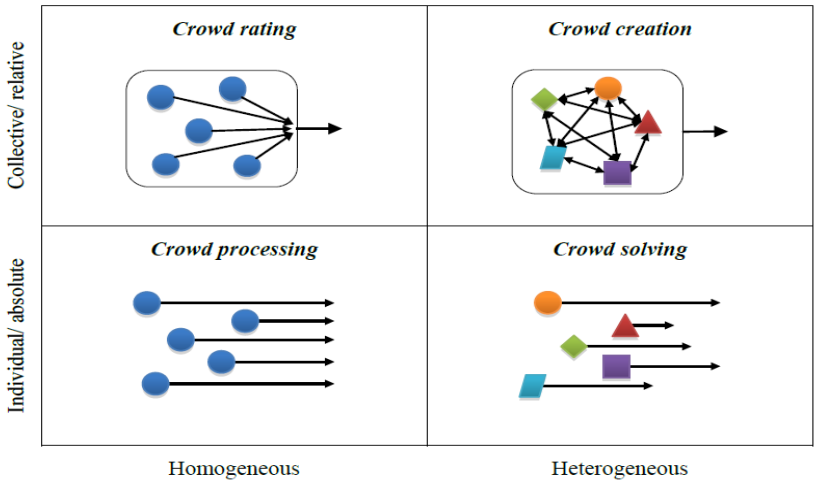


Abbildung 2-1: vier Typen von Crowdsourcing-Systemen

2.1.2 Crowdsourcing-Anwendungen

Wegen der Web-Service-Technologien entwickelt sich Crowdsourcing in Crowdsourcing-Service. Jetzt gibt es schon viele Crowdsourcing-Service auf dem Markt. Ein Verzeichnis von Crowdsourcing-Services wird von Webseite [2] angeboten. Diese Webseite nicht nur sammelt die Crowdsourcing-Services, sondern auch klassifiziert sie in unterschiedliche Kategorien, z.B. Amazon Mechanical Turk gehört zur Kategorie „Micro Tasks“, „iStock Photo“ gehört zur Kategorie „Contents Markets“. Am Nächsten werden drei Beispiele sowie „iStock Photo [3] “, „Amazon Mechanical Turk [4] “ und „BOINC[5]“ vorgestellt. Damit wird angezeigt, wie Crowdsourcing verwendet wird.

iStock Photo

Bei „iStock Photo“ kann ein Teilnehmer eigenen Fotos hochladen und muss einen Vertrag über Eigentum der Fotos unterschreiben. Ein Teilnehmer, der Fotos für Poster oder Werbungen suchen möchte, kann ein Foto bei „iStock Photo“ kaufen und dafür bezahlen. Die Kosten sind häufig niedriger als Marktpreise von Poster-Fotos. iStock profitiert eine kleine Teile von der Zahlung. Der Foto-Anbieter bekommt diese verbleibende Zahlung.

Amazon Mechanical Turk

Ein Nutzer kann in AMT (abkürzt für **A**mazon **M**echanical **T**urk) eine Aufgabe stellen und bietet geleichzeitig eine Vergütung an. Andere Nutzer können diese Aufgabe schauen und eigene Lösungen abgeben. Der Aufgabe-Anbieter kann eine optimale Lösung auswählen und die Vergütung vergeben.

BOINC

BOINC ist ein Open-Source Software für Volunteer-Computing und Grid-Computing. In BOINC kann Teilnehmer eigene Rechnung-Ressourcen von Computern beitragen. Mit diesen Ressourcen können unterschiedlichen Aufgaben erledigt werden, z.B. die globale Klimaerwärmung zu studieren, Pulsare zu entdecken oder viele andere wissenschaftliche Forschungen zu unterstützen. BOINC kann eine Crowdsourcing-Infrastruktur sein, weil sie von unterschiedlichen Projekten verwendet werden. Aber die Crowdsourcing-Ressourcen aus einem Teilnehmer ist einzigartig, sowie Rechnung-Ressourcen in Computer. Jetzt hat BOINC eine Version für Smartphone oder Tablet, d.h. Teilnehmer auch die Rechnung-Ressourcen von Smartphone oder Tablet beitragen kann. In mobilen Geräten rechnet BOINC nur wenn die Smartphone oder Tablet am Strom angeschlossen und vollständig geladen ist.

[2] http://www.resultsfromcrowds.com/features/crowdsourcing-services/
[3] http://www.istockphoto.com/
[4] https://www.mturk.com/mturk/welcome
[5] Adresse: http://boinc.berkeley.edu/ [Zugriffszeit: 05.02.2015]

Aus diesen drei Beispielen können einige Aspekte zusammengefasst werden,

- Eine Aufgabe kann in einem Crowd erstellt werden.
- Die angebotene Lösung können gesammelt werden.
- Abhängig von einer Lösung sollte Lösung-Anbieter von dem Aufgabe-Anbieter prämiert werden, z.B. durch Geld.

Diese drei Eigenschaften können eine grundlegende Definition von Crowdsourcing bauen.

2.1.3 Mobile Crowdsourcing

Außer Crowdsourcing-Service gibt es noch eine wesentliche Entwicklung, sowie die neuen Crowdsourcing-Anwendungen in mobilen Geräten.

Die Entwicklung der Mobile-Geräten und -Kommunikation brachte neue Chance für Crowdsourcing. Die Anzahl von verkauften Smart-Phone hat das traditionelle PC´s überschritten. Das brachte einige Vorteile für Mobile-Crowdsourcing.

- Die Anzahl von Nutzern in unterschiedlichen Crowdsourcing-Anwendungen wird erhöht, damit können mehr Lösungen aus Nutzern bekommen werden. Dadurch kann die Qualität von Crowdsourcing verbessert werden, z.B. eine bessere Lösung eines Problems finden.
- Mobile Geräte besetzen höhere Mobilität, im Vergleichen zu Festsitzen vor einem PC kann ein Mobile Grate durch Drahtlose Netzwerk irgendwo Internet besuchen. Natürlich erhöht das die Anzahl der Verwendung von Crowdsourcing-Anwendungen. Gleichzeitig wird die Anzahl von Lösungen erhöht.
- Eine andere wichtige Eigenschaft von Mobile Geräten, die die traditionalen PCs nicht haben, ist Lokalisieren. Lokalisierung kann die Verwendung von Crowdsourcing erweitern, z.B. in Gigwalk [1] werden die Crowdsourced-Aufgaben in einer digitalen Karte abhängig von Geo-Daten markiert. Ein Benutzer kann abhängig von einer eigenen Position entscheiden, die nächste Crowdsourcing-Aufgabe auszuwählen.
- Wegen höherer Mobilität kann ein mobiles Gerät mit mehrere Sensoren einschließen und ihren Daten in Echtzeit sammeln.

Jetzt ist das „Mobile Crowdsensing“ ein Forschungsfachgebiet, z.B. laut Literatur [5] wird durch Motivierung von Menschen eine kostengünstige Lösung für Überwachung von Lärmverschmutzung angeboten. Deshalb wird eine Crowdsourcing-Anwendung in Smartphone implementiert, die die städtische Lärm-Verschmutzung messen kann. Die Abbildung 2-2 zeigt diese Mobile-Crowdsensing-Anwendung an.

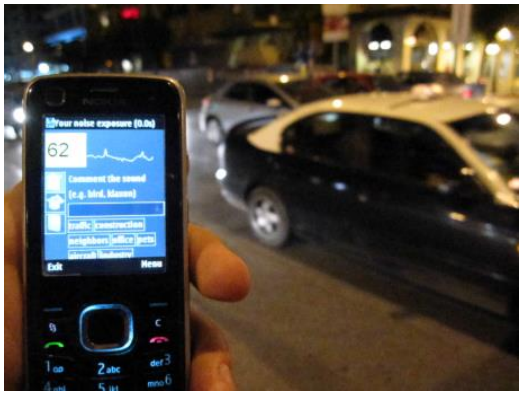


Abbildung 2-2: The NoiseTube Mobile application in use in a busy street in Thessaloniki, Greece [5]

Neben den Vorteilen aus mobilen Geräten für Crowdsourcing gibt es auch Nachteilen. Zuerst, im Vergleichen zu PCs hat ein mobiles Gerät nur begrenzte Ressourcen, z.B. RAM, Speicher, CPU- und Akku-Kapazität. Zweitens, Drahtloser Netzverkehr ist begrenzt und kostet teuer. Dieser Nachteile behindert, die Benutzer in Crowdsourcing teilzunehmen und eigene Lösungen oder Ressourcen beizutragen.

2.1.4 Definition von Crowdsourcing Termen

Bis hier haben die Leser die Grundlagen von Crowdsourcing erfahren, wie Crowdsourcing durch konkrete Beispiele verwendet werden kann. Da keine standardisierte Definition von Crowdsourcing vorhanden, um die Terme von Crowdsourcing in dieser folgenden Diplomarbeit zu verstehen, werden die Begriffe im Folgenden verdeutlicht.

> **Definition 2** Crowdsourcing-Derivat
>
> *Crowdsourcing-Derivat ist eine Applikation oder Software, die allein oder deren einige Komponenten das Ziel von Crowdsourcing erledigen kann oder können.*

Normalerweise ist das Ziel von Crowdsourcing, Probleme zu lösen, deshalb kann ein Crowdsourcing-Derivat eine Applikation oder Software sein, die durch Crowdsourcing einige Probleme lösen kann. Die Applikation „NoiseTube" ist eine Crowdsourcing-Derivat, weil sie das Problem sowie Lärmüberwachung lösen kann. Die Hauptaufgabe in Projekt MapBiquitous ist, extern und intern ein Gebäude zu lokalisieren, aber Nutzer kann durch Crowdsourcing-Funktion die Geo-Daten korrigieren und lesen. Deshalb gehört MapBiquitous zu Crowdsourcing-Derivat.

> **Definition 3** Crowdfunder
>
> *The person or entity requiring one or several resources that are not locally available, but distributed within a flexible group of system entities, the crowdsourcers*. [6]

Crowdfunder bietet normalerweise Probleme in einem Crowdsourcing-System. Um die Probleme zu lösen, braucht Crowdfunder Lösungen aus Crowdsourcer. Außerdem soll Crowdfounder auch die Crowdsourcer motivieren, z.B. durch „Anreiz" oder Prämien.

„Anreiz“ präsentiert alle Faktoren, die die Crowdsourcer motivieren können, eigene „Ressourcen“ beizutragen. Im MapBiquitous werden Nutzer durch „Interesse“ motiviert, eigene Geo-Daten hochzuladen. Diese Daten werden danach in den Gebäudeserver gespeichert und von anderen Benutzern zum Lokalisieren oder zur Navigation verwendet.

Definition 4 Crowdsourcer

An individual member of the crowd using a service provided by the crowdfunder or an entity affiliated with the crowdfunder, producing crowdsourced resources as a side-effect of system usage or actively providing resources to the crowdsourcing process. [6]

Crowdsourcer in einem Crowdsourcing-System verwendet ein Crowdsourcing-Derivat, eigene Ressourcen beizutragen. Im MapBiquitous sind die normalen Benutzer die Crowdsourcer.

Definition 5 Crowdsourcing-Aufgaben

Crowdsourcing-Aufgaben sind gleich wie Probleme, die Crowdsourcing lösen will. Normalerweise werden von Crowdfunder angeboten.

Crowdsourcing-Aufgaben sowie Probleme sind notwendige Teile in Crowdsourcing, weil Crowdsourcing ein verteiltes Problemlösung-Model ist. Nicht nur eine Crowdsourcing-Aufgabe sondern auch mehreren können in einem Crowdsourcing-Derivate existieren. Diese Aufgaben können vor Entwicklung eines Derivats festgelegt oder während der Verwendung in einem Derivat dynamisch erstellt werden. Z.B im MapBiquitous, die einzige Crowdsourcing-Aufgabe sowie Hochladen von Geo-Daten wird am Anfang der Entwicklung schon festgelegt. Z.B. in „Amazon Mechanical Turk“ können Crowdsourcing-Aufgaben dynamisch von Benutzern erstellt werden.

In einem Crowdsourcing-Derivat müssen die Crowdsourcing-Aufgaben klar und attraktiv sein. Eine attraktive Aufgabe kann mehr Teilnehmer faszinieren. Je mehr Teilnehmer, desto mehr Ressourcen können bekommen werden. Ein Forschungsgebiet „wie die Crowdsourcing-Aufgaben sollten effektiv entworfen werden“ konzentriert sich darauf.

Definition 6 Crowdsourcing-Anreiz

Anreiz bedeutet auf alle Faktoren, die die Crowdsourcer motivieren können, seine Crowdsourced-Daten beizutragen.

Anreiz muss von Crowdfunder definiert werden. Anreiz können in unterschiedlichen Arten sein, z.B. Interesse, Prämien, Ehre oder Geld. Anreiz spielt eine wichtige Rolle in Crowdsourcing und kann grundsätzlich die Kosten eines Crowdsourcing präsentieren, weil Crowdfunder Anreiz bezahlen müssen. Anreiz einer Crowdsourcing-Aufgabe sollte so wenig wie möglich kosten aber attraktiv sein.

Definition 7 Crowdsourced-Daten

A subset of (crowdsourced) resources. – Any type of measurands ascertained and digitalized. Such measurands may be physical readings, numerical inputs, et cetera. [6]

Hier wird diese Definition etwas ergänzt. Die aus Crowdsourcer quellenden Crowdsourced-Daten bedeuten nicht nur auf „Measurands" sondern auch auf die Lösungen oder Ressourcen, die die Crowdsourcing-Aufgaben lösen können. In BOINC sind die Rechnung-Ergebnisse die Crowdsourcing-Daten.

Definition 8 Crowdsourced Information

Any type of knowledge extractable from (crowdsourced) data. Either the information is directly contained in the data, or it can be computed as an indirect result from the data. [6]

Die aus Crowdsourced-Daten extrahierten Informationen sind sehr vielfältig, z.B. wie viele Crowdsourced-Daten schon gesammelt wurden, wer diese Crowdsourced-Daten abgegeben hat. Diese Informationen sind sehr wichtig dafür, die Qualität von Crowdsourcing zu beurteilen und zu erhöhen. Wenn die gesammelten Crowdsourced-Daten nicht ausreichend sind, eine Crowdsourcing-Aufgabe lösen zu können, scheitert diese Crowdsourcing. Aus der Submission-Anzahl eines Crowdsourcers kann es gefunden, wer ein „Experte" ist. Wenn „Experte" mehr Crowdsourced-Daten oder Lösungen beiträgt, erhöht sich die Erfolg-Wahrscheinlichkeit von Crowdsourcing.[6] Ein Forschungsgebiet sowie „wie erhöht die Qualität von Crowdsourcing?" konzertiert sich darauf, z.B. die „Experte" aus Crowdsourcer zu finden und die „Experte" mehr Aufgaben zu vergeben.

Die Ereignisse in Crowdsourcing sind ähnlich wie, zuerst werden die Crowdsourcing-Aufgaben von Crowdfunder erstellt, dann werden die Crowdsourced-Daten aus Crowdsourcer durch ein Crowdsourcing-Derivat an den Crowdfunder übertragen. Schließlich kann Crowdsourcer „Anreiz" bekommen.

2.2 Modularisierung

In diesem Abschnitt werden die Grundbegriffe von Modularisierung auf die Java Plattform vorgestellt, weil eine generelle Crowdsourcing-Infrastruktur in dieser Diplomarbeit durch Pragrammsprache JAVA realisiert wird.

2.2.1 Definition

Hier gibt es eine Definition eines Moduls.

Definition 9 Modul

A Software is a deployable, manageable, natively reusable, composable, stateless unit of software that provides a concise interface to consumers. [7]

[6] Annehmen: „Experte" kann richtigere und sinnvollere Crowdsourced-Daten beitragen.

Die „Software“ in dieser Definition könnte „*classes“, „packages“, „components“, „services“*, oder sogar „*Applications“* sein. Aus dieser Definition können sechs Eigenschaften zusammengefasst und kurz erklärt werden.

Deployable

Ein Modul ist eine Einheit von Deployment und kann normalerweise von unterschiedlichen Applikationen verwendet werden, deshalb präsentieren Module mehr physikalischer und grobkörniger als „*class“* und „*package*“. Die deployable Module sind zum Beispiel sind WAR- und JAR-Dateien.

Management

Ein Modul ist eine Einheit von Management. Module können in einer Laufzeitumgebung installiert, deinstalliert und aktualisiert werden. Während der Entwicklungsphase eines Software Systems kann ein neues und unabhängiges Modul hinzugefügt werden.

Natively Reusable

Module sind Einheiten, die nur in einem Prozess wiederverwendet werden können, d.h. dass ein Modul immer native aufgerufen wird. Das unterscheidet von Applikationen oder Services.

Composable

Module können miteinander verbinden, eine Funktion zu erledigen.

Stateless

Ein Modul hat keinen Zustand. Es gibt nur eine Instanz von einem bestimmen Version eines Moduls.

eine Meinung ist, sowie „the best candidate as the unit of modularity on the java platform is the JAR file! [7]”

Es gibt zwei Anwendungsfälle für Modularisierung. Erster Anwendungsfall, Modularisierung in einer Laufzeitumgebung konzentriert sich auf die Verwaltung eines modularisierten Systems. Falls ein Modul oder Service nicht funktionieren würde, z.B. beim Ausfall oder Aktualisieren, würde das ganze System nicht wieder starten, d.h. ein Modul in einem komplizierten System dynamisch verändert werden kann. Eine schöne standardisierte Modularisierung Framework ist OSGi(abkürzt für **O**pen **S**ervice **G**ateway **I**nitiative). In diesem Framework kann ein Modul wie ein Service dynamisch definiert und verwaltet werden.

Zweiter Anwendungsfall ist Modularisierung in Entwicklungsprozess einer Applikation. Ein Modularisierungskonzept könnte einem Entwickler helfen, wiederverwendbare, erweiterbare und low-coupling Module aufzubauen. Das ist sehr wichtig für die generelle Crowdsourcing-Infrastruktur, weil die vorhandene Crowdsourcing-Infrastruktur in MapBiquitous eingebettet wird, d.h. sie hat eine starke Korrelation mit MapBiquitous. Durch Modularisierung kann die Abhängigkeit zwischen Modulen darin reduziert werden.

Leichtgewichtige Komponenten können in anderen Applikationen wieder verwendet werden.

2.2.2 Beziehung zwischen Use und Reuse

Ein Ziel von Modularisierung ist, die Wiederverwendbarkeit eines Moduls zu realisieren. Offensichtlich können mit wiederverwendbaren Modulen die Kosten und Zeit einer neuen Applikation gespart werden. In diesem Abschnitt wird die dialektische Beziehung zwischen Verwenden und Wiederverwenden kurz vorgestellt.

Um Module wiederzuverwenden, sollten sie flexible sein. In diesem Fall wird die Komplexität eines Systems erhöht. Wegen höherer Komplexität wird es behindert, ein Modul zu verwenden. Die Abbildung 2-2 zeigt die Beziehung zwischen Verwenden und Wiederverwenden.

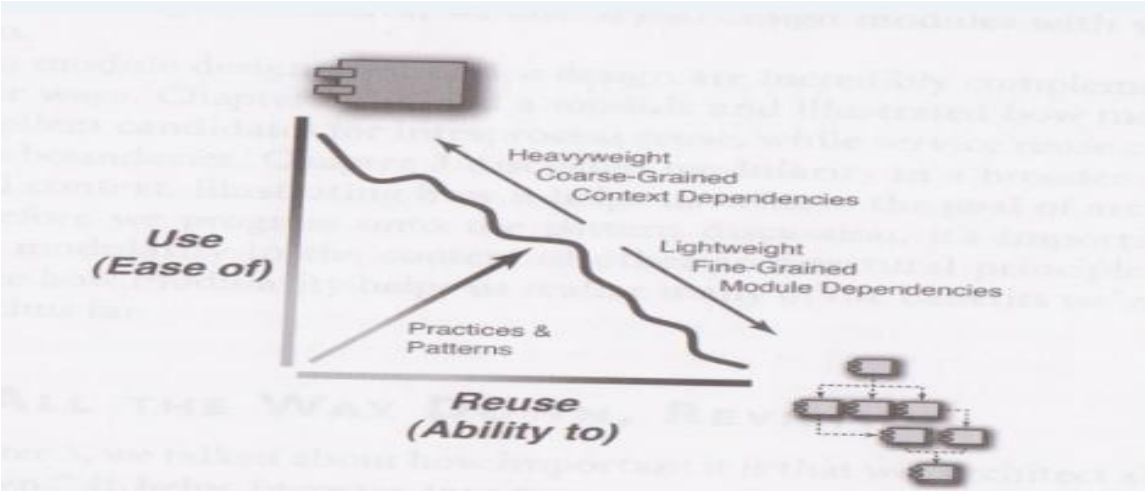


Abbildung 2-3: Beziehung zwischen Verwendbarkeit und Wiederverwendbarkeit [7]

Trotzdem können zwei Regeln vorgestellt werden,

- Coarse-grained modules are easier to use, but fine-grained modules are more reusable [7]. (R1)
- Light-weight modules are more reusable, but heavyweight modules are easier to use [7]. (R2)

Granulieren bedeutet die Aufteilung eines Systems in kleine Komponenten. Normalerweise bietet ein corse-grained Modul mehr Funktionen als ein fine-grained Modul. Eine Methode für Modularisierung ist die Kombination aus fine-grained Modulen in corse-grained Modulen.

„Weight“ bezieht sich darauf, wieviel ein Modul von anderen Modulen in Umgebung abhängig ist. Ein *heavy-weight* Modul bedeutet, dass ein Modul mehr stark von seiner Umgebung abhängig ist. *Weight* sowie Abhängigkeit spielt eine wichtige Rolle bei Modularisierung. Um die Wiederverwendbarkeit eines Moduls zu erhöhen, sollte seine Abhängigkeiten von anderen Modulen reduziert werden.

2.2.3 Prinzipien der Modularisierung

In Software Engineering gibt es unterschiedliche Prinzipien für Erstellung einer Modularisierung Struktur für ein Projekt. Hier werden zuerst drei klassische Prinzipien angesprochen.

- Low Coupling and High Cohesion (P1)

Stevens et.al [8] schlug vor, sich auf die Daten-Kommunikation-Abhängigkeit in einem System zu konzentrieren. Ein Merkmal von erfolgreicher Modularisierung ist, dass die Elemente in einem Modul highly cohesive sind, z.B. die classes in einem JAR File enge Abhängigkeit zueinander haben sollten. Im Gegensatz sollten die Elemente von unterschiedlichen Modulen low-coupled sein, z.B. sollten die Klassen in unterschiedlichen JAR lose Abhängigkeit zueinander haben. Dieses Prinzip könnte als ein Standard-Ansatz für Analysierung der Modularisierung sein.

- Information Hiding (P2)

Parnas [9] stellte ein Prinzip von „*Information-Hiding*“ vor, sowie Entscheidungen in Modulen verstecken. Schlanke Schnittstellen sollen die Fassade eines Modules präsentieren. Jede Entscheidung sollte in einem unabhängigen Modul gekapselt werden. Das Prinzip zeigt die Wichtigkeit von Schnittstellen in Modularisierung und reduziert die Kosten von Veränderung bei Entwürfen eines Moduls.

- Conway ' s Law (P3)

Conway [10] bietet ein allgemeines Gesetz an,

> Organizations which design systems [...] are constrained to produce designs which are copies of the communication structures of these organizations

Obwohl Conway ' s Law eine Richtlinie für Modularisierung nicht direkt zeigt, wird ein Prinzip angeboten.

Die drei oben genannten Prinzipien werden häufig zitiert. Außerdem gibt es auch andere Prinzipien, die Modularisierung stark beeinflussen können.

- Martin [11] bietet „*Three Package Design Principe*“, sowie reuse-release equivalence principle, common-reuse principle, common-closure principle.
- Layered-Architectur beschreibt eine Regel von Struktur-Abhängigkeit für Modularisierung.
- Unter das Konzept von Aspect-Orieted Programming könnte Modularisierung sich auf non-cross-cutting concerns beziehen.

2.3 Android Mobile OS

Android Mobile OS ist die bekanntestes Mobile OS. Es hat fast 85% Smartphone OS Market im zweiten Quartal 2014 gewonnen[7]. Wegen *Opensource* hat er immer mehr Anwendung-Entwickler angezogen. Hier wird das Applikation Model in Android

[7] http://www.idc.com/prodserv/smartphone-os-market-share.jsp [Zugriffszeit: 01.03.2015]

vorgestellt. Dieses Modell wird in der Abbildung 2-4 angezeigt. Die anderen zwei wichtigen Mobile OS sowie iOS und Windows Phone benutzen auch dieses Model.

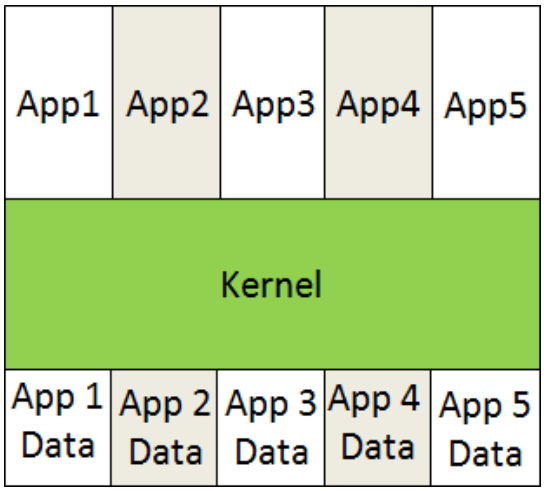


Abbildung 2-4: Applikation Model in Android

Android OS ist ein Multi-User Linux-System. Jede Applikation darin ist ihrer User. Standardmäßig läuft jede Android-Anwendung in einem separaten Prozess unter einem einzigartigen UID mit unterschiedlichen „*Permissions*“. Da jeder Prozess in Android OS eigene VM(abkürzt: Virtual Machine) hat, könnte eine Anwendung normalerweise die Daten oder Ressourcen von anderen Anwendungen nicht lesen und verändern. Die Kernel-Sandbox verwaltet die Daten und Ressourcen jeder Anwendung. Um die Daten und Ressourcen zwischen unterschiedlichen Modulen mitzubenutzen, müssen die Berechtigungen statisch während der Installierung einer Anwendung konfiguriert werden. Zusätzlich wird ein Datentyp wie „*Intent*“ verwendet, die Informationen zwischen Anwendungen zu übertragen. Diese Kommunikation wird von dem Android OS verwaltet, z.B. wie eine „*Activity*“ einen ankommenden oder rückgebenden „*Intent*“ akzeptiert? Eine typische Kommunikation ist die Kommunikation zwischen „*Activity*“ und „*Service*“. Android OS und Applikationen darin spiegeln sich die Modularisierung Prinzipien wieder, weil jede Applikation als ein Modul agieren kann. Der Lebenszyklus eines Moduls und die Kommunikation zwischen Modulen werden auch von Android OS verwaltet.

Eine wichtige Komponente in Android OS für diese Diplomarbeit ist die Web-Komponente, weil fast jedes Crowdsourcing-Derivat das Netzwerk benutzt. Für die Web-Kommunikation verwendet Android nicht nur das „*java.net*“ Package, sondern auch „Apache HTTPClient“[8]. Die beiden Packages basieren auf dem HTTP-Protokoll und könnten die Anforderungen von Web-Kommunikation in meisten Fällen erfüllen. Die beiden Packages wurden schon in Android SDK[9] eingepackt und können direkt benutzt werden.

[8] Google Android 1.0 was released with a pre-BETA snapshot of Apache HttpClient [20]. This means the Apache HTTPClient 4.0 APIs can be uesed in Android Application.
[9] Android SDK abkürzt für Android Software Development Kit, damit kann eine Android Applikation entwickeln.

2.4 Zusammenfassen

In dem ersten Teil dieses Kapitels werden die Definitionen von Crowdsourcing zuerst vorgestellt. Da jede Definition nur auf spezifischen Forschungsaspekt konzentriert, gibt es wenige Definitionen genereller Crowdsourcing-Infrastruktur. Das bringt ein Problem für Erstellung einer Crowdsourcing-Infrastruktur. Nach der Vorstellung von Crowdsourcing-Definitionen werden einige Crowdsourcing-Derivate zum Beispiel angenommen, sowie *iStock Photo, AMT (abkürzt für **A**mazon **M**echanical **T**urk)* und *BOINC*. Obwohl sie erfolgreiche Crowdsourcing-Derivate sind, gibt es noch Begrenzungen in diesen drei Beispielen, z.B. die Crowdsourcing-Aufgabe in *iStock Photo* ist einfach und festgelegt, sowie Foto hochladen. Obwohl unterschiedlichen Crowdsourcing-Aufgaben in *AMT* dynamisch erstellt werden können, kann *AMT* nicht in unterschiedlichen Projekten implantiert werden. Software und Projekt BOINC kann in unterschiedlichen Projekten verwendet werden, aber Teilnehmer sowie Crowdsourcer trägt nur eigene Rechnung-Ressourcen bei, d.h. die Arten von Crowdsourced-Daten aus unterschiedlichen Crowdsourcern in unterschiedlichen Crowdsourcing-Derivaten fast gleich sind.

Wegen der Entwicklung von mobilen Geräten gibt es ein neues Forschungsfachgebiet sowie „Mobile Crowdsensing“. Die Crowdsourcing-Derivate in Mobilen Geräten sammeln häufig die Crowdsourced-Daten, die häufig aus Sensoren in einem Mobilen Geräte quellen. Normalerweise werden diese Crowdsourced-Daten auf Serverseite analysiert und gespeichert, weil Mobile Geräte begrenzte Ressourcen haben, z.B. CPU- und Akku-Kapazität. Zum Ende dieses ersten Teils werden grundlegende Crowdsourcing-Termen vorgestellt. Mit diesen grundlegenden Crowdsourcing-Termen wird eine Definition für eine generelle Crowdsourcing-Infrastruktur entstanden.

Außer Vorstellung von Crowdsourcing werden die grundlegenden Ergebnisse von Modularisierung in dem zweiten Teil vorgestellt. Das Ziel von Modularisierung ist, Widerwendbarkeit zu realisieren. Diese Eigenschaft ist für eine generelle Crowdsourcing-Infrastruktur notwendig. Nach den Modularisierung-Prinzipien kann die Wiederverwendbarkeit einer Crowdsourcing-Infrastruktur beurteilen.

Zum Ende dieses Kapitels wird Android OS kurz vorgestellt, weil die Crowdsourcing-Infrastruktur in dem Android OS implementiert. Am nächsten Kapitel werden die Anforderungen einer Crowdsourcing-Infrastruktur vorgestellt.

3 Anforderungsanalyse

Bevor ein Konzept dieser Diplomarbeit gestellt wurde, ist es notwendig, zu bestimmen, welche Anforderungen das Konzept erfüllen sollte, z.B. welche Funktionen angeboten werden oder wie die Implementierung-Ergebnisse ausgewertet werden sollten.

Das Ziel dieser Diplomarbeit ist, eine generelle Crowdsourcing-Infrastruktur zu erhalten, die die einfache Deployment neues Crowdsourcing-Derivats möglichst intuitiv und mit weniger Konfiguration- und Implementierungsaufwand ermöglichen kann.

Um eine generelle Infrastruktur zu entwerfen, ist eine allgemeine Crowdsourcing-Definition notwendig. Am nächsten wird eine grundlegende Crowdsourcing Definition zuerst vorgestellt, danach werden zwei konkreten von dieser Definition abhängenden Crowdsourcing-Derivate vorgestellt.

3.1 Generelle Crowdsourcing Definition

Heutige Definitionen von Crowdsourcing sind miteinander unterschiedlich. Crowdsourcing ist eine Methode. Dadurch kann ein komplexes Problem verteilt und gelöst werden. Das Ziel von Crowdsourcing ist, mit Anwendung von „Crowd Wisdom“ die Kosten einer Problemlösung zu reduzieren.

Eine spezifische Crowdsourcing-Definition für Crowdsourcing-Infrastruktur sowie Definition 10 wird erstellt. In diese Definition werden die in Kapitel 2.1.4 vorgestellten Crowdsourcing Terme verwendet.

> **Definition 10** Generelle Crowdsourcing
>
> *Ein Crowd besteht aus einer großen Menge von Crowdsourcer und Crowdfunder. Crowdfunder kann eine Crowdsourcing-Aufgabe definieren. Crowdfunder soll Crowdsourcing-Anreiz für Crowdsourced-Daten anbieten. Ein Crowdsourcer kann eine Aufgabe nachfragen und eigene Crowdsourced-Daten dafür an den Crowdfunder vorlegen. Abhängig von Crowdsourced-Daten kann Crowdfunder einen Anreiz für die Crowdsourcer entscheiden. Schließlich bekommen die Crowdsourcer diesen Anreiz.*

Diese Definition sieht ähnlich wie das „*Worker-Slave*“ Modell aus. In anderen Definitionen existieren mehr Identitäten, z.B. *Crowdfounder*, der ein *Crowd* gründet. *Teilnehmer*, der in einem *Crowd* angemeldet haben. Diese Identitäten sowie *Crowdfounder* und *Teilnehmer* existieren nur in einem Crowdsourcing mit Autorisation und Authentifizierung. Crowdsourcer und Crowdfunder als Besitzer von Crowdsourcing-Daten sind die zwei zu mindestens notwendigen Identitäten.

Neben Identitäten beschreibt diese Definition auch den Ereignisse-Fluss zwischen Crowdsourcer und Crowdfunder. Dieser Ablauf kann in jedem Crowdsourcing gefunden werden.

Hier werden einigen Aktivitäten ignoriert, z.B. wie Crowdsourcer eine Lösung generiert. Diese ignorierten Aktivitäten in unterschiedlichen Crowdsourcing sind miteinander sehr unterschiedlich, z.B. ein Crowdsourcer in einem Crowdsourcing-Derivat kann alle Crowdsourced-Daten kommentieren, aber in anderen Crowdsourcing-Derivaten können

die Crowdsourced-Daten von anderen Crowdsourcern nicht gelesen werden. Diese Aktivität sowie Kommentieren für Crowdsourced-Daten ist von den Anforderungen unterschiedlicher Crowdsourcing-Derivate abhängig.

Diese Definition bewahrt den „Anreiz Mechanismus“. Die Abbildung 3-1 zeigt die Unterschiede zwischen 4 Problem-Lösung-Pattern. Ohne „Anreiz Mechanismus“ ist Crowdsourcing nicht mehr Crowdsourcing. Alle Faktoren, die einen Crowdsourcer motivieren könnten, können als Anreiz abstrahiert werden, z.B. Interesse, Geld und Ehrung. Durch Crowdsourcer-Motivierung kann Crowdfunder so viele Lösungen laut Literatur [12] wie möglich bekommen. Die Anzahl von Lösungen beeinflusst die Qualität von Crowdsourcing-Ergebnissen stark.

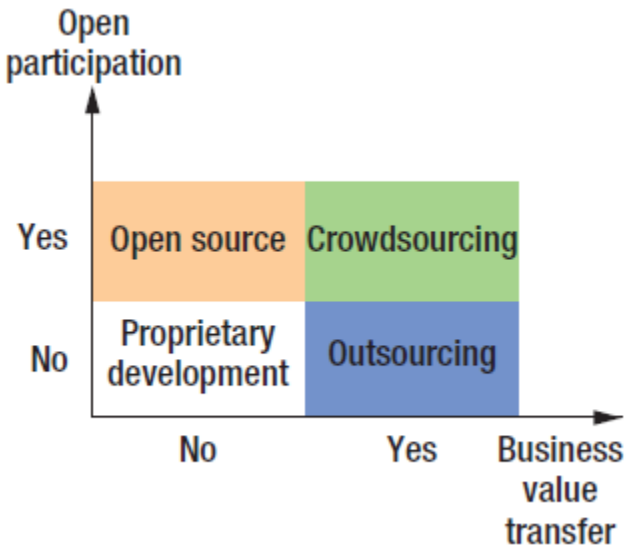


Abbildung 3-1: Die Unterschiede zwischen Crowdsourcing, Outsourcing, Open Source und proprietäre Software-Entwicklung [13]

Die Definition 10 ist die grundlegendste Definition für Crowdsourcing. Um das zu beweisen, wird Definition 10 mit anderen Definitionen verglichen. Im Folgenden gibt es eine neue Definition von Crowdsourcing.

> **Definition 11** *Crowdsourcing is more than just creating a website where people just work. There must be a structure which guides the clients and the crowd in the desired directions, assists the clients in describing the challenges they want solved, and aids the crowd in executing tasks. It is also important that a crowdsourcing platform has a sufficient crowd, must be appealing to encourage participation, and must be relatively easy and feasible for the crowd to use.* [12]

Die Definition 11 ist komplizierter als Definition 10. Neben Anforderungen für Crowd-Teilnehmer wird die einfache und attraktive Interaktion zwischen Benutzer und Crowdsourcing-Plattform auch benotet. Diese zusätzlichen Anforderungen sind die Unterthemen von Crowdsourcing.

Nach Definition 10 kann Crowdsourcing MapBiquitous modelliert werden. Die Crowdfunder sind Gebäudeserver. Die Aufgabe ist nur, einige Position zu korrigieren. Lösung-Anbieter ist der Benutzer von MapBiquitous, der aus Interesse eigene Position Daten hochladen möchte. Der Crowdsourcing-Anreiz in MapBiquitous ist „Interesse". Deshalb brauchen Crowdsourcer tatsächlich nicht, den Crowdsourcing-Anreiz zu erhalten.

3.2 Crowdsourcing-Anwendung Beispiel und Benutzer Szenario

Wie sieht ein Crowdsourcing-Derivat aus? Durch ein konkretes Crowdsourcing-Derivat können die Anforderungen für Crowdsourcing-Infrastruktur gestellt werden. Hier werden zwei Crowdsourcing-Derivate-Beispiele angenommen. Diese Dummy Crowdsourcing-Derivate sind von Definition 10 abhängig. Sie imitieren das schon in Kapitel 2.1.2 vorgestellten Amazon Mechanical Turk. Amazon Mechanical Turk ist ein Vertreter in Crowdsourcing-Anwendungen. Amazon Mechanical Turk wurde nicht nur in vielen Aufsätzen und Zeitschriften zitiert, sondern auch in einigen Lehrveranstaltungen als Einführung von Crowdsourcing angenommen. Zur Überprüfung der Wiederverwendbarkeit werden zwei Dummy Crowdsourcing-Derivate angenommen.

3.2.1 Dummy-Crowdsourcing-Derivate FAF

Ein Dummy-Crowdsourcing-Derivate für Android OS wird als FAF (abkürzt für: **F**ragen-**A**ntwort **F**orum System) genannt. Um dieses Crowdsourcing-Derivat einfach zu implementieren, wird die Erstellung einer Crowdsourced-Aufgabe ignorieren, d.h. es gibt nur statistisch einige Crowdsourced-Aufgaben in FAF, wenn ein Crowdsourcer FAF Server besucht.

Die Prämien wie Crowdsourcing-Anreiz ist ein besonderes Zeichen zu bekommen. Wenn ein Crowdsourcer eine Frage beantwortet, kann er ein besonderes Zeichen bekommen. Die ganzen Fragen werden in FAF Server gespeichert. Ein Benutzer kann nach einem Stichwort die entsprechenden Fragen suchen und daraus eine Frage auswählen, danach eine Antwort dafür abgeben. Die Antwort wird in FAF Server gespeichert. Bei jedem Vorlegen einer Antwort kann der Antwort-Anbieter eine in dieser Frage gestellte Belohnung wie ein besonderes Zeichen bekommen. Diese Belohnung-Informationen werden auch in Crowdsourcer Konto gespeichert. Durch Nachfragen von Benutzer-Information kann ein Belohnung-Rekord erhalten. Jeder Benutzer kann durch Nachfragen einer Frage alle Antworten dafür lesen. Da Crowdsourcer die Prämien bekommen kann, braucht FAF die Userverwaltung. Wegen Sicherheitsproblemen ist User-Anonym notwendig.

3.2.2 Dummy-Crowdsourcing-Derivate NSÜ

Das neue Crowdsourcing-Derivat wird NSÜ System (abkürzt: **N**etzwerk-**S**ignalstärke **Ü**berwachung System) genannt. Ein Crowdsourcer kann eigene Handy-Signalstärke mit Geo-Information hochladen. Crowdsourcer ist der normalen Netzwerk-Benutzer. Die Crowdsourcing-Aufgabe ist Sammlung von Netzwerk-Signalstärken. Crowdsourcing-Anreiz ist Interesse, d.h. aus Interesse werden die Crowdsourced-Daten von Crowdsourcer beigetragen. Die Crowdsourced-Daten enthalten Netzwerk-Signalstärke und Geo-Informationen. Offensichtlich braucht NSÜ keine Verwaltung von Crowdsourcer.

Im Vergleichen zu FAF ist NSÜ einfacher, weil nur eine einzige Crowdsourcing-Aufgabe in NSÜ existiert und die Operation von Crowdsourcer nur Hochladen eigener Crowdsourced-Daten ist.

3.2.3 Benutzer Szene einer generellen Crowdsourcing-Infrastruktur

Jetzt möchten FAF und NSÜ die generelle Crowdsourcing-Infrastruktur verwenden. Es wird zusätzlich erklärt, dass FAF und NSÜ gleichzeitig in einem Mobilen Gerät laufen. Die Crowdfunding Server werden physikalische voneinander abgetrennt. Deshalb wird ein Benutzer Szenario erhalten und in Abbildung 3-2 angezeigt. Daraus werden einigen Punkte zusammengefasst.

- auf Clientseite mehrere Crowdsourcing-Derivate in einem Mobilen Gerät laufen, sie benutzen die Crowdsourcing-Infrastruktur zusammen.
- Auf Serverseite gibt es auch mehrere Server, aber diese Server sind unterschiedlich miteinander, weil jeder Server eigene Crowdsourcing-Derivate unterstützt. Die Crowdsourcing-Infrastruktur schließt die Derivate und Server an und kann wiederverwandet werden.
- Die generelle Crowdsourcing-Infrastruktur kann nicht nur ein Proxy sein, z.B. SANE, sondern auch wiederverwendbaren Komponenten auf Clientseite und Serverseite enthalten. In diesem Fall wird die Entwicklung eines neuen Crowdsourcing-Derivats mehr vereinfacht.

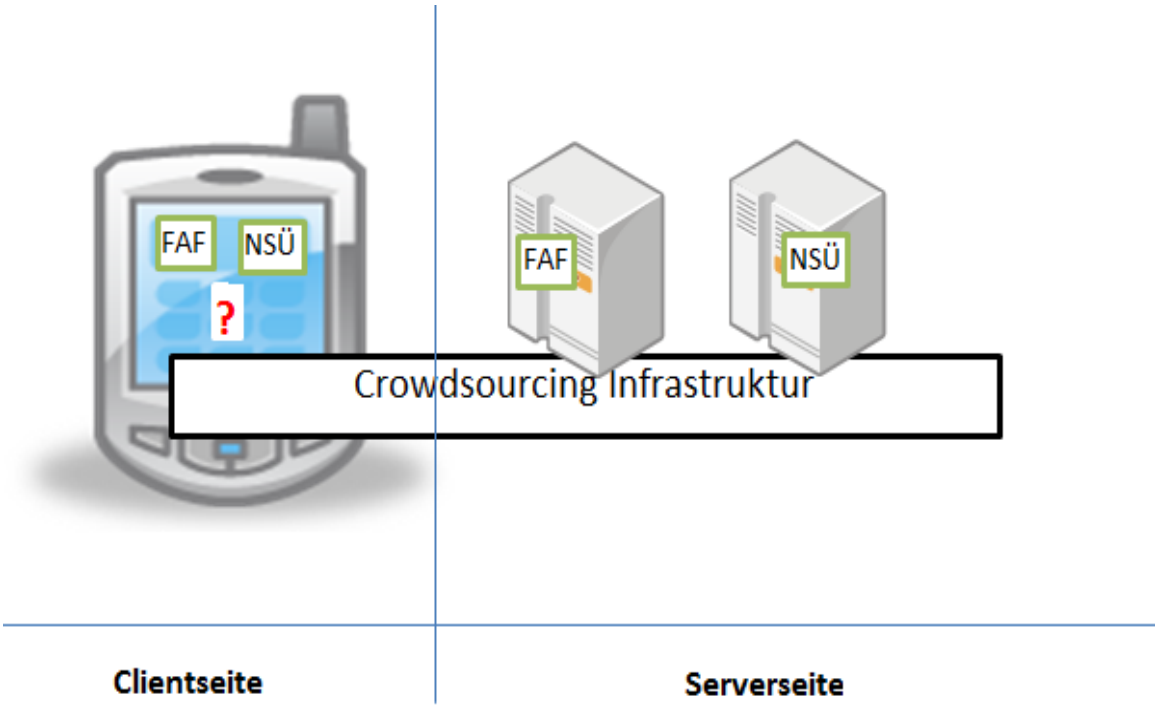


Abbildung 3-2: Benutzer Szenario von Crowdsourcing-Infrastruktur

3.3 Anforderungen für die generelle Crowdsourcing Infrastruktur

3.3.1 Clientseitige Anforderungen der generelle Crowdsourcing-Infrastruktur

Auf Clientseite sollte eine Crowdsourcing-Infrastruktur von Crowdsourcing-Derivaten abgetrennt werden, d.h. nicht jedes Derivat eigene Infrastruktur benutzt, aber unterschiedliche Crowdsourcing-Derivate nur die einzige Infrastruktur in einem Mobilen Gerät zusammen verwenden. Der Crowdsourcing-Infrastruktur sollte eine alleine Anwendung in einem mobilen Geräte sein. Wenn die Crowdsourcing-Infrastruktur einmalig installiert würde, könnte sie mehre Crowdsourcing-Derivate bedienen. Außerdem gibt es noch einen Vorteil. Die Aktualisierung der Crowdsourcing-Infrastruktur beeinflusst Crowdsourcing-Derivate nicht.

Nach dem Begriff von SOA(englisch: Service Oriented Architecture) sollte die generelle Crowdsourcing-Infrastruktur in mobilen Geräten nur ein Service sein, der mehrere Crowdsourcing-Derivate unterstützen könnte. Hier gibt es eine Frage, warum nicht mehrere Instanzen von Crowdsourcing-Infrastruktur parallel laufen? Wegen parallelen Verarbeitungen könnte Crowdsourcing-Infrastruktur natürlich mehrere Aufrufe verarbeiten und in kürzerer Zeit die Ergebnisse von Aufrufen ausgeben. Aber gibt es die Begrenzung von Ressourcen wie RAM in mobilen Geräten. Wenn jedes Crowdsourcing-Derivat eigene Crowdsourcing-Infrastruktur läuft, d.h. mehrere Instanzen von Crowdsourcing-Infrastruktur gleichzeitig in RAM existieren würden, würden sie RAM-Ressourcen verschwenden.

Tatsächlich ist die Beziehung zwischen Crowdsourcing-Derivate und -Infrastruktur, dass mehrere Crowdsourcing-Derivate die einzige Infrastruktur aufrufen, d.h. nur eine Instanz von Crowdsourcing-Infrastruktur in dem System die mehrere Aufrufe aus unterschiedlichen Crowdsourcing-Derivaten verarbeiten soll. Jedoch ist es möglich, dass mehrere unterschiedliche Aufrufe gleichzeitig bei Crowdsourcing-Infrastruktur ankommen könnten. Weil es keine neue Instanz von Crowdsourcing-Infrastruktur dafür gibt, sollten die Aufrufe nicht einfach gelöscht werden. Deshalb ist die Verwaltung von mehreren Aufrufen bei der einzigen Crowdsourcing-Infrastruktur notwendig, z.B. die Aufrufe in einer Warteschlange temporär gespeichert werden könnten.

Der Crowdsourcing-Infrastruktur-Service kann von einem Aufruf gestartet werden. Um die System-Ressourcen zu sparen, z.B. RAM freizusetzen, sollte Service automatisch und sofort selbst ausschalten, wenn es keine Aufrufe mehr gibt. Manchmal könnte diese Aufgabe von Operation-System erledigt werden.

Die Kommunikation zwischen Crowdsourcing-Derivate und -Infrastruktur ist von dem Operation-System in mobilen Geräten abhängig. Trotzdem sollte ein Derivat direkt die Infrastruktur aufrufen, danach sollte die Infrastruktur direkt die Service-Ergebnisse an dieses Derivat zurückgeben, d.h. keine Broadcast verwendet werden sollte. Das ist eine Erwägung für Sicherheit. Ein Sicherheitsrisiko könnte in der Broadcast von Ergebnissen von Infrastruktur passieren, weil andere Derivate auch eine Broadcast aus der Crowdsourcing-Infrastruktur akzeptieren könnten.

Die Operationen in der Tabelle 3-1 werden von Crowdsourcern verwendet und sollte von der generellen Crowdsourcing-Infrastruktur unterstützt werden.

Clientseitige Operationen	Benötige Daten-Objekte
Crowdsourcing-Aufgaben lesen	Aufgaben
Crowdsourced-Daten vorlegen und verwalten	Crowdsourced-Daten
Crowdsourcing-Anreiz bekommen	Crowdsourcing-Anreiz

Tabelle 3-1: Clientseitige Funktionen-Anforderungen der generellen Crowdsourcing-Infrastruktur

3.3.2 Anforderungen eines Standardtools für Entwicklung eines neues Crowdsourcing-Derivats

Die Entwickler eines neuen Crowdsourcing-Derivats sollten die Schnittstellen von Crowdsourcing-Infrastruktur überlegen und verwenden. Ein Schwerpunkt beim Aufrufen von Crowdsourcing-Infrastruktur ist Daten- oder Parameter-Übertragung zwischen Applikationen. Obwohl der Aufruf von dem Client-System abhängig ist, sollten die Parameter oder Aufrufe mit einem deutlichen Format festgelegt werden. Deshalb ist eine Komponente notwendig, z.B. durch diese Komponente eine Anfrage für den Crowdsourcing-Infrastruktur-Service erstellt, gesandt werden kann und die Service-Ergebnisse akzeptiert werden können. Offensichtlich wird diese Komponente in jedem Derivat wieder verwendet. Diese Komponente kann als eine Library in jedem Crowdsourcing-Derivat eingebettet werden. Das spiegelt die Modularisierungsprinzip P2 sowie „*Information Hiding*“ wider. Die Informationen, z.B. der Komponente-Adresse und -Namen[10] von Crowdsourcing-Infrastruktur werden während Aufrufen in dieser Bibliothek versteckt.

Außer Erstellung einer Anfrage für Crowdsourcing-Service und Erfassen von Ergebnissen aus Crowdsourcing-Service sollte die Bibliothek andere Funktionen haben. Weil es nur einzigen Crowdsourcing-Infrastruktur-Service gibt, gibt es Verzögerung zwischen Service-Anfrage und –Ergebnisse. Das Absenden von Service-Anfragen und Empfängen von Service-Ergebnissen in einem Crowdsourcing-Derivat sollte von dieser Bibliothek abgetrennt werden, d.h. die Kommunikation zwischen Crowdsourcing-Derivaten und –Infrastruktur-Service asynchrones ist. Wegen asynchroner Kommunikation sollte diese Bibliothek Crowdsourcing-Service-Ergebnisse miteinander unterscheiden.

Zwar die Ergebnisse aus Crowdsourcing-Infrastruktur von Crowdsourcing-Derivat verarbeitet werden, aber sollte die Library eine Schnittstelle anbieten. Mit dieser Schnittstelle kann Entwickler eine individuelle Schnittstelle realisieren.

Die Formate und Typen von Crowdsourced-Daten in unterschiedlichen Crowdsourcing-Derivaten könnten vielfältig sein, z.B. Text- und Audio-Daten. Deswegen sollte das Crowdsourcing-Tool die unterschiedlichen Crowdsourced-Daten unterstützen, d.h. unterschiedlichen Crowdsourced-Daten in Service-Anfrage eingepackt werden können.

3.3.3 Serverseitige Anforderungen der generellen Crowdsourcing-Infrastruktur

Serverseitige Crowdsourcing-Infrastruktur sollte aus mehreren losen Komponenten bestanden werden. Jede Komponente hat eine bestimmte Funktion. Nach Anforderungen

[10] Crowdsourcing-Infrastruktur ist eine Android Applikation, ihre Komponente Adresse und Name wird nach Installieren festgelegt.

aus Entwicklern könnten unterschiedlichen Komponenten dynamisch ausgewählt und kombiniert werden.

Die Kommunikation zwischen Crowdfunding-Server und Client wird durch ein Web Protokoll realisiert. Das Web Protokoll sollte allgemein sein, d.h. das Protokoll von meisten mobilen Geräten unterstützt wird. Um die Crowdsourcing-Infrastruktur einfach zu implementieren, muss asynchrone Kommunikation zumindest unterstützen.

Die meisten Daten in Crowdsourcing, z.B. Crowdsourcing-Aufgaben, Crowdsourced-Daten, werden auf Serverseite gespeichert. Deshalb sollte Serverseitige Crowdsourcing-Infrastruktur die Operationen nicht nur auf Clientseite sondern auch auf Serverseite unterstützen, z.B. Crowdfunder kann eine Crowdsourcing-Aufgabe dynamisch erstellen. Außerdem gibt es besondere Funktionen auf Serverseite, sowie Crowdsourcer-Verwaltung und anonymes Crowdsourced-Daten-Vorlegen. Die Tabelle 3-2 zeigt die Funktionen-Anforderungen auf serverseitige Crowdsourcing-Infrastruktur.

Die Daten-Objekte-Arten in unterschiedlichen Crowdsourcing-Derivaten sind miteinander unterschiedlich. Deshalb kann man die Daten-Objekte in Crowdsourcing-Infrastruktur entwerfen, z.B. die Attribute in Crowdsourcing-Daten-Objekt dynamisch definiert werden könnten. Das ist auch eine Voraussetzung für Wiederverwendung einer generellen Crowdsourcing-Infrastruktur.

Operation	Benötige Daten-Objekte
Operation von Crowdsourcern unterstützen	Aufgaben, Crowdsourced-Daten, Crowdsourcing-Anreiz
Crowdsourced-Daten lesen	Crowdsourced-Daten
Crowdsourcing-Anreiz verwalten	Crowdsourcing-Anreiz
Crowdsourcer verwalten	Nutzer-Informationen
Crowdsourced-Daten anonym vorlegen	Vorlegen-Information

Tabelle 3-2: Serverseitige Funktionen-Anforderungen der generellen Crowdsourcing-Infrastruktur

3.4 Zusammenfassung

In diesem Kapitel wird zuerst eine grundlegende Crowdsourcing Definition gestellt. Im Vergleichen zu anderen Crowdsourcing-Definitionen konzentriert sich diese Definition besonders auf die grundlegende Operationen und Daten-Objekte in Crowdsourcing. Die generelle Crowdsourcing-Infrastruktur basiert sich auf diese Definition.

Danach werden zwei Crowdsourcing-Derivate sowie FAF und NSÜ vorgestellt. Abhängig von diesen Crowdsourcing-Derivaten wird eine ausführliche Benutzer Szene erstellt. Abhängig von dieser Benutzer Szene einige Anforderungen für Crowdsourcing-Infrastruktur werden analysiert vorgestellt. Um einfach diese Anforderungen im verfolgenden Kapitel zu zitieren, werden sie am nächsten wieder aufgezahlt und nummeriert.

1. Anforderungen der Crowdsourcing Bibliothek in Crowdsourcing-Derivaten
    a. Als eine Bibliothek in jedem Crowdsourcing-Derivat wieder verwendet werden.
    b. Dadurch unterschiedlichen Anfragen für Crowdsourcing-Infrastruktur-Service generiert, gesandt werden.
    c. Unterschiedliche Typen und Formate von Crowdsourced-Daten unterstützen.
    d. Dadurch die Ergebnisse von Crowdsourcing-Infrastruktur-Service erfasst und erkennt werden.
    e. Davon individuelle Schnittstellen für Verarbeitung der Service-Ergebnisse angeboten werden.
2. Anforderungen für clientseitige Crowdsourcing-Infrastruktur
    a. Crowdsourcing-Infrastruktur von Crowdsourcing-Derivaten in einem mobilen Gerät getrennt sollte.
    b. Crowdsourcing-Infrastruktur als einen Service spielen.
    c. Einzige Instanz von Crowdsourcing-Infrastruktur in Laufzeit in mobilen Geräten garantieren.
    d. Mehrere Aufrufe aus unterschiedlichen Crowdsourcing-Derivaten verwalten.
    e. Automatisch Crowdsourcing-Infrastruktur selbst schließen kann, falls es keine Aufrufe von Crowdsourcing-Derivaten mehr gibt.
    f. Ein Crowdsourcing-Derivat muss direkt den Crowdsourcing-Service aufrufen. Der Service muss die Ergebnisse direkt an dieses Derivat zurücksenden.
3. Anforderungen auf Serverseite
    a. Aus losen Komponenten bestehen.
    b. Unterschiedlichen Daten-Arten unterstützen.
    c. Grundlegende Operationen von Daten-Objekten realisieren.
    d. Unterschiedliche Daten-Objekte in unterschiedlichen Derivate dynamisch definiert werden könnten.
    e. Crowdsourcer verwalten.
    f. Crowdsourcer kann anonym Crowdsourced-Daten vorlegen.
4. Anforderungen für Kommunikation
    a. Kommunikation zwischen Crowdsourcing-Derivat und –Service asynchron sein.
    b. Kommunikation zwischen Client und Server synchron sein.
    c. Allgemeines Web-Protokoll zwischen Client und Server verwenden.

Diese Anforderungen einer Crowdsourcing-Infrastruktur sind von der Definition 10 und zwei Dummy-Crowdsourcing-Derivaten abhängig.

Am nächsten Kapitel wird ein Konzept für eine generelle Crowdsourcing-Infrastruktur vorgestellt.

4 Konzept für die generelle Crowdsourcing-Infrastruktur

Die Grundidee ist, dass zum Sparen bei Systemressourcen in mobilen Geräten nur ein Crowdsourcing-Service mehrere Crowdsourcing-Derivate bedienen sollte. Offensichtlich werden die gleichen Funktionen oder Komponenten von unterschiedlichen Crowdsourcing-Derivaten in diesem Service integriert. Dieser Service sollte komplett von Crowdsourcing-Derivaten abgetrennt werden. Um diesen Service einfach zu verwenden, wird eine Komponente wie Programm-Bibliothek in jedem Crowdsourcing-Derivat eingebettet oder implantiert. Das nicht nur vereinfacht die Benutzung der generellen Crowdsourcing-Infrastruktur, sondern auch spiegelt das Modularisierung Prinzip sowie „*Information Hiding*“ wider. Wie eine Anfrage für den Crowdsourcing-Service generiert wird, ist in dieser Bibliothek versteckt. Um die Crowdsourcing Funktionen, z.B. Userverwaltung, flexible zu unterstützen, wird ein Proxy allein verwendet. Die Kommunikation mit diesem Proxy ist eine fakultative Mode. Die serverseitige Infrastruktur konzentriert sich auf die Verwaltung von Daten, die aus Proxy oder direkt aus Client kommen. In der Abbildung 4-1 wird die Architektur dieser generellen Crowdsourcing-Infrastruktur angezeigt.

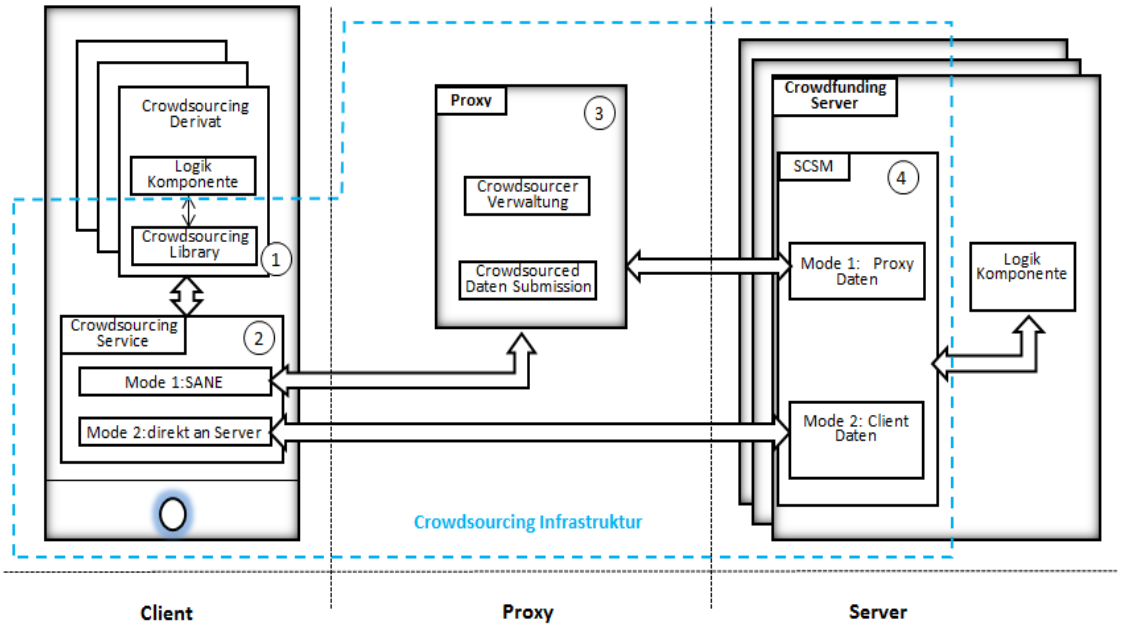


Abbildung 4-1: System Architektur von Crowdsourcing-Infrastruktur

Kommunikation zwischen Crowdsourcing-Derivaten und Crowdfunding Server muss durch *Crowdsourcing-Service* erledigt werden. *Crowdsourcing-Service* kann Multi Crowdsourcing-Derivate bedienen. Nutzer kann Mode 1[11]: sowie Proxy Mode und auch Mode 2: „direkt an den Server Mode“ benutzen. In Mode 1 werden alle Web-Anfragen an Proxy gesandt. In Mode 2 werden alle Web-Anfragen direkt an den Crowdfunding Server gesandt. Ein Server verarbeitet die Anfragen aus Proxy und *Crowdsourcing Service*. Die Tabelle 4-1beschreibt die Grundfunktionen jeder Komponente.

[11] Aus Rücksicht auf Erweiterung wird hier das Wort „Mode“ verwendet und nummeriert, weil mehrere Moden in dieser Infrastruktur verwendet werden könnten, z.B. in Mode 3 ein neuer Proxy mit neuen Funktionen verwendet wird.

Nr.	Komponente Name	Beschreibung
①	Crowdsourcing-Library	• In jedem Crowdsourcing-Derivat implantiert und wiederverwendet wird. • Damit eine *Service-Request* generiert und an die Komponente *Crowdsourcing-Service* gesandt wird. • Danach die Ergebnisse aus *Crowdsourcing-Service* akzeptiert werden.
②	Crowdsourcing Service	• Als eine alleine Applikation in einem mobilen Gerät System laufen, z.B. „singleton“ Crowdsourcing-Service in diese Applikation angeboten werden. • Mehrere Derivate und mehrere Anfragen bedienen kann. • Die Kommunikation zwischen Crowdsourcing-Derivat und Server machen, inkl. Eine *Service-Request* akzeptieren, Web-Anfragen erstellen und senden, die *Service-Response* aus Web-Antwort herausziehen und an *Crowdsourcing-Library* zurück senden. • Zwei Kommunikation-Moden unterstützen, Mode 1: Proxy Mode, Mode 2: direkt an den Server Mode. • Um System Ressourcen zu sparen, kann Service selbst schließen, wenn es keine Anfrage gibt.
③	Proxy	• Als ein *black box* verwendet werden. • Crowdsourcer Anonymous realisieren. Als eine fakultative Kommunikation-Mode für Crowdsourcer. • Hauptsächlich werden die Funktionen sowie Crowdsourcer Verwaltung und Submission Verfolgen angeboten.
④	SCSM(**S**erver-**C**rowd**s**ourcing-**M**odul)	• In jedem Crowdfunding Server eingebettet werden. • Die Web-Anfragen aus Proxy oder direkt aus Crowdsourcer akzeptiert und die Daten darin verwaltet werden. • Die Ergebnisse an den Proxy oder direkt an den Client zurückgesandt werden. • Daten-Objekte verwalten.

Tabelle 4-1: Komponenten Beschreibung von Crowdsourcing-Infrastruktur

In Abbildung 4-2 wird die Komponenten in Crowdsourcing-Infrastruktur mit UML Diagramm beschrieben. Um die Infrastruktur in folgen Abschnitten deutlich vorzustellen, werden die Schnittstellen darin nummeriert. Die nummerierten Schnittstellen werden in der Tabelle 4-2 dargestellt. Zwei Schnittstellen werden von der ganzen Infrastruktur angeboten. Durch die Schnittstelle 1 auf Clientseite kann Crowdsourcer dem Crowdfunder eigenen Crowdsourced Daten abgeben. Durch die Schnittstelle 7 auf Serverseite kann Crowdfunder die Daten verwalten, z.B. Crowdsourced Aufgabe Erstellen und Crowdsourced Daten Lesen. Am Nächsten werden die Komponenten in der Crowdsourcing-Infrastruktur weiter vorgestellt.

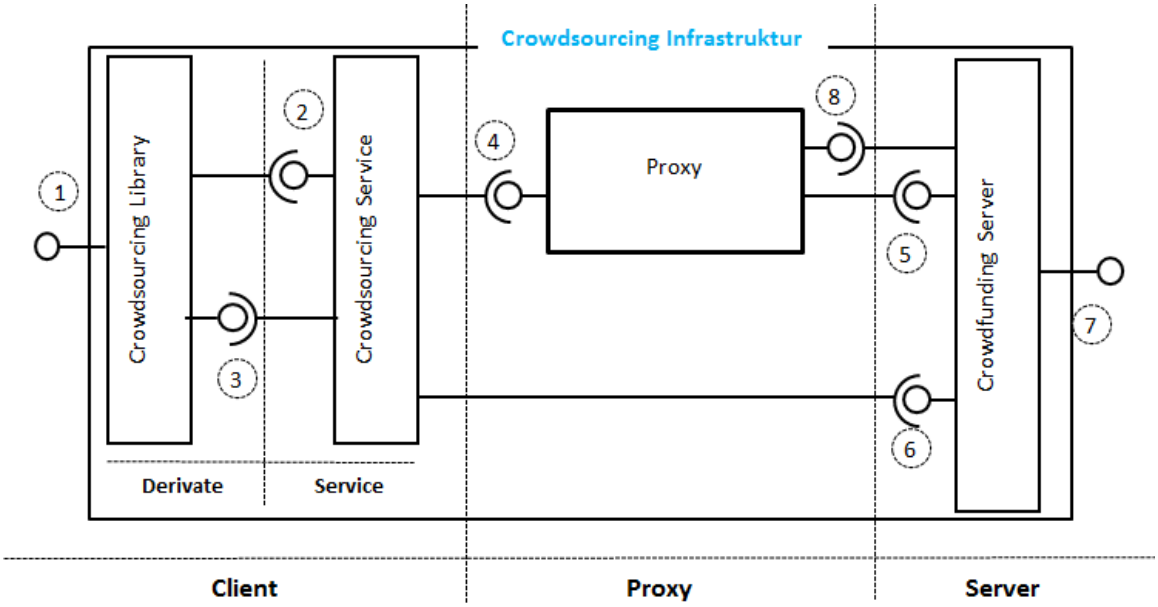


Abbildung 4-2: UML Diagramm von Komponenten

Schnittstelle Nummer	Beschreibung
①	Als Eingang und Ausgang von clientseitigen Daten in der ganzen Crowdsourcing-Infrastruktur. Diese Schnittstelle wird hauptsächlich für Crowdsourcer angeboten.
②	Die Anfrage aus *Crowdsourcing-Library* akzeptieren und verarbeiten.
③	*Crowdsourcing-Library* akzeptiert die Ausgaben aus Crowdsourcing-Service.
④	Durch diese Schnittstelle kann Userverwaltung und Anonymus erledigt werden.
⑤	Durch diese Schnittstelle speichert Server die Daten aus Proxy.
⑥	Durch diese Schnittstelle speichert Server die Daten direkt aus Client.
⑦	Als Eingang und Ausgang von serverseitigen Daten in der ganzen Crowdsourcing-Infrastruktur. Diese Schnittstelle wird hauptsächlich für Crowdfunder angeboten.
⑧	Für die Nachfragen aus Server über Quelle von Crowdsourced-Daten.

Tabelle 4-2: Beschreibung von Schnittstellen in Crowdsourcing-Infrastruktur

4.1 Clientseitige Crowdsourcing-Infrastruktur

Die clientseitige Crowdsourcing-Infrastruktur besteht aus zwei Komponenten, sowie *Crowdsourcing-Library* und *–Service*. Die beiden laufen gleichzeitig in einem Client sowie in einem mobilen Gerät. Die Hauptaufgabe von der clientseitigen Crowdsourcing-Infrastruktur ist, durch Schnittstelle 1 eine Web-Anfrage zu generieren und senden, danach die dazu gehörte Web-Antwort aus Server zu verarbeiten.

Eine ausführliche Architektur von der clientseitigen Crowdsourcing-Infrastruktur wird in der Abbildung 4-3 angezeigt. Die Nummern darin beschreiben den Ereignisablauf auf Clientseite. Der Ereignisablauf wird in der Tabelle 4-3 dargestellt.

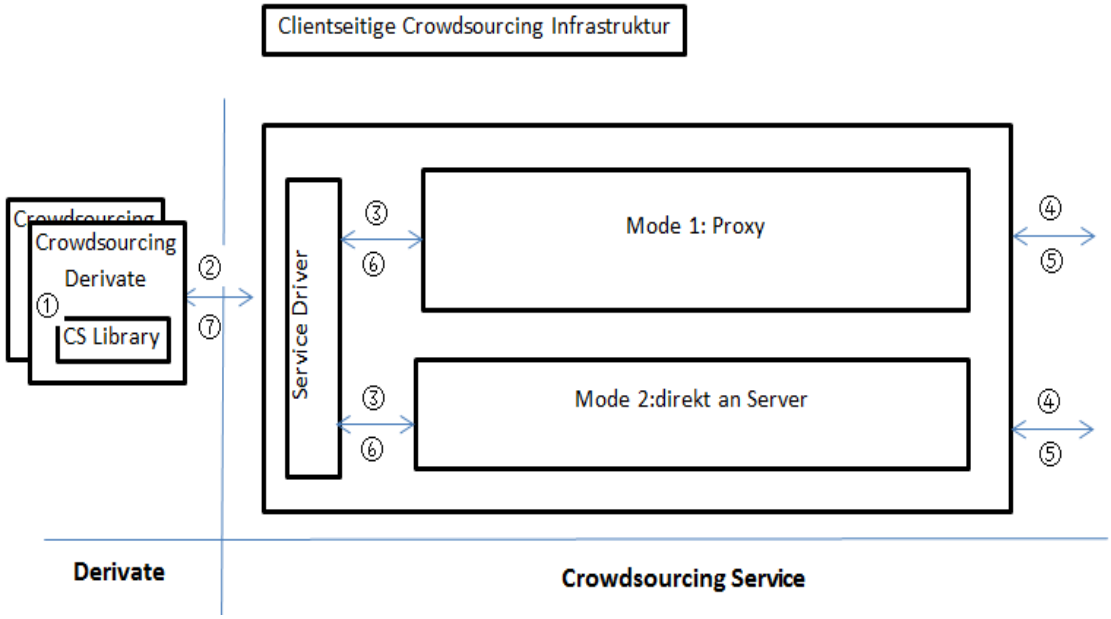


Abbildung 4-3: Architektur von der Komponente *Crowdsourcing-Service*

Nr.	Ereignis	Nr.	Ereignis
①	Crowdsourced-Daten erstellen	②	*Crowdsourcing-Service* aufrufen
③	Eine Kommunikation-Mode auswählen	④	Web-Anfragen senden
⑤	Web-Antworten bekommen	⑥	Informationen in Web-Antworten ausziehen
⑦	Ergebnisse zurücksenden		

Tabelle 4-3: Ereignisse in Komponente *Crowdsourcing-Service*

Am nächsten werden die zwei Komponenten sowie *Crowdsourcing-Library* und *Crowdsourcing-Service* in Details vorgestellt.

4.1.1 Crowdsourcing-Library

Crowdsourcing-Library ist eine Komponente, die in jedem Crowdsourcing-Derivat eingebettet wird. Für die ganze Infrastruktur bietet die Komponente *Crowdsourcing-Library* den Eingang und Ausgang von clientseitigen Daten, deshalb spielt sie sehr wichtige Rolle. Ohne *Crowdsourcing-Library* kann ein *Crowdsourcing-Derivat* die Infrastruktur nicht verwenden. *Crowdsourcing-Library* bietet hauptsächlich Schnittstellen 1 und 3 in der Abbildung 4-2. Am nächsten werden die Operationen in Schnittstelle 1 vorgestellt.

Operation:	Create_Service_Req(*R_Adress, Back_Adress, Req_ID, S_Para*);
Beschreiben:	Eine *Service-Request* für Komponente *Crowdsourcing-Service* erstellen.
Anbieter	Komponente: *Crowdsourcing-Library*
Eingaben:	
Name	Beschreibung
Back_Adress	Die Adresse von Operation Recieve_Res(Req_ID, S_Res) in Schnittstelle 3. Durch diesen Parameter ruft *Crowdsourcing-Service* zurück auf. Typischerweise besteht dieser Parameter sich aus der Adresse von einem Derivat und dem Namen von Operation Recieve_Res(Req_ID, S_Res).
R_Adress	eine Adresse von einem *Receiver*, der die *Service-Response* akzeptieren und verarbeiten kann. Durch diese Adresse wird ein *Receiver* von *Crowdsourcing-Library* aufgerufen[12]. Dieser Parameter wird nicht in *Service-Request* eingepackt.
Req_ID	Eine Identität von *Service Request*. Diese Parameter wird von Entwickler dynamisch definiert, die unterschiedlichen *Service-Response* zu unterschieden. Der Typ von *Req_ID* könnte ein Grundtyp sein, z.B. *„int“*. Der Wert von *Req_ID* wird ohne Veränderung bei *Crowdsourcing-Service* zurückgesandt.
S_Para	Paarweise Parameter für *Crowdsourcing Service*.
Ausgaben:	None
Exception:	None

Tabelle 4-4: Darstellung von Operation: Create_Service_Req(*R_Adress, Back_Adress, Req_ID, S_Para*);

Das Ziel von dieser Operation ist, ein *Service-Request* zu erstellen. Alle Anfragen-Informationen für *Crowdsourcing Service* werden in diesem Objekt eingepackt. Nach Modularisierung Prinzip „*Information Hiding*“ wird das Objekt *Service-Request* in *Crowdsourcing-Library* versteckt. Ein Entwickler kann nicht das Objekt zugreifen.

Service-Request ist ein Objekt für Kommunikation zwischen *Crowdsourcing-Library* und –*Service* in einem Mobile Gerät, deshalb ist es von einem Mobile OS abhängig, z.B. *Service-Request* in Android Mobile OS ist eine „*Intent*“ Klasse, weil „*Intent*“ Klasse zur Kommunikation zwischen unterschiedlichen Komponenten in Android OS verwendet wird.

Es gibt einige Voraussetzungen für diese Operation. Um die *Service-Response* zu erfassen, muss ein *Receiver* von Entwickler vordefiniert werden. Die Eingaben von einem *Receiver* muss *Req_ID* und *S_Res* sein. Eingabe *Req_ID* ist gleich wie der Parameter *Req_ID* in der Tabelle 4-4. Eingabe *S_Res* präsentiert eine *Service-Response* aus *Crowdsourcing Service.* Durch Eingabe *Req_ID* kann Empfänger weißen, zu welcher schon gesandten *Service-Request* die Eingabe *S_Res* gehört.

Um eine *Service-Response* zu erkennen, muss der Parameter *Req_ID* einzigartig in unterschiedlichen *Service-Request*s eingestellt werden. Aber das ist schwer, zu realisieren. Zuerst muss *Receiver* jederzeit den neuen Wert von jedem *Req_ID* erfahren. Zweitens

[12] Es wird angenommen, dass eine Objekt oder Funktion in einem Mobile OS durch ihre Adresse aufgerufen kann.

muss die Einzigartigkeit von *Req_ID* geprüft werden. Es gibt Probleme in Prüfen von *Req_ID*, z.B. welche *Req_ID* wieder verwendet werden können. Aber gibt es eine einfache Lösung dafür. Die Entwickler müssen die *Req_ID* kümmern. Normalerweise wird eine *Service-Request* wegen eines Benutzer-Ereignisses generiert, deshalb können Entwickler eine von einem Benutzer-Ereignis abhängige *Req_ID* definieren, z.B. ein bestimmtes Button-Drücken.

Operation:	Add_Data(*Data*);
Beschreibung:	Die *Crowdsourced Data* werden in einer *Service-Request* hinzugefügt.
Anbieter	Komponente: *Crowdsourcing-Library*
Eingaben:	
Name	Beschreibung
Data	Dieser Parameter präsentiert die Crowdsourced-Daten für die Schnittstellen auf Server- oder Proxy-Seite.
Ausgaben:	None
Exception:	None

Tabelle 4-5: Darstellung von Operation: Add_Crowdsourced_Data(Csed_Data);

Durch die Operation in der Tabelle 4-5 können die Crowdsourced-Daten in eine *Service-Anfrage* addiert werden. Um die Crowdsourcing-Infrastruktur flexible zu verwenden, könnten unterschiedlichen Datentypen unterstützt werden, z.B. *LinkedHashTable* und *String*.

Weil die *Crowdsourced-Data* auf Serverseite gespeichert und verarbeitet werden, sollten ihre Typen und Formate die Schnittstellen auf Serverseite anpassen, d.h. die *Crowdsourced-Data* in dieser Operation müssen wellgeformt werden. Die Komponenten *Crowdsourcing-Library* und *–Service* prüfen das Format von *Crowdsourced-Daten* nicht.

Operation:	Send_Req(*S_Address*);
Beschreibung:	Eine *Service-Request* wird an den *Crowdsourcing-Service* gesandt.
Anbieter	Komponente: *Crowdsourcing-Library*
Eingaben:	
Name	Beschreibung
S_Address	Die Ziel-Adresse einer *Service-Request* erklären, z.B. einen File Pfad oder URI von *Crowdsourcing Service*. Dieser Parameter wird von Mobile System abgehängt, weil *Crowdsourcing-Service* als eine alleine Applikation in einem System installiert. Dieser Parameter kann als einer Konfiguration Wert von Infrastruktur in *Crowdsourcing-Library* gespeichert werden
Ausgaben:	None
Exception:	None

Tabelle 4-6: Darstellung von Operation: Send_Req(*S_Address*);

Diese Operation Tabelle 4-6 ist von einem Mobile OS abhängig. Normalerweise werden bestimmte Schnittstellen in einem Mobile OS für die Kommunikation zwischen Komponenten angeboten, deshalb verwendet diese Operation die von System angebotenen Schnittstellen für Senden einer *Service Request*.

Die Schnittstelle 1 wird dem Crowdsourcer oder Entwickler angeboten. Das Ziel von Schnittstelle 1 ist, eine *Service-Request* für den *Crowdsourcing-Service* zu erstellen und senden. Das Objekt *Service-Request* wird in *Crowdsourcing-Library* versteckt. Alle Parameter für *Crowdsourcing-Service* werden in diesem Objekt eingepackt.

Im Vergleichen zu Schnittstelle 1 ist Schnittstelle 3 eine interne Schnittstelle in der Crowdsourcing-Infrastruktur. Schnittstelle 3 akzeptiert die *Service-Response* aus *Crowdsourcing Service* und wird nur von dem *Crowdsourcing-Service* aufgerufen. In Schnittstelle 3 gibt es nur eine Operation, die in der Tabelle 4-7 dargestellt wird.

Operation:	Recieve_Service_Res(Req_ID, S_Res);
Beschreibung:	Die aus *Crowdsourcing-Service* zurückgesandten Ergebnisse akzeptieren. Die Parameter sowie *Req_ID* und *S_Res* können weiter übertragen werden, z.B. in eine vordefinierte Schnittstelle.
Anbieter	Komponente: *Crowdsourcing-Library*
Eingaben:	
Name	Beschreibung
Req_ID	Gleich wie der Parameter *Req_ID* in der Tabelle 4-4.
S_Res	Die Ergebnisse von *Service-Requests* in *Service-Response* eingepackt werden.
Ausgaben:	None
Exception:	
Name	Beschreibung
NullS_Res	Die *Service-Response* ist *null*.

Tabelle 4-7: Darstellung von Operation: Recieve_Service_Res(Req_ID, S_Res);

4.1.2 Crowdsourcing Service

Die Komponenten in der Abbildung 4-3 werden in der Tabelle 4-8 dargestellt. Die Komponente *Service Driver* ist hauptsächlich für die Kommunikation zwischen *Crowdsourcing-Library* und *–Service* verantwortlich. Durch Komponente Mode 1 kann eine *Service-Request* in eine Web-Anfrage, die Web-Schnittstellen bei Proxy anpasst, gewechselt werden. Mode 2 ist einfacher als Mode 1, weil durch Komponente Mode 2 die Web-Anfrage direkt an die eingegebene Server Adresse gesandt. Es gibt fast keine komplexe Verarbeitung in Mode 2.

Obwohl beide Komponenten sowie Mode 1 und 2 für die gleichen Funktion sowie Web-Kommunikation verantwortlich sind, können sie nicht in eine Komponente integriert werden, weil die Schnittstellen bei Proxy und Server nicht gleich sein könnten. Zusätzlich sollen die Komponenten nach Modularisierung-Prinzipien los sein. Wenn der Proxy verändert würde, würde nur Komponente Mode 1 sich verändern. Das zeigt auch die Erweiterbarkeit dieser Crowdsourcing-Infrastruktur. In der Zukunft könnte eine Komponente mit unterschiedlichen Funktionen alleine entwickelt und einfach hinzugefügt werden.

Komponente	Beschreibung
Service Driver	Die aus *Crowdsourcing-Library* generierte *Service-Request* akzeptieren und die Parameter darin analysieren. Die Ergebnisse aus *Mode 1* und *2* an die Schnittstelle bei *Crowdsourcing-Library* senden.
Mode 1: Proxy	Eine Web-Anfrage generiert und an den Proxy gesandt werden. Die Web-Antworten aus dem Proxy verarbeitet und die Information daraus weiter an den *Service Driver* übertragen werden.
Mode 2: direkt an Server Mode	Nach den in *Service-Request* angeboten Parametern eine Web-Anfrage generiert und direkt an den Server gesandt wird. Die Web-Antwort verarbeiten und weiter an den *Service Driver* übertragen.

Tabelle 4-8: Beschreibung von Komponenten in *Crowdsourcing-Service*

Von der Komponente *Crowdsourcing-Service* in der Abbildung 4-2 wird nur eine Schnittstelle 2 angeboten. Durch diese Schnittstelle wird eine ankommende *Service-Request* verarbeitet. Es gibt einige Operationen in dieser Schnittstelle.

Operation:	Receive_Service_Req(S_Req);	
Beschreibung:	Eine *Service-Request* verarbeiten, z.B. eine Mode auswählen.	
Anbieter:	Komponente: Service Driver	
Eingaben:		
Name	Beschreibung	
S_Req	Die Anfrage für *Crowdsourcing Service.*	
Ausgaben:		
Name	Beschreibung	
S_Res	Die Antwort aus *Crowdsourcing Service.*	
Exception:	Name:	Beschreibung
	ModeNotSupport	Die eingegebene Mode wird nicht unterstützt.
	NoWebConnection	Es gibt keine Verbindung mit dem Netzwerk.

Tabelle 4-9: Darstellung von Operation: Receive_Service_Req(S_Req);

Diese Operation Tabelle 4-9 akzeptiert die ankommende *Service-Request* und prüft die Parameter darin, z.B. die Parameter für Auswählen einer Mode. Das vorhandene Konzept von der generellen Crowdsourcing-Infrastruktur stellt nur zwei Kommunikation-Moden vor, aber in der zukünftigen Erweiterung könnten mehrere Moden addieren. Für die Mode-Parameter gibt es eine Exception sowie „*ModeNotSupport*“.

Crowdsourcing-Service macht die Kommunikation zwischen Client und Server. Bevor eine Web-Anfrage gesandt wurde, wird Netzwerkverbindung zuerst geprüft. Wenn es keine Netzwerkverbindung in einem Mobile Geräte gibt, wird eine Exception „*NoWebConnection*“ ausgegeben. Natürlich werden die Informationen einer Exception in eine *Service-Response* eingepackt und an *Crowdsourcing-Library* zurückgesandt.

Diese Operation in Tabelle 4-10 wird von der Operation *Receive_Service_Req(S_Req)* aufgerufen. Diese Operation ist hauptsächlich für die Web-Kommunikation zwischen Client und Server verantwortlich.

Operation:	Communicate_Proxy(*Proxy_Para*, *Proxy_URL*);
Beschreibung:	Eine Web-Anfrage wird erstellt und an Proxy gesandt. Danach wird die Web-Antwort aus Proxy verarbeitet.
Anbieter:	Komponente: Mode 1 Proxy
Eingaben:	
Name	Beschreibung
Proxy_Para	Paarweise Parameter für Proxy.
Proxy_URL	Die Adresse von Proxy.
Ausgaben:	
Name	Beschreibung
S_Res	Die Ergebnisse aus Proxy werden in diesem Objekt eingepackt.
Exception:	
Name	Beschreibung
ConnectionTimeExceeded	Verbindungszeit überschritten.

Tabelle 4-10: Darstellung von Operation: Communicate_Proxy(Csed_Data);

Crowdsourcing-Service läuft ähnlich wie ein „singleton“ Service. Er kann nicht kontinuierlich auf eine Web-Antwort warten, sonst werden die nachfolgenden *Service-Requests* stark verzögert. Deshalb gibt es eine *Exception* sowie „Verbindungszeit Überschritten“, falls *Crowdsourcing-Service* nach lange Zeit keine Web-Antwort erhält.

Operation:	Comunicate_direkt_Server(*URL, Methode, Data*);
Beschreibung:	Eine Web-Anfrage wird erstellt und direkt an Server gesandt. danach wird die Web-Antwort aus Server verarbeitet.
Anbieter:	Komponente: Mode 2 direkt an Server
Eingaben:	
Name	Beschreibung:
URL	Die Web Adresse von einem Crowdsourcing Server.
Methode	Eine Methode von Web Protokoll, z.B. „Post“.
Data	Die Daten werden bei dem Server verarbeitet.
Ausgaben:	
Name	Beschreibung
S_Res	Die Antworten aus Server werden in diesem Objekt eingepackt.
Exception:	
Name	Beschreibung
ConnectionTimeExceeded	Verbindungszeit überschritten.

Tabelle 4-11: Darstellung von Operation: Comunicate_direkt_Server(*URL, Methode, Data*).

In dieser Operation werden *Crowdsourced-Daten* an einen Server direkt gesandt, deshalb ist die Server-Web-Adresse notwendig. Außerdem verpackt diese Operation *Crowdsourced-Daten* in eine Web-Anfrage, aber der Dateityp und das Datenformat müssen die Anforderung von Server anpassen, z.B. wenn ein Server nur JSON Daten akzeptieren würde, wäre Dateityp Konvertierung in diese Operation notwendig.

4.2 Proxy in Crowdsourcing-Infrastruktur

Die ausführliche Architektur von Proxy wird in der Abbildung 4-4 gezeigt. Die Nummern darin präsentieren die Kommunikation-Ereignisse zwischen Client und Server.

Die Ereignisse werden in der Tabelle 4-12 aufgezählt. Die Komponenten in dieser Architektur werden in der Tabelle 4-8 dargestellt.

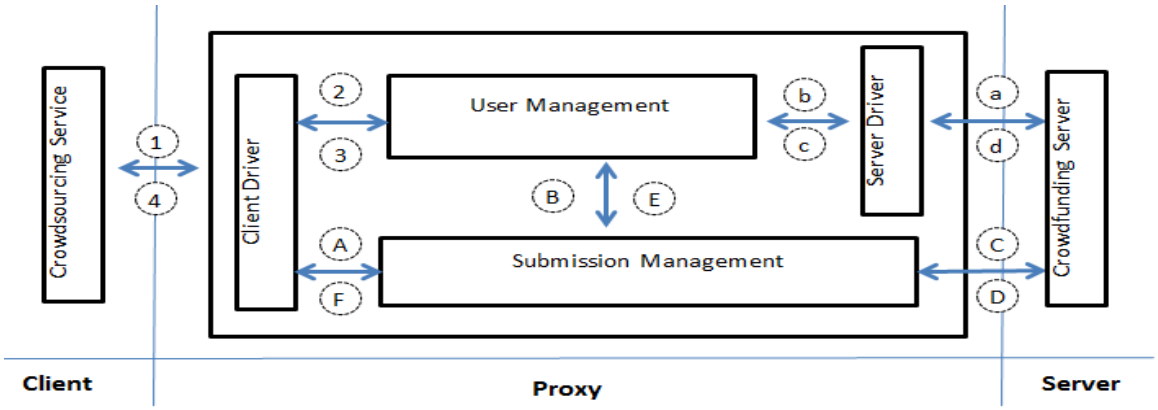


Abbildung 4-4: Architektur von Proxy in Crowdsourcing-Infrastruktur.

Nummer Reihenfolge	Ereignisablauf	Operation
①	Client-Anfrage akzeptieren.	Nutzerverwaltung, z.B. Registrieren
②	User-Informationen verwalten.	
③	Verwaltung-Ergebnisse erstellen	
④	Proxy-Antworten zurück senden	
①	Client-Anfrage akzeptieren.	Crowdsourced-Daten verwalten
A	Crowdsourced-Daten verwalten	
B	Nutzer-Information untersuchen[13]	
C	Proxy-Anfrage an Server senden	
D	Antworten aus Server akzeptieren[14]	
E	Nutzer-Information verändern	
F	Verwalten-Ergebnisse erstellen	
④	Proxy-Antworten zurück senden	
a	Server-Anfragen akzeptieren.	Crowdsourcer nachfragen
b	Crowdsourcer suchen	
c	Crowdsourcer-Information ausgeben	
d	Web-Antworten zurück senden	

Tabelle 4-12: Ereignisse-Ablauf in der Proxy in Crowdsourcing-Infrastruktur

[13] Die Verwaltungen von Crowdsourced-Daten müssen unter einem existierenden Nutzerkonto erledigt werden.
[14] Die *SubmissionIDs* werden unter einem Nutzerkonto gespeichert. Sie werden während Löschen und Erstellung der Crowdsourced-Daten verändert.

Komponente	Beschreibung
Client Driver	Die aus *Crowdsourcing-Service* gesandten *Web Requests* akzeptieren und die Parameter darin analysieren. Die Web Ergebnisse zurücksenden.
User Management	Benutzer verwalten, z.B. „UserCreate“.
Submission Management	Bei jeder Submission wird eine einzigartige *SubmissionID* erstellt. Diese *SubmissionID* wird bei dem entsprechenden Benutzer gespeichert. Die *Crowdsourced Daten* und *SubmissionID* werden zusammen an einen Server gesandt.
Server Driver	Die Web-Anfrage über Crowdsourcer aus Crowdfunding Server akzeptieren.

Tabelle 4-13: Beschreibung von Komponenten in *Proxy*.

Nach dieser Architektur können die Operationen bei Schnittstelle 4 beschrieben werden.

Operation:	Receive_Web_Request(*Web_Request*);
Beschreibung:	Die aus *Crowdsourcing-Service* ankommenden Web-Anfragen verarbeiten. Die Ergebnisse von der Verarbeitung zurückgeben.
Anbieter:	Komponente: Client Driver in Proxy
Eingaben:	
Name	Beschreibung
Web_Request	Eine Web-Anfrage aus *Crowdsourcing Service* gesandt werden. Darin gibt es alle Parameter für Proxy.
Ausgaben:	
Name	Beschreibung
Web_Response	Eine Web-Antwort für *Crowdsourcing Service*. Darin werden die Ergebnisse über Crowdsourcer-Anfrage eingepackt.
Exception:	
Name	Beschreibung
Servererro	Die Server Fehler beschreiben, z.B. Server-Ausfall.

Tabelle 4-14: Darstellung von Operation: Receive_Web_Request(*Web_Request*);

Diese Operation in der Tabelle 4-14 bei Proxy verarbeitet die aus *Crowdsourcing Server* gesandten Web-Anfragen. Nach den Parametern in Web-Anfrage wird eine Operation ausgewählt.

Operation:	User_Management(*U_Action, U_Parameter*);
Beschreibung:	Die User Information verwalten.
Anbieter:	Komponente: User Management
Eingaben:	
Name	Beschreibung
U_Action	Eine Operation von Benutzer-Konto, z.B. „Create“.
U_Parameter	Parameter für die Nutzerverwaltung.
Ausgaben:	

Name	Beschreibung
Success	Die Operation von Benutzer wurde schon erfolgreich erledigt.
Fail	Gründen für fehlgeschlagene Operationen, z.B. ein „*User_Name*“ bei einer Operation „*UserDelete*“ nicht existiert.

Exception:	
Name	Beschreibung
Internerro	Die internen Fehler beschreiben, z.B. Datenbank-Ausfall.

Tabelle 4-15: Darstellung von Operation: User_Management(*Cs,U_Action, U_Parameter*).

Diese Operation in der Tabelle 4-15 realisiert die Benutzer-Verwaltung. Die zweite Parameter „*U_Parameter*“ ist von dem ersten Parameter abhängig, d.h. der Parameter „*U_Action*“ zuerst bei dem Proxy analysiert werden. Die erstellten Benutzer-Information werden nur in dem Proxy gespeichert.

Operation:	Submission_Management(*Cs, User, Methode, Csed_Data*);
Beschreibung:	Abhängig von Parameter „*Cs*“ wird eine Web-Anfrage mit *Csed_Data* und *SubmissionID* bei jeder Submission an den entsprechenden Server gesandt. Danach werden die Antworten aus Server weiter an den Client zurückgesandt. Die *SubmissionID* wird unter dem eingegebenen Parameter *User* gespeichert.
Anbieter:	Komponente: Submission Management
Eingaben:	
Name	Beschreibung
Cs	Einen Crowdfunding Server präsentieren, typischerweise durch den Name eines Crowdfunding Servers.
User	Einen Crowdsourcer präsentieren, z.B. durch einen Namen von einem Benutzer.
Methode	Die Operation für die „*Csed_Data*“ auf Serverseite.
Csed_Data	Eine Lösung für eine Crowdsourcing-Aufgabe sein. Diese Daten werden an den Crowdfunding Server gesandt.
Ausgaben:	

Name	Beschreibung
Success	Die Operation von Benutzer wurde schon erfolgreich erledigt.
Fail	1. Informationen von fehlgeschlagenen Operationen bei dem Proxy, z.B. „user not exists“ oder „Crowdfunding Server not exists“. 2. Gründen für fehlgeschlagene Operationen auf Serverseite, z.B. die *Crowdsourced Data* eine falsche „*AufgabeID*“ enthält. Natürlich sollen diese Informationen aus Server quellen.
Exception:	

Name	Beschreibung
Internerro	Die internen Fehler beschreiben, z.B. Datenbank-Ausfall.

Tabelle 4-16: Darstellung von Operation: Submission_Management(*Cs, User, Methode, Csed_Data*).

Diese Operation realisiert die Submission von Crowdsourced Daten. Der erste Parameter „*Cs*“ wird zur Crowdfunding-Server-Navigation verwendet, weil der Proxy an unterschiedlichen Crowdfunding Server angeschlossen wird. Eine „*SubmissionID*“ wird einzigartig und nur während einer Submission generiert. Die „*Csed_Data*“ werden nicht

bei dem Proxy gespeichert und ohne Veränderung an den Server gesandt. Das passt ein Prinzip sowie „Crowdfunder besitzt Crowdsourced Daten“ an. Falls ein Crowdsourcer eigene Crowdsourced Daten verwalten möchte, z.B. Crowdsourced-Daten Lesen, findet Proxy die entsprechenden *SubmissionIDs* unter diesem Crowdsourcer zuerst, dann werden die Crowdsourced Daten aus Server nach diesen *SubmissionIDs* nachgefragt, schließlich kann der Crowdsourcer eigene Crowdsourced Daten aus Proxy bekommen.

Durch „*SubmissionID*“ kann die Funktion sowie Crowdsourcer Anonymus realisieren. Bei Crowdsourced-Daten-Verwaltungen erhält ein Server nur Crowdsourced Data plus „*SubmissionID*“. Es gibt keine Informationen über Crowdsourcer darin. Deswegen sind Crowdsourcer unsichtbare auf Serverseite. Jedoch ist es ein Problem, dass Crowdfunder die Crowdsourcer nicht motivieren können, weil Crowdfunder die Crowdsourcer nicht weißen. Durch „*SubmissionID*“ kann das Problem gelöst werden. Durch Nachfragen von einer „*SubmissionID*“ bei Proxy kann Server den Inhaber von Crowdsourced Daten erfahren, dann kann Crowdfunder diesen Crowdsourcer anreizen. Die Operation in Schnittstelle 8 kann das Nachfragen über Crowdsourcer erledigen. Diese Operation wird in der Tabelle 4-17 dargestellt.

Operation:	Crowdsourcer_Querry(*Cs, SubmissionID*);	
Beschreibung:	Die User Information verwalten.	
Anbieter:	Komponente: Server Driver in Proxy	
Eingaben:		
Name	Beschreibung	
Cs	Einen Crowdfunding Server präsentieren, typischerweise durch den Name eines Crowdfunding Servers.	
SubmissionID	Einzigartige Identität für eine Submission.	
Ausgaben:		
Name	Beschreibung	
User	Identität eines Benutzers.	
Not exists	Der Besitzer von der eingegebenen *SubmissionID* existiert nicht.[15]	
Exception:		
Name	Beschreibung	
Internerro	Die internen Fehler beschreiben, z.B. Datenbank-Ausfall.	

Tabelle 4-17: Darstellung von Operation: Crowdsourcer_Querry(*Cs, SubmissionID*).

Das „Anreiz-Mechanismus“ in Crowdsourcing ist ein kompliziertes Thema. Es gehört häufig zu Logik-Komponenten nicht zu dieser Crowdsourcing-Infrastruktur. Das Ausführen von „Anreiz“ basiert auf die Ergebnisse von Crowdsourced-Daten Analysieren, z.B. welche und wie die beste Crowdsourced-Daten ausgewählt werden sollen. Außerdem muss das Anreiz-Ausführen von Crowdfunder bestimmt werden. Diese Operation Tabelle 4-17 bietet nur eine grundlegende Funktion dafür, sowie die Crowdsourcer können verfolgt werden.

In der Tat bricht diese Operation den Crowdsourcer-Anonymus. Theoretisch kann ein Server durch Anfragen von allen ankommenden „*SubmissionID*“ alle Crowdsourcer

[15] Dieser Fall würde passieren, wenn ein Benutzer in Proxy gelöscht würde, aber seiner Crowdsourced Daten auf Serverseite nicht gelöscht würden.

erfahren. Eine Lösung dafür ist, dass Proxy den „Anreiz“ ausführt. In diesem Fall könnte Server nur eine Anreiz-Entscheidung an Proxy senden. Diese Operation sollte ähnlich wie *Crowdsourcer_motivate(Cs, SubmissionID, Motivation)* sein. In diesem Fall weiß Server nicht, welcher Crowdsourcer die Crowdsourced Daten abgab und welcher Crowdsourcer schon angereizt wurde. Zusätzlich können die Anreiz-Rekorde einfach unter einem Crowdsourcer-Konto verwaltet werden.

4.3 Serverseitige Crowdsourcing-Infrastruktur

Die Daten spielen eine sehr wichtige Rolle in Crowdsourcing, z.B. Crowdsourced Daten und –Information. Normalerweise werden diesen Daten von Crowdsourcer auf Clientseite erstellt und von Crowdfunder auf Serverseite gesammelt, verarbeitet und analysiert. Deshalb ist die Daten-Verwaltung eine gemeinsame Anforderung auf Serverseite. Um unterschiedlichen Crowdfunding Server zu unterstützen, soll eine generelle Crowdsourcing-Infrastruktur die Verwaltung-Schnittstellen von unterschiedlichen Daten anbieten. Aber die Daten in unterschiedlichen Derivaten sind vielfaltig. Um die gemeinsamen Eigenschaften zu finden, werden die Daten zuerst modelliert.

4.3.1 Modellierung von Daten in Crowdsourcing

Die Abbildung 4-5 zeigt die Beziehungen zwischen Daten Objekten in Crowdsourcing. Diese Modellierung basiert auf Crowdsourcing-Definition 10 auf Seite 23.

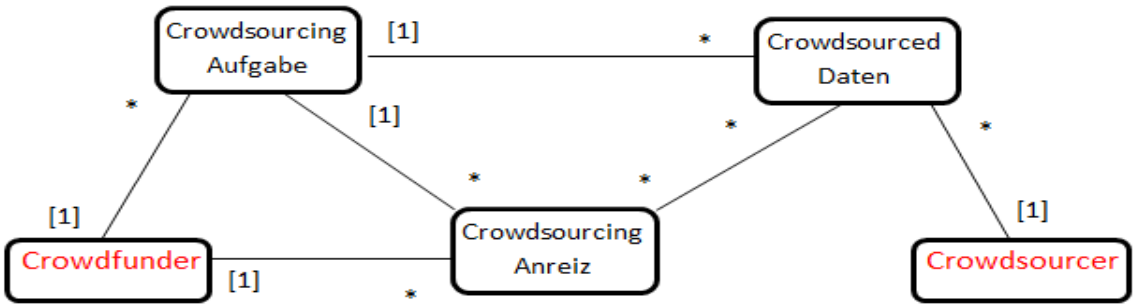


Abbildung 4-5: Beziehungen zwischen Crowdsourcing Daten Objekt

Eine *Crowdsourcing-Aufgabe* kann unbestimmte Anzahl von *Crowdsourcing Anreiz* haben, weil unterschiedlichen Anreizen für eine Aufgabe angeboten werden können. Aber ein *Crowdsourcing-Anreiz* gehört nur zu einer *Crowdsourcing-Aufgabe*. Eine *Crowdsourcing-Aufgabe* kann unbestimmte Anzahl von *Crowdsourced Daten* bekommen. Tatsächlich spielen Crowdsourced Daten ähnlich wie Lösungen[16] oder Ergebnisse einer Aufgabe. Ein *Crowdsourced Daten* gehört nur zu einer *Crowdsourcing-Aufgabe*.[17] Ein *Crowdsourced Daten* kann theoretisch unbestimmte Anzahl von *Crowdsourcing Anreizen* bekommen. Ein *Anreiz* kann für mehrere *Crowdsourcing Daten* bezahlt werden. Ein Crowdsourcer kann mehrere *Crowdsourced Daten* abgeben, aber eine *Crowdsourced Daten* gehört nur zu einem *Crowdsourcer*. Diese Beziehung zwischen *Crowdsourcer* und *Crowdsourced Daten* ist nicht nur für *Crowdfunder* und *Crowdsourcing-Aufgabe*,

[16] In dieser Diplomarbeit sind *Crowdsourced Daten* und *Lösung* synonym in Crowdsourcing.
[17] Eine Crowdsourced Daten kann nur einmalige in einer Aufgabe verwendet werden. In einigen Forschung, kann sie wieder verwendet. Wenn die Crowdsourced-Daten wieder verwendet werden können, werden die Kosten von Crowdsourcing reduziert.

sondern auch für *Crowdfunder* und *Crowdsourcing Anreiz* gültig. Die Abbildung 4-6 zeigt die Entity Diagramm von ganzen Objekten an.

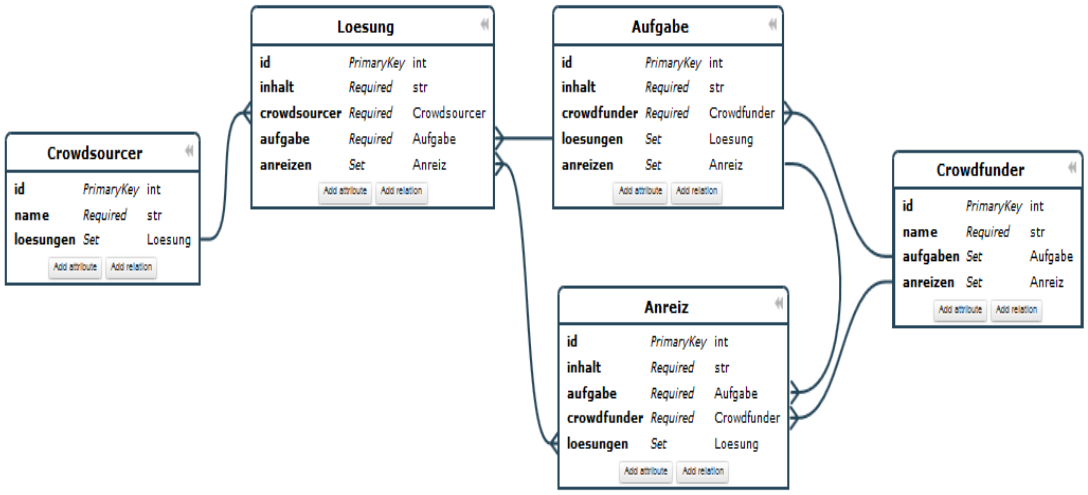


Abbildung 4-6: Entity Diagramm von Crowdsourcing Daten

Diese drei Data-Objekte sowie *Lösung*, *Aufgabe* und *Anreiz* haben gemeinsamen Eigenschaften, z.B. in diesen drei Objekten nur Informationen gespeichert werden und es keine Operationen oder Funktionen darin gibt. Das bringt einige Vorteile, z.B. in Datenspeichern kann JDO [18] verwendet werden, in Datenaustausch zwischen Anwendungen kann JSON verwendet werden.

Das Objekt *Crowdsourcer* und *Crowdfunder* sind von den vorigen Daten Objekten verschieden. Sie können die vorigen Objekte verwalten, z.B. Operation CRUD einer Lösung. Hier die Beziehung zwischen *Crowdsourcer* und Daten Objekt bedeutet nur „Haben".

Die Daten Beziehungen sind sehr wichtig für die Daten-Operationen, z.B. bei Erstellung einer *Crowdsourcing Daten* muss die Daten-Konsistenz geprüft werden, ob die angebotene *Aufgabe ID* existiert. Bei Löschen einer *Aufgabe* müssen alle zu deren gehörte *Crowdsourcing Daten* gelöscht werden. Abhängig von richtigen Daten-Beziehungen kann die Richtigkeit und Konsistenz von Daten Objekten garantiert werden.

4.3.2 Konzept von Server-Crowdsourcing-Modul(SCSM)

Die Serverseitige Crowdsourcing-Infrastruktur sowie SCSM bietet drei Schnittstellen in der Abbildung 4-2 an. Die Schnittstelle 5 und 6 werden den Crowdsourcern angeboten, z.B. Crowdsourcer eigene Daten durch Schnittstelle 5 und 6 bei Crowdfunder abgeben kann. Schnittstelle 7 wird für die Crowdfunder angeboten, z.B. Crowdfunder durch diese Schnittstelle eine Crowdsourcing-Aufgabe erstellen und die Lösungen dafür sammeln kann. Ein allgemeiner Punkt von Schnittstellen 5, 6 und 7 ist, eine Operation für einen

[18] JDO abkürzt für Java Data Object, ist ein offizielle Sun-Spezifikation für von Datenbank unabhängigen Speicherung von Java-Objekt.

bestimmten Datentyp auszuführen. Ein Unterschied zwischen Schnittstellen 5, 6 und 7 ist, wer eine Schnittstelle benutzen kann, z.B. Crowdsourcer nur eine Crowdsourcing-Aufgabe lesen kann, aber Crowdfunder eine Crowdsourcing-Aufgabe erstellen kann.

Die serverseitige Crowdsourcing-Infrastruktur basiert sich auf einige Prinzipien. Diese Prinzipien werden am nächsten vorgestellt.

1. Eine *Ressource* präsentiert einen Crowdsourcing-Datentyp oder eine Datenressource.
2. Eine *Ressource* muss eindeutig sein, typischerweise wird sie durch URI beschriebt.
3. Parameter für jede Operation von Ressourcen muss ausreichende und notwendige Information enthalten.
4. Jede Operation ist zustandslos, d.h. es keine Beziehung zwischen eine Operation und letzte Operation gibt.
5. Während Datenverwaltung werden die Beziehung zwischen Daten-Objekten gefolgt, z.B. die Datenkonsistenz garantieren.
6. Außer erfolgreichen Information sollten die Fehler-Gründen für einer fehlgeschlagene Daten-Operation ausgegeben werden.

Nach diesen Prinzipien können die Schnittstellen 5, 6 und 7 in der Abbildung 4-2 auf serverseitiger Crowdsourcing-Infrastruktur vereinfacht werden. Die Operationen in diesen Schnittstellen können in eine Operation integriert werden. Diese Operation wird in der Tabelle 4-18 dargestellt.

Operation:	DataOperation(*Ressource, Methode, Data*);
Beschreibung:	Die Daten auf Serverseite verwalten.
Anbieter:	Server-Crowdsourcing-Modul
Eingaben:	
Name	Beschreibung
Ressource	Sie präsentiert eine Datenressource. Typischerweise wird diese Datenressource durch eine Adresse beschrieben.
Methode	Die Operation von dieser Ressource.
Data	Parameter für die Operationen.
Ausgaben:	
Name	Beschreibung
Success	Information für erfolgreiche Operationen und Ergebnisse dieser Operation, z.B. Ein Dataobjekt während Lesen.
Fail	Gründen für fehlgeschlagene Operationen, z.B. die eingegeben „*AufgabeID*“ einer „*Lösung*“ nicht existiert[19].
Exception:	
Name	Beschreibung
Internerro	Die internen Fehler beschreiben, z.B. Datenbank-Ausfall.

Tabelle 4-18: Darstellung von Operation: DataOperation(Ressource, Operation, Data);

[19] Eine „Lösung“ muss eine existierende „AufgabeID“ enthalten.

Die grundsätzlichen Daten-Operationen sind CRUD. Die Parameter *Methode* in dieser Operation sollte CRUD sein, aber begrenzt nicht nur auf CRUD. Abhängig von Anforderungen kann dieser Parameter erweitert werden, z.B. „*Methode=analyse*“ bedeutet Analysierung einer Daten-Ressource.

4.4 Kommunikation in Crowdsourcing-Infrastruktur

Am nächsten wird die Kommunikation zwischen Komponenten in der Crowdsourcing-Infrastruktur vorgestellt. Vor Kommunikation-Beginn muss ein Crowdsourcing-Derivat durch *Crowdsourcing-Library* eine *Service-Request* erstellen und einen Ergebnisse-Empfänger definieren. Dieser Empfänger wird von dem *Crowdsourcing-Library* aufgerufen, wenn *Crowdsourcing-Library* eine *Service-Response* erhält.

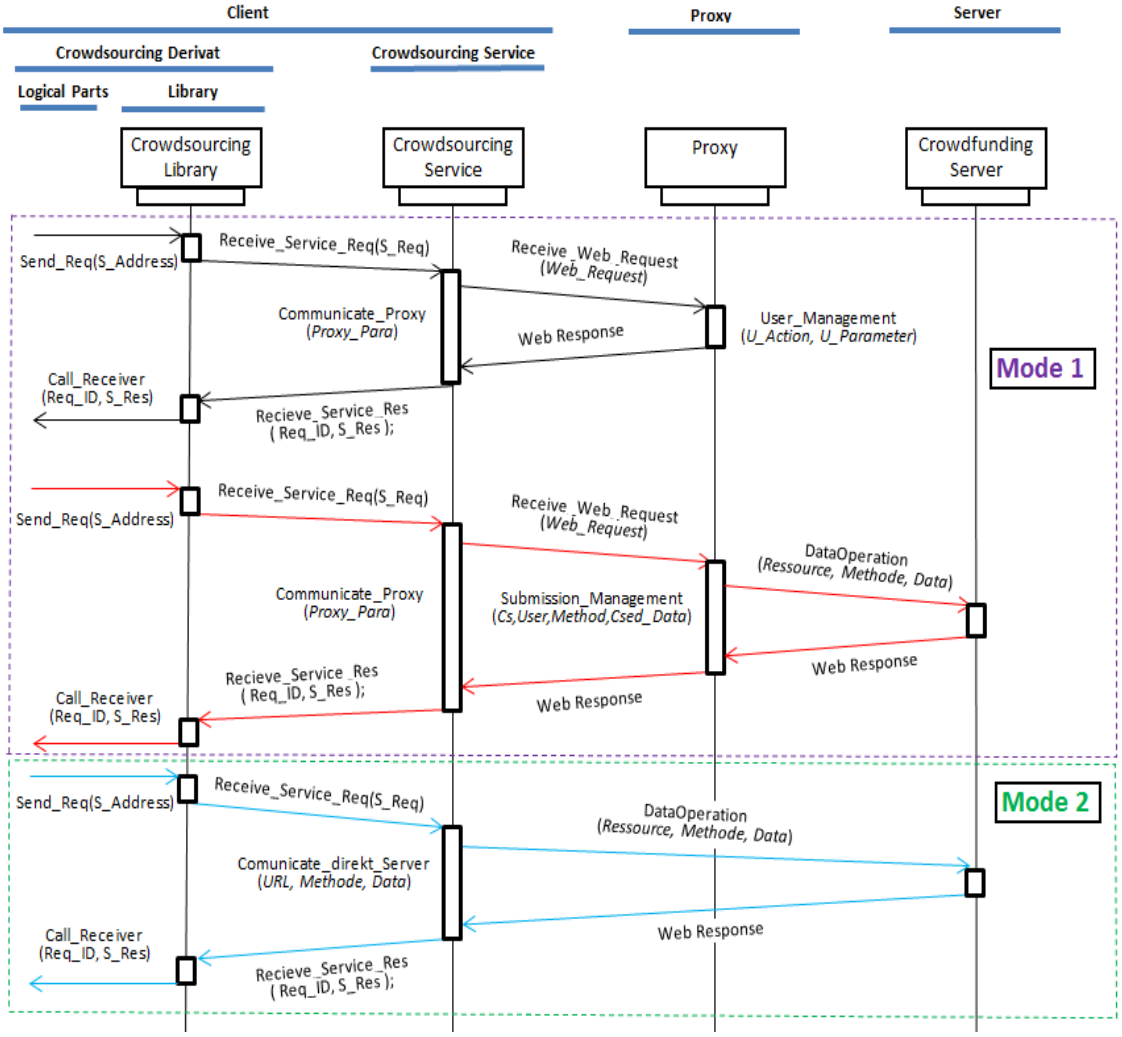


Abbildung 4-7: Kommunikationssequenz von Crowdsourcing-Infrastruktur

Diese Abbildung 4-7 beschreibt die Kommunikation zwischen Komponenten in der generellen Crowdsourcing-Infrastruktur. Die generelle Crowdsourcing-Infrastruktur unterstützt zwei Kommunikation Moden. Mode 1 ist eine Proxy Mode. Die Kommunikation mit Schwarz Farbe in Abbildung 4-7 bedeutet die Operationen von

Benutzerverwaltung. Weil diese Operationen in Proxy erledigt werden können, gibt es keine weitere Kommunikation mit Crowdfunding Server. In der Abbildung 4-7 präsentiert die Kommunikation mit Rot Farbe eine typische Submission Kommunikation zwischen Crowdsourcing-Derivat, Proxy und Server. Die Crowdsourced-Daten werden zuerst an den Proxy gesandt, dann weiter an den Server gesandt, schließlich dort gespeichert.

Mode 2 ist eine „direkt an den Server" Mode. Alle Anfragen aus *Crowdsourcing-Service* werden direkt an den Crowdfunding Server gesandt. Die Kommunikation mit Blue Farbe in Abbildung 4-7 beschreibt diese Mode. Mode 2 ist eine einfache Mode und lässt sich Komponente *Crowdsourcing-Service* nicht nur von Proxy abhängen. Diese zwei Kommunikationsmoden können von einem Crowdsourcer sowie frei ausgewählt werden.

Für Crowdfunder gibt es noch zwei Kommunikationen. Durch eine Kommunikation kann Crowdfunder die Daten verwalten, z.B. Crowdsourced Daten lesen und Crowdsourcing-Aufgabe erstellen. Durch die zweite Kommunikation kann Crowdfunder nach „*SubmissionID*" über Crowdsourcer bei Proxy nachfragen. Die beiden Kommunikationen werden in der Abbildung 4-8 beschreibt. In diesen zwei Kommunikationen fragt ein Crowdfunder direkt bei dem Server oder Proxy nach.

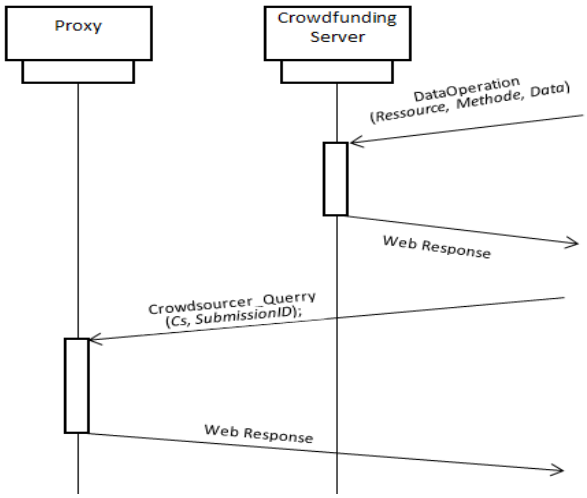


Abbildung 4-8: Kommunikation von Crowdfundern

4.4.1 Synchronisation und Nebenläufigkeit

Die Kommunikation zwischen *Crowdsourcing Server, Proxy* und *Crowdfunding Server* ist synchrone Kommunikation, d.h. eine Anfrage auf die Antwort warten muss. Aber die Kommunikation zwischen Crowdsourcing-Derivaten und Crowdsourcing-Service ist besondere. Die Kommunikation zwischen *Crowdsourcing-Library* und *–Service* ist Asynchron. Ein Derivat sendet eine *Service Request,* danach wird die Rückschnittstelle automatisch von dem *Crowdsourcing-Service* aufgerufen. Deshalb kann ein Crowdsourcing-Derivat asynchron mehrere Anfragen senden. Anfrage A und C in Abbildung 4-9 zeigt diese Situation. Natürlich kommen die nebenläufigen Anfragen bei *Crowdsourcing-Service* aus unterschiedlichen Derivaten, z.B. Anfrage C und Anfrage D in der Abbildung 4-9. *Crowdsourcing-Service* sollte auch die nebenläufigen Anfragen

verarbeiten. Weil *Crowdsourcing-Service* Komponente ein „Singelton“ ist, braucht er eine Schlange, die Anfragen temporär zu speichern.

Es gibt eine Warteschlange in *Crowdsourcing Service*, wo alle Anfragen aus Derivaten gespeichert werden. Eine Anfrage kommt beim *Crowdsourcing Service* an, falls Service schon startet, wird diese Anfrage in Warteschlange gespeichert, sonst wird Service zuerst gestartet.

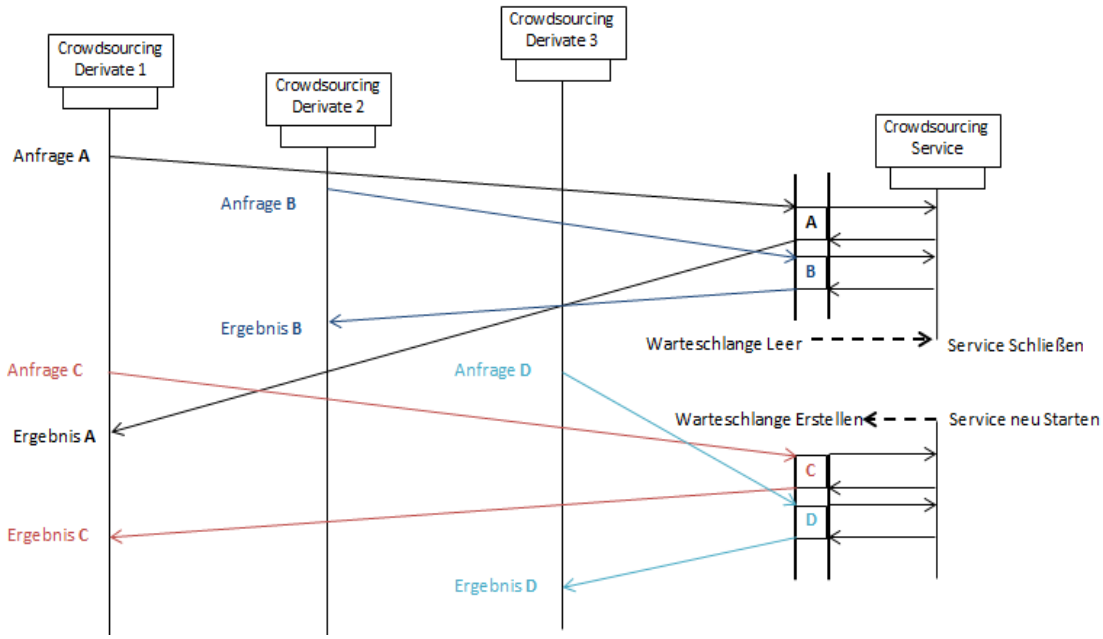


Abbildung 4-9: Situation von Multi *Crowdsourcing-Service Anfragen*

Weil die Anfragen durch das Operation-System in einem mobilen Gerät übertragen werden, könnte die gleiche Übertragung-Zeit von diesem System nicht gesichert werden, d.h. der *Crowdsourcing-Service* weiß nicht die Erstellung-Zeit-Reihenfolge von Anfragen, z.B. Crowdsourcing-Service weiß nicht, ob Anfrage C früher als Anfrage D passiert, aber weiß er, dass Anfrage C früher als Anfrage D ankommt. Diese Warteschlange nimmt FIFO(abkürzt: First In First Out) sowie FIFS(abkürzt: First In First Serve) Prinzip an. Wenn die Warteschlange leer ist, schließ der Crowdsourcing selbst ohne Schließen-Befehl aus Crowdsourcing-Derivat. Die Abbildung 4-10 zeigt die Pseudo Kode von *Crowdsourcing-Library* und *–Service*.

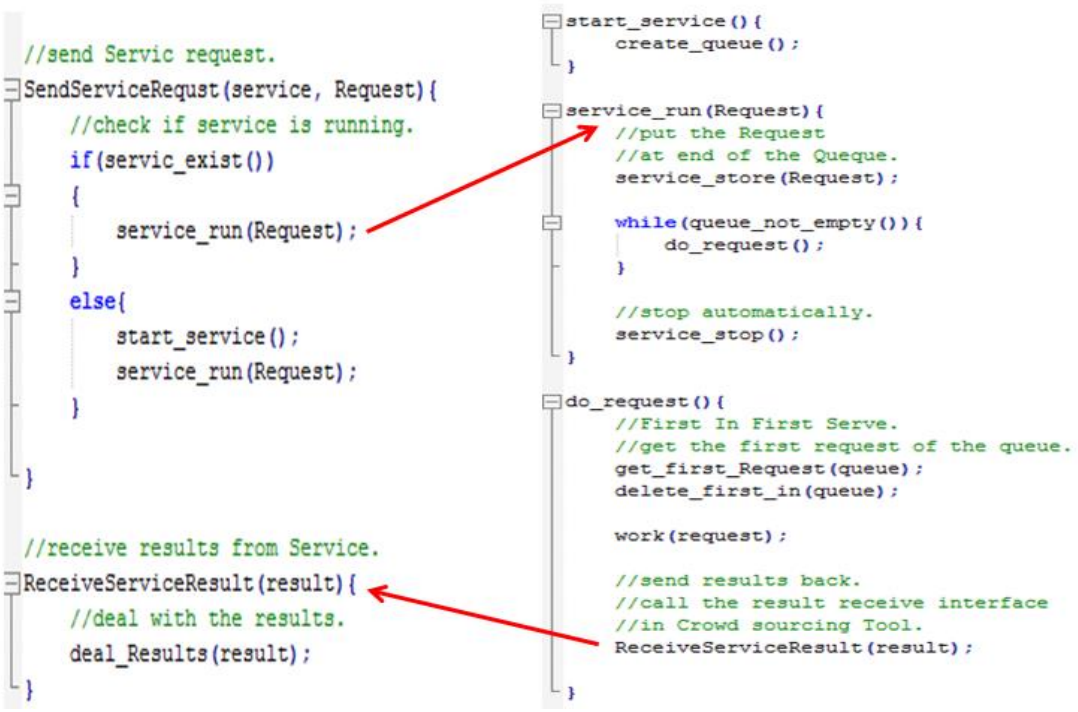


Abbildung 4-10 Pseudo Kode von Kommunikation in *Crowdsourcing-Library* (Links), in *Crowdsourcing-Service* (rechts)

4.4.2 Deadlock Risikos auf Clientseite

Die Deadlock Risikos konzentrieren sich hauptsächlich auf *Crowdsourcing-Library* und *Crowdsourcing Service*. Weil ein Derivat unterschiedlichen *Service Anfrage* senden kann, könnte ein Deadlock bei der Verarbeitung von unterschiedlichen *Crowdsourcing-Service Ergebnissen* passieren, z.B. die Verarbeitung von *Ergebnis A* wartet oder braucht die *Ergebnis C*, aber die Verarbeitung von *Ergebnis C* wartet oder braucht *Ergebnis A*, dann gibt es ein Deadlock. Deshalb müssen *Ergebnissen* aus *Crowdsourcing-Service* parallel verarbeitet werden, d.h. die *Ergebnisse* nicht voneinander abhängen müssen. Entwickler eines Derivats kann sich um das Entwerfen von parallelen *Service Anfragen* kümmern, damit kann das Risiko vermeidet werden.

Ein anderes Risiko könnte bei Aufrufen passieren, z.B. ein Derivat A ruft *Crowdsourcing-Service* auf. Gleichzeitig ruft *Crowdsourcing-Service* den Empfänger in Derivate A auf. In diesem Fall kann *Crowdsourcing-Service* den neuen Aufruf aus Derivat A nicht weiter verarbeiten, schließlich passiert ein Deadlock. Eine Grundidee über diese Crowdsourcing-Infrastruktur ist, dass *Crowdsourcing-Service* nur für Senden von *Service Ergebnissen* verantwortlich ist, d.h. *Crowdsourcing-Service* keine Antwort aus *Receiver* in einem Derivat braucht. Nach Senden von einem *Service-Response* kann die Verarbeitung einer neuen *Service Anfrage* gestartet werden. Ob das *Service Ergebnis* von einem Crowdsourcing-Derivat schon akzeptiert und verarbeitet wurde, wird nicht von *Crowdsourcing-Service* gekümmert und geprüft. In diesem Fall kann ein von einem Derivat verpasstes *Service Ergebnis* nicht wieder hergestellt werden.

4.5 Event-Flow in Crowdsourcing-Infrastruktur

Bis hier kann der Crowdsourcer-Ereignisablauf im Abbildung 4-11 zusammengefasst werden. In dieser Abbildung zeigt keinen Ereignisablauf von Crowdfunder. Aber der Ereignisablauf von Crowdfunder ist einfach. Crowdfunder erstellt Daten, z.B. Crowdsourcing-Aufgabe und –Anreiz. Diesen Daten werden in Crowdfunding Server gespeichert. Crowdfunder kann auch die Crowdsourced Daten sammeln. Um die Crowdsourcer anzureizen, kann Crowdfunder durch *SubmissionID* bei Proxy nachfragen. Wenn die Crowdsourced Daten von Crowdsourcer nicht anonym an den Server gesandt werden, dann kann Crowdfunder den Crowdsourcer finden.

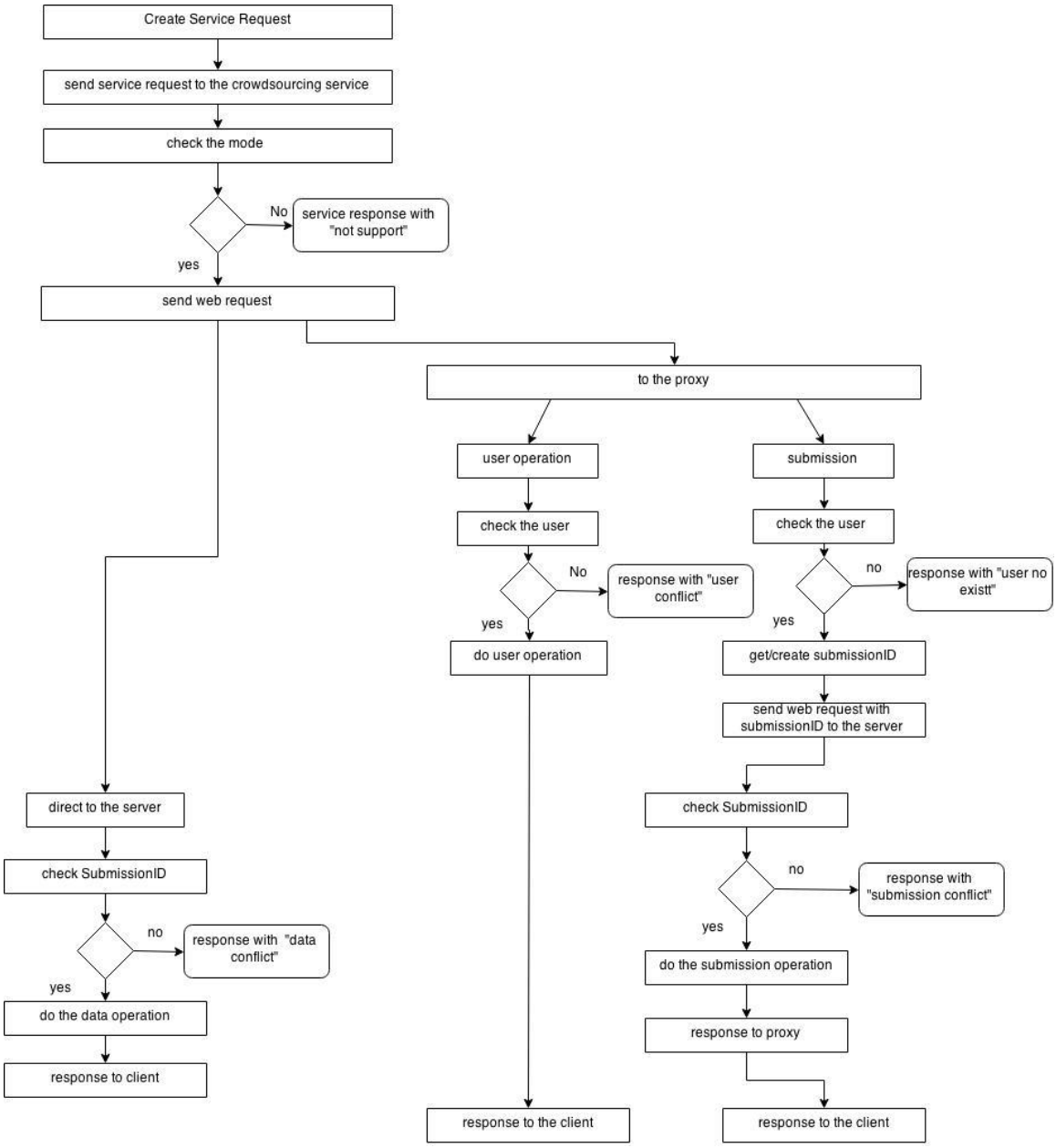


Abbildung 4-11: Ereignisablauf-Diagramm von Crowdsourcer.

4.6 Crowdsourcing-Derivate mit Crowdsourcing-Infrastruktur

Die Motivationen von Crowdsourcing-Infrastruktur sind, nicht nur die System Ressourcen in mobilen Geräten zu sparen, sondern auch die Entwicklung neues Crowdsourcing-Derivats zu vereinfachen. Um die Anforderungen einer generellen Crowdsourcing-Infrastruktur zu analysieren, wurden zwei Dummy-Crowdsourcing-Derivate sowie FAF (abkürzt für: Fragen-Antwort Forum System) und NSÜ (abkürzt für: Netzwerk-Signalstärke Überwachung System) schon in Abschnitt 3.2 auf Seite 25 entworfen. Um FAF zu realisieren, überlegen die Entwickler nur, wie die Daten auf Clientseite angezeigt werden und wie die Prämie einer Frage bezahlt werden sollten. Crowdsourcing-Infrastruktur kann die User-Verwaltung machen und die Crowdsourcer-Informationen einer bestimmten Antwort ausgeben. Natürlich werden alle Daten in Crowdsourcing-Infrastruktur gespeichert. Um NSÜ zu realisieren, überlegen die Entwickler nur, wie die Sensor-Daten in einem mobilen Gerät erfasst werden. Die Crowdsourcing-Infrastruktur kann die Daten auf Serverseite speichern. Außerdem müssen beide Crowdsourcing-Derivate eigene Datenobjekte definieren.

4.7 Zusammenfassung

In diesem Kapitel wird ein Konzept einer generellen Crowdsourcing-Infrastruktur zuerst vorgestellt. Grundsätzlich besteht sie aus 4 kleinen Komponenten. In jeder Komponente werden die Operationen diskutiert. Gleichzeitig wird der Ereignisse-Ablauf darin angezeigt.

Nach Vorstellung der Architektur wird den Kommunikation-Mechanismus in diesen Infrastruktur vorgestellt, darin werden auch die Möglichkeiten von Deadlock auch diskutiert.

Der ganze Ereignisablauf dieser Crowdsourcing-Infrastruktur wird in dem Kapitel 4.5 beschrieben.

In dem nächsten Kapitel wird eine wirkliche Crowdsourcing-Infrastruktur nach diesem Konzept realisiert.

5 Implementierung

Im letzten Kapitel wurde schon ein Konzept einer generellen Crowdsourcing-Infrastruktur generiert und vorgestellt. In diesem Kapitel wird das Konzept umgesetzt. Das Konzept hat drei Tiere wie „Client-Proxy-Server“, deshalb hat diese Umsetzung drei Teile. Die Clientseite basiert auf Android OS. In Proxy-Tier wird ein vorhandener Proxy wie SANE verwendet. Auf Serverseite wird die Infrastruktur durch das RESTful-Pattern realisiert. Schließlich kommunizieren alle Komponenten miteinander durch das HTTP-Protokoll. Die Daten zur Übertragung werden in JSON formatiert.

5.1 Clientseitige Crowdsourcing-Infrastruktur

Das Android OS wird als die Implementierung-Umgebung auf Clientseite ausgewählt, weil die meisten Smart Phone das Android OS benutzt. Weil das Android OS auf JAVA Programm-Sprache basiert, wird die clientseitige Infrastruktur durch JAVA Sprache programmiert. Eine bekannte JAVA-IDE wie Eclipse [20] wird in Implementierung verwendet.

Die Implementierung clientseitiger Crowdsourcing-Infrastruktur hat zwei Teile, wie *Crowdsourcing-Library* und *Crowdsourcing Service*. Die *Crowdsourcing-Library* ist ein JAVA-Library und wird in jedem Crowdsourcing-Derivat eingebettet und wieder verwendet. Crowdsourcing-Service ist eine Android-Applikation und bietet hauptsächlich nur einen Service an, der alle Anfragen aus unterschiedlichen Crowdsourcing-Derivaten verarbeitet. Zuerst wird die Komponente *Crowdsourcing-Library* vorgestellt.

5.1.1 Crowdsourcing-Library

Die Bibliotheken zur Implementierung von *Crowdsourcing-Library* werden in der Tabelle 5-1 aufgezählt.

Bibliotheken	Version	Beschreibung
Android SDK	4.4.2	Grundlegende Bibliothek für Entwicklung einer Android-Applikation.
Gson	2.3	Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of. [14]

Tabelle 5-1: Bibliotheken zur Implementierung von *Crowdsourcing-Library*

Die Crowdsourcing-Library ist für die Kommunikation zwischen *Crowdsourcing-Derivat* und *–Service* verantwortlich. Die Kommunikation zwischen unterschiedlichen Komponenten in Android OS wird durch Klasse „*Intent*“ realisiert, deshalb werden die *Service-Anfragen* und *-Antworten* in clientseitiger Crowdsourcing-Infrastruktur durch „*Intent*“ Objekte realisiert, d.h. alle Eingaben und Ausgaben von dieser clientseitigen Crowdsourcing-Infrastruktur in „*Intent*“ Objekten eingepackt werden. Die Einstellung von Parametern in *Crowdsourcing-Library* wird durch paarweise Methode sowie <Name, Value> realisiert, z.B. die Einstellung von *Req_ID* ist ähnlich wie `intent.putExtra(resultcode, values)`. In *Crowdsourcing-Service* wird ein Parameter-

[20] Eclipse ist ein quelloffenes Programmierwerkzeug zur Entwicklung von Software mit JAVA Programmsprache.

Wert abhängig von einem Namen gelesen, z.B. `intent.getString(resultcode)`, deshalb sind die Namen von Parametern die reservierten Wörter während der Erstellung einer Service-Anfrage.

Die Implementierung der Operation: Add_Data(*Data*) in Konzept unterstützt zwei Datentypen, sowie „*String*“ und „*LinkedHashMap*“. Datentyp „*String*“ ist eine gute Wahl für die Eingabe von einer generellen Infrastruktur, weil „*String*“ als ein Grunddatentyp von meisten Systemen oder Programm-Sprachen unterstützt wird. Außerdem könnten andere Datentypen, z.B. Video- und Bilddaten, in „*String*“ konvertiert werden.

Typ „*LinkedHashMap*“ ist für die Kommunikation Mode sowie Proxy Mode gültig. Durch „*LinkedHashMap*“ können die den Proxy sowie SANE anpassenden Web-Anfragen einfach erstellt werden. z.B. *LinkedHashMap.put(action,deleteuser)*. Weil „*LinkedHashMap*“ nicht direkt in „*Intent*“ hinzugefügt werden kann, wird eine externe Bibliothek für Konvertierung verwendet. Gson-2.3 ist eine Java-Library. Sie kann Java-Objekte in JSON-Repräsentation und umgekehrt konvertieren. Zuerst wird „*LinkedHashMap*“ durch die externe Library Gson-2.3 in JSON konvertiert. JSON ist „*String*“, deshalb kann „*LinkHashMap*“ in „*Intent*“ hinzugefügt werden, z.B. `intent.putExtra(cs_data,Crowdsourced_Daten).`

In Realisierung von Operation Send_Req(*S_Address*) wird eine von Android OS angebotene Funktion `startService(intent)` verwendet. Durch diese Funktion kann ein Crowdsourcing-Derivat den *Crowdsourcing-Service* [21] direkt aufrufen. Weil *Crowdsourcing-Service* als eine Android-Applikation installiert wird, wird die Adresse von *Crowdsourcing-Service* in einem Mobilen Gerät festgelegt, d.h. die Adresse von *Crowdsourcing-Service* ein statischer Wert in *Crowdsourcing-Library* gespeichert wird.

Die Operation *Recieve_Service_Res(Req_ID, S_Res)* in Konzept wird von *Crowdsourcing-Service* aufgerufen. In Android OS kann eine Applikation nicht einfach eine Komponente in anderer Applikation aufrufen. Aber in Android OS kann die Schnittstelle *onReceiverResult(int,Bundle)* in Klasse „*ResultReciever*“ durch „*ResultReceiver.send(int,Bundle)*“ aufgerufen werden. Deshalb verwendet *Crowdsourcing-Library* den Mechanismus „ResultReceiver“. In diesem Fall wird ein „*Receiver*“ Objekt in einem „*Intent*“ eingepackt und an *Crowdsourcing-Service* gesandt. Durch *Receiver.send(int, Bundle)* ruft *Crowdsourcing-Service* die Schnittstelle *onReceiverResult(int, Bundle)* in *Crowdsourcing-Library* auf. Im „*Receiver*“ Objekt gibt es eine Schnittstelle. Durch Einstellung dieser Schnittstelle kann *Crowdsourcing-Library* die Ergebnisse aus *Crowdsourcing-Service* weiter in eine kundenindividuelle Klasse aufrufen.

Die Klasse-Struktur in Eclipse wird in Abbildung 5-1 angezeigt. Grundsätzlich es gibt drei Klassen darin. Ihre Beschreibungen werden in der Tabelle 5-2 angezeigt. Das UML-Diagramm von Klassen in *Crowdsourcing-Library* wird in der Abbildung A-1 angezeigt.

[21] Crowdsourcing Service bedeutet ein Android Applikation. Diese Applikation ist ein Teil von Crowdsourcing-Infrastruktur.

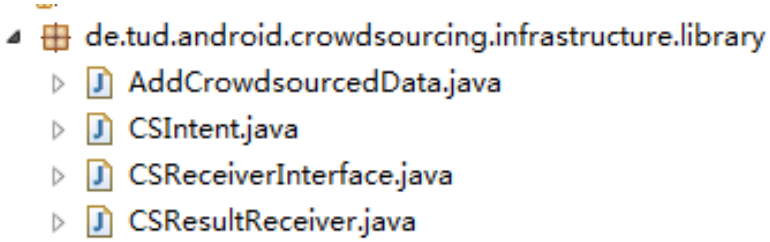


Abbildung 5-1: Screenshot von Klasse-Struktur über Crowdsourcing-Infrastruktur-Library in Eclipse

Klassen Name	Beschreibung
CSIntent	Einen spezifischen *Intent* für Crowdsourcing-Service generieren. Die Schnittstelle *AddCorwdsourcedData* implementieren. Die Operation Create_Service_Req(*R_Adress, Back_Adress, Req_ID, S_Para*) in Konzept realisieren.
CSReceiver -Interface	Durch „*Implements*“ dieser Schnittstelle eine kundenindividuelle Klasse gezeigt werden. Unterschiedlichen Exceptions aus *Crowdsourcing-Service* erfassen.
CSResult-Receiver	Eine Vererbung-Klasse von „*ResultReceiver“* in Android OS. Darin gibt es ein „*CSReceiverInterface*“ Objekt. Diesem Objekt kann eingesetzt werden. Wenn *onReceiverResult(int, Bundle)* aufgerufen wird, werden die Operationen in dem „*CSReceiverInterface*“ Objekt ausführt.
AddCorwdsourcedData	Eine Schnittstelle für Hinzufügung von *Crowdsourced Daten.*

Tabelle 5-2: Klassen in *Crowdsourcing-Library* und ihre Beschreibung

Weil diese *Crowdsourcing Library* sowie ein *JAVA-Package* keine spezifischen Komponenten in Android OS, z.B. *Activity, Service* und *Broadcast*, hat, braucht er keine Konfiguration in einem Android-Applikation Projekt, d.h. das File *AndroidManifest.xml*[22] nicht braucht, zu verändern.

5.1.2 Crowdsourcing Service

Abhängig von dem Konzept der Komponente *Crowdsourcing-Service* wird eine Android-Applikation *„CSInfrastruktur (abkürzt für: **C**rowd**s**ourcing Infrastruktur)“* durch Eclipse realisiert. Die Bibliotheken für diese Applikation sind gleich wie die Bibliotheken in der Tabelle 5-1.

Es gibt drei Unter-Komponenten in der Komponente *Crowdsourcing-Service*, deshalb besteht diese Applikation aus drei *Packages*. Die Abbildung 5-2 zeigt die Package-Struktur in Applikation *CSInfrastruktur* an.

[22] Ein notwendiges File einer Android Applikation sein. Darin gibt es die Konfiguration-Information über diese Applikation in Android OS.

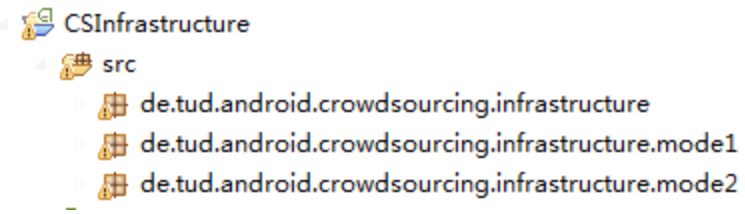


Abbildung 5-2: Screenshot in Eclipse über Klassen-Struktur von Android-Applikation *CSInfrastruktur*

Package Infrastructure realisiert die Unter-Komponente *Service-Driver* in der Abbildung 4-3 auf Seite 36. In diesem *Package* gibt es nur einen „*Intentservice*[23]“. Dieser „*Intentservice*“ akzeptiert und verantwortet die aus unterschiedlichen Crowdsourcing-Derivaten ankommenden Intent-Anfragen. Im Vergleichen zu anderen Android APIs, z.B. *AsyncTask*[24], *Thread*[25], hat *Intentservice* einige besondere Eigenschaften,

- Es gibt nur eine Instanz während Laufzeit.
- *Intentservice* hat eine private Schlange, die ankommende *Intent* speichern kann. Wenn *Intentservice* eine Aufgabe erledigt, nimmt *Intentservice* eine neue aus der Schlange.
- *Intentservice* kann selbst schließen, falls es keine weitere Arbeit in der Schlange gibt.

Ein Nachteil von *Intentservice* ist, dass er nicht parallel die ankommenden *Intents* verarbeiten kann.

Package Mode1 realisiert die Kommunikation mit dem Proxy. In der Implementierung wird SANE als Proxy benutzt. SANE braucht eine „*Signature*“ in jeder Web-Anfrage, deshalb gibt es eine Klasse für „*Signature*“ in *Package Mode1*. Die Quellcodes kommen aus Projekt MapBiquitous. Ein Beispiel für eine Web-Anfrage bei SANE ist ähnliche wie: http://SANEURL/?cs=map&user=tom&signature().

Package Mode2 realisiert die Kommunikation direkt mit dem Server. Wegen RESTful-Technologien auf Serverseite wird die Ziel-Adresse einer Web-Anfrage durch URI präsentiert. Alle Crowdsourced-Daten werden in dem Body einer HTTP-Web-Anfrage hinzugefügt.

Nach Absenden von Web-Anfragen warten Mode 1 und Mode 2 die Web-Antworten aus dem Server. Die Verarbeitung von Webantworten in Mode 1 und 2 ist gleich. Der Zustand-Kode von HTTP-Web-Antworten und Inhalt von *Respons Body* werden

[23] IntentService is a base class for Services that handle asynchronous requests (expressed as Intents) on demand. The service is started as needed, handles each Intent in turn using a worker thread, and stops itself when it runs out of work.[23]

[24] AsyncTask enables proper and easy use of the UI thread. This class allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.[24]

[25] A Tread is a concurrent unit of execution. It has its own call stack for methods being invoked, their arguments and local variables.[25]

herausgezogen. „*Intentservice*“ verpackt die beiden Informationen in einem „*Bundle*[26]“. Durch *receive.send(requestID,Bundle)* werden die Informationen zurückgesandt.

In dem Konzept von *Crowdsourcing-Service* würden drei Arten *Exceptions* ausgegeben, sowie *ModeNotSupport, NoWebConnection, ConnectionTimeExceeded*. Um diese Ausnahme zu übertragen, werden einige *RequestID-Werte* reserviert. „RequestID=1“ bedeutet *NoWebConnection*. „RequestID=2“ bedeutet *ModeNotSupport*. „RequestID=1“ bedeutet *ConnectionTimeExceeded*. In Verwendung dieser Crowdsourcing-Infrastruktur muss das verstehen.

5.2 Implementierung von dem Proxy

Proxy benutzt SANE(abkürzt für: Server Access Network Entity). SANE wurde schon von T. Hara schon entwickelt. Die

Abbildung 5-3 und Abbildung 5-4 zeigt die Architektur von SANE. Die Ziele von SANE sind, Crowdsourcing-Services anzubieten, Crowdsourcing-Client von Crowdfunding-Server abzutrennen.

SANE bietet Crowdsourcing-Funktionen sowie Crowdsourcer-Verwaltung und Crowdsourcer-Anonymous an. Bei jeder Submission wird eine Submission-ID in SANE erstellt. Submission-ID enthält keine Crowdsourcer-Information. Crowdsourcer-Informationen und Submission-ID werden zusammen in SANE gespeichert. Danach sendet SANE weiter die Crowdsourced-Daten mit der Submission-ID an den Server. Offensichtlich kann Server keine Crowdsourcer-Informationen bekommen. Deshalb weiß Crowdfunding Server nicht, wer die Crowdsourced-Daten anbietet.

Weil SANE durch *„HashTable“* miteinander angeschlossen werden, hat SANE die Eigenschaft von Skalierbarkeit. Natürlich können SANE von unterschiedlichen Crowdfunding-Servern benutzt werden, deshalb hat SANE die Eigenschaft von Wiederverwendbarkeit.

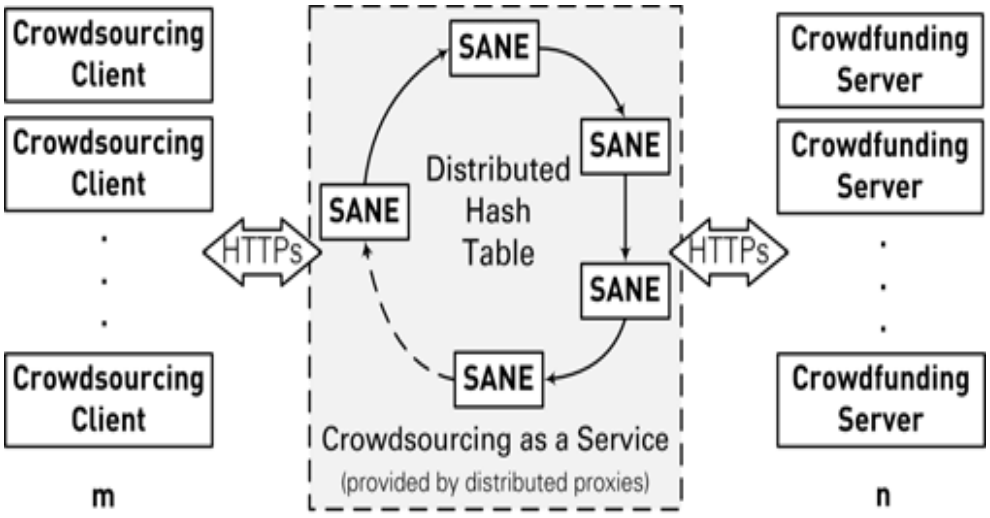


Abbildung 5-3: Architektur von einer Crowdsourcing Plattform bei SANE [6]

[26] A mapping from String values to various Parcelable types

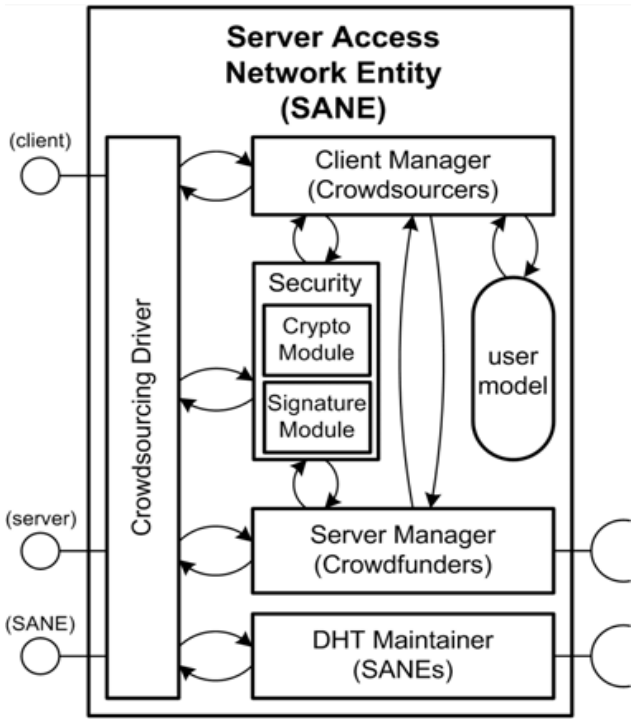


Abbildung 5-4: Architektur von einem SANE [6]

SANE bietet einige Crowdsourcing-Services an. Auf dieser Webseite[27] werden die Anwendung-Schnittstellen von SANE zusammengefasst. SANE in dieser generellen Crowdsourcing-Infrastruktur ist für die Userverwaltung und den Nutzeranonymus verantwortlich.

In der Implementierung wird Kommunikation mit SANE nicht erfolgreich erledigt. Ein Grund dafür ist, dass SANE HTTPs-Protokoll mit einem *Self-Signed* Zertifikat verwendet. Zusätzlich scheint es, dass es einige Probleme bei Web-Service-APIs von SANE gibt.

Um die SANE in dieser Crowdsourcing-Infrastruktur in der Zukunft zu integrieren, wird die nur für SANE gültigen Schnittstellen auf Serverseite implementiert, d.h. die Serverseite die Daten aus SANE akzeptieren kann.

5.3 Serverseitige Crowdsourcing-Infrastruktur

In dem Konzept von serverseitiger Crowdsourcing-Infrastruktur werden die Daten als Ressourcen betont. Alle Operationen konzentrieren sich auf die Daten-Ressourcen. Deshalb wird die Implementierung serverseitiger Crowdsourcing-Infrastruktur in zwei Teile abgetrennt. Zuerst wird die Daten-Ressourcen in der Infrastruktur vorgestellt. Danach folgt die Realisierung von serverseitigen Operationen.

[27] https://sane.hawk310.startdedicated.de/ [Zugriffszeit: 20.01.2015]

5.3.1 Daten-Ressourcen in Crowdsourcing-Infrastruktur

Um die serverseitige Crowdsourcing-Infrastruktur zu implementieren, muss die Datenstruktur festgelegt werden. In Konzept werden die *Daten-Ressourcen* schon modelliert. Es gibt drei grundlegende Ressourcen sowie *Crowdsourcing-Aufgabe*, *Crowdsourced-Daten* und *Crowdsourcing-Anreiz* auf Serverseite, aber die Beziehungen zwischen unterschiedlichen *Daten-Ressourcen* sind kompliziert, z.B. eine Aufgabe hat mehrere Anreize. Um die Implementierung zu vereinfachen, wird ein einfaches Modell angenommen. Dieses Modell wird in der Abbildung 5-5 beschriebt. In diesem Modell hat eine *Crowdsourcing Aufgabe* nur einzigen oder keinen *Crowdsourcing Anreiz*. Wenn man eine *Crowdsourced-Daten* vorlegt, kann man diesen oder keinen *Anreiz* bekommen.

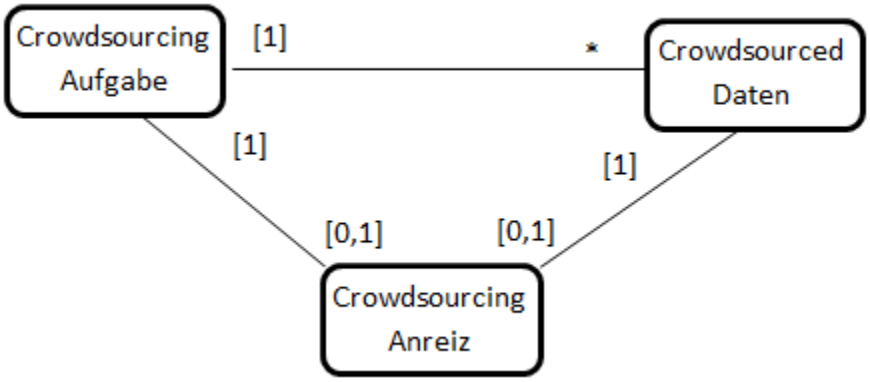


Abbildung 5-5: vereinfachte Beziehung zwischen Crowdsourcing Daten

Außer diesen drei *Daten-Ressourcen* gibt es noch eine *Daten-Ressource*, die nur von SANE verwendet wird. SANE sendet *Crowdsourced-Daten* mit einer *Submission-ID* an einen Server. *Submission-ID* muss einzigartig auf Serverseite gespeichert. Die *Daten-Ressource* für SANE kann als eine *Kind-Ressource* von *Crowdsourced-Daten* sein.

Die Abbildung 5-6 zeigt die Entity-Diagramm von serverseitigen *Daten-Ressourcen* in Crowdsourcing-Infrastruktur. Welche Daten in Crowdsourcing-Derivate verarbeitet und gesammelt werden, wird nicht von Crowdsourcing-Infrastruktur überlegt. Deshalb können die Attribute in Data-Objekten, z.B. „*Lösung_Inhalt*“, von Entwicklern selbst verändert werden. Aber das Datenformat in einem Crowdsourcing-Derivat muss die Crowdsourcing-Infrastruktur anpassen.

Es gibt keine separaten *Daten-Ressourcen* für Crowdsourcer und Crowdfunder in Crowdsourcing-Infrastruktur. Aber diese Informationen werden als Attribute in *Lösung* und *Aufgabe* enthalten. Statt *Anreiz*-Ressource in Konzept gibt es eine Datenressource *Anreiz-Rekord*, die die Anreize-Informationen statisch unter einem Nutzer-Namen speichert. Durch *Anreiz-Rekord* kann ein Anreiz-Mechanismus einfach realisiert werden. Zuerst erfahren Crowdfunder einen *Crowdsourcer-Namen* einer *Lösung*. Nach diesem Namen verändert Crowdfunder den *Anreiz-Rekord*. Crowdsourcer kann durch Lesen von einem *Anreiz-Rekord* erfahren, ob er einen „*Anreiz*“ schon bekommen hat.

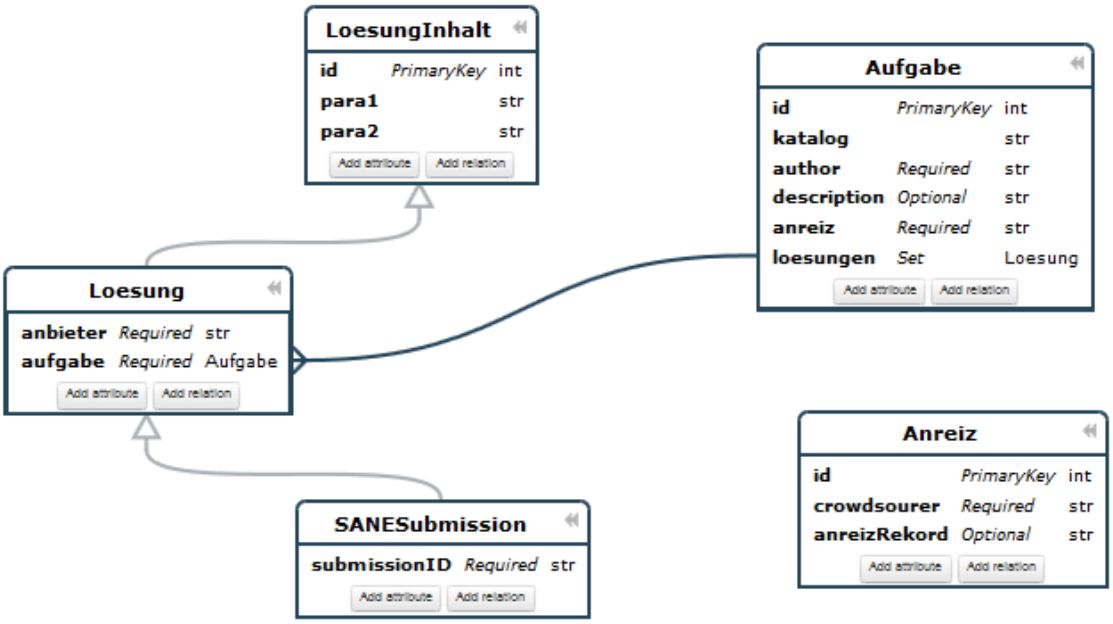


Abbildung 5-6: Entity-Diagramm von *Daten-Ressourcen* auf serverseitiger Crowdsourcing-Infrastruktur

Um diese relationale Datenbank zu realisieren, wird JDO 2.3[28] verwendet. JDO 2.3 unterstützt die Operation von „*cascading-delete*“, z.B. während Löschen eines „*SANELoesung*“ Objekt muss die „*Loesung*“ Objekt darin gleichzeitig gelöscht werden.

Um die unterschiedlichen Crowdsourced-Datentypen zu unterstützen, kann die Klasse *LösungInhalt.java* nach Anforderungen aus Crowdsourcing-Derivaten dynamisch verändert werden.

5.3.2 Serverseitige Operationen in Crowdsourcing-Infrastruktur

Im letzten Abschnitt werden die *Daten-Ressourcen* auf Serverseite schon dargestellt. Die Realisierung von serverseitiger Crowdsourcing-Infrastruktur basiert auf das RESTful-Pattern. Die Bibliotheken zur Implementierung serverseitiger Crowdsourcing-Infrastruktur werden in der Tabelle 5-3 aufgezählt.

Werkzeug	Version	Beschreibung
JAVA-JDK	1.7.0_25	Grundlegende Library für Java-Programmsprache
Google App Engine	1.9.9	Google App Engine is a Platform as a Service (PaaS) offering that lets you build and run applications on Google's infrastructure. [15] This Library contains the JDO 2.3 Library.
Jersey	1.18	Developing RESTful Web services that seamlessly support exposing your data in a variety of representation media types and abstract away the low-level details of the client-server communication is not an easy task without a good toolkit. [16]

Tabelle 5-3: Bibliotheken auf serverseitiger Crowdsourcing-Infrastruktur

[28] Java Data Objects (JDO) is a standard interface for storing objects containing data into a database. The standard defines interfaces for annotating Java objects, retrieving objects with queries, and interacting with a database using transactions. [3]

Bei RESTful gibt es Standard-Technologies, z.B. URI präsentiert eine Ressource, HTTP-Protokoll ist für Web-Kommunikation, Daten zur Übertragung werden in JSON-Datenformat formatiert. Deshalb wird HTTP-Webprotokoll für Web-Kommunikation in der generellen Crowdsourcing-Infrastruktur. Die Daten zur Übertragen zwischen Client und Server in der generellen Crowdsourcing-Infrastruktur nehmen JSON-Datenformat an.

Die Operationen auf Serverseite in Crowdsourcing Infrastruktur werden in Tabelle 5-4, Tabelle 5-5 und Tabelle 5-6 auf Seite 66 beschrieben. Die erfolgreichen Ausgaben von jeder Operation werden durch JSON-Datenformat formatiert und in HTTP-Antworten mit dem Zustand-Kode „200“ zusammen eingepackt. Alle Ausnahmen von diesen Operationen werden mit Zustand-Kode „4XX“ in HTTP-Webantworten eingepackt. Außer den Ausnahmen, die in diesen Tabellen schon aufgezählt werden, gibt es noch allgemeine Ausnahmen, z.B. interne Fehlers von Servern und Datenbanken. Diese allgemeinen Ausnahmen werden mit Zustand-Kode „5XX“ in HTTP-Webantworten eingepackt.

In der Abbildung 5-7 wird die Package-Struktur in Eclipse angezeigt. Jedes Package ist für eine Datenressource verantwortlich.

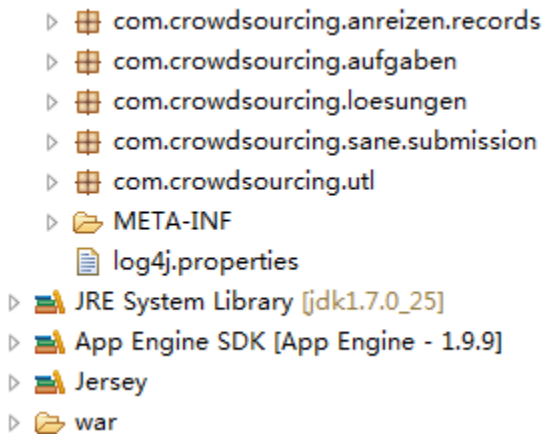


Abbildung 5-7: Screenshot der Package-Struktur auf serverseitiger Crowdsourcing-Infrastruktur in Eclipse

Operationen	URI: http://Serverurl/ Crowdsourcing/	HTTP-Methode	Ausgaben	Exceptions
Get_Aufgabe_List()	Aufgabelist/{cs_Derivat}	Get	Aufgabenlist	
Get_Aufgaben_By_ID()	Aufgabe/{aufgabeID}	Get	Aufgabe	AufgabeID nicht existiert.
Get_Anreiz_By_User()	Anreiz/{user}	Get	Anreiz	User nicht existiert.
Create_Lösung_By_AufgabeID()	Loesung/	Post	DatenID	AufgabeID nicht existiert.
Delete_Lösung_By_ID()	Loesung/{loesungID}	Delete	„Success“	DataID nicht existiert.
Get_ Lösung _By_AufgabeID()	Loesung/list/{aufgabeID}	Get	Datenlist	AufgabeID nicht existiert.
Update_data_By_ID()	Loesung/{loesungID}/{para}	Post	„Success“	DataID nicht existiert.

Tabelle 5-4: Operationen für Crowdsourcer in Mode 2: direkt an den Server Mode

Operationen	URI: http://Serverurl/ Crowdsourcing/	HTTP-Methode	Ausgaben	Exceptions
Delete_Aufgaben_By_ID()	Aufgabe/{aufgabeID}	Delete	„Success“	AufgabeID nicht existiert
Create_Aufgabe()	Aufgabe/	Post	AufgabeID	
Create_Anreiz()	Anreiz/	Post	AnreizID	
Delete_Anreiz()	Anreiz/	Delete	„Success“	AnreizID nicht existiert.

Tabelle 5-5: Operationen für Crowdfunder

Operationen	URI: http://Serverurl/ Crowdsourcing/	HTTP-Methode	Ausgaben	Exceptions
Get_Submission_By_SubmissionID()	SANE/loesung/{submissionID}	Get	Lösesung	SubmissionID nicht existiert.
Delete_Submission_By_SubmissionID()	SANE/loesung/{ submissionID }	Delete	„Success“	SubmissionID nicht existiert. LösungID nicht existiert.
Create_Submission()	SANE/loesung/	Post	„Success“	SubmissionID conflict. AufgabeID nicht existiert.
Update_Submission_By_SubmissionID()	SANE/losung/{ submissionID }	Put	„Success“	SubmissionID nicht existiert. LösungID nicht existiert. AufgabeID nicht existiert.

Tabelle 5-6: Operationen für SANE in Mode 1: Proxy Mode

5.4 Dummy-Crowdsourcing-Derivat mit Benutzerschnittstelle

In dieser Crowdsourcing-Infrastruktur gibt es fast keine grafische Benutzeroberfläche. Um die Verwendung dieser Crowdsourcing-Infrastruktur zu zeigen, wird ein Dummy-Crowdsourcing-Derivat mit einfacher Benutzeroberfläche entwickelt. Dieses Dummy-Crowdsourcing-Derivat wird auch zur Evaluation verwendet.

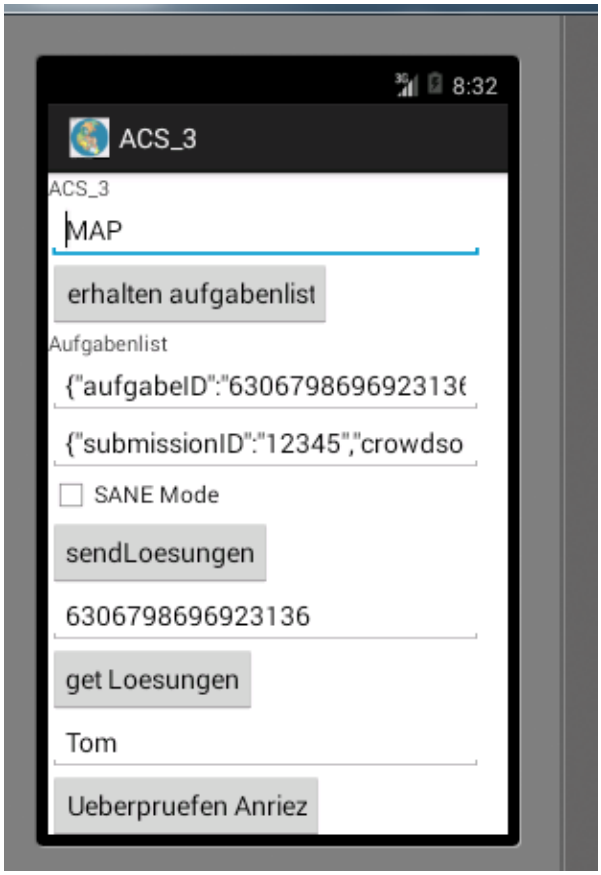


Abbildung 5-8: Dummy-Benutzeroberfläche der Crowdsourcing-Infrastruktur in einem Crowdsourcing-Derivat

Die Benutzeroberfläche von diesem Dummy-Crowdsourcing-Derivat wird in der Abbildung 5-8 angezeigt. Zuerst kann Nutzer durch *Key-Word* die Aufgabenliste lesen. In jedem „*Aufgabe-Objekt*“ wird die Anforderung einer Lösung dargestellt. Nutzer wählt eine Aufgabe aus und erstellt eine Lösung dafür. Diese Prozesse gehören zu den logischen Komponenten von einem wirklichen Crowdsourcing-Derivat. Natürlich werden die Lösungen in Crowdsourcing-Derivat schon in JSON formatiert. Die Lösungen werden durch SANE oder direkt an den Server gesandt. Schließlich wird die Antwort aus dem Server angezeigt. Abhängig von Attribute „*AufgabeID*“ einer Aufgabe kann Nutzer auch die Lösungen unter dieser Aufgabe lesen. Um den Anreiz-Mechanismus zu unterstützen, kann Nutzer seinen Anreiz-Rekord eines Nutzers schauen.

Tatsächlich sollten die Eingaben bei Mode 1 keine *SubmissionID* enthalten. Die *SubmissionID* soll von dem SANE erstellt werden. In dieser Benutzeroberfläche wird es nur gezeigt, dass die Daten mit *SumissionID* auf Serverseite akzeptiert werden können.

Diese Benutzeroberfläche zeigt nicht alle von Crowdsourcing-Infrastruktur angebotene Operationen, z.B. eine Lösung löschen oder einen Nutzer bei SANE anmelden. Nach ausführlichen Anforderungen eines Crowdsourcing-Derivats können die Funktionen der Crowdsourcing-Infrastruktur flexible kombiniert und verwendet werden.

Auf Serverseite kann Crowdfunder während Erstellung von Crowdfunding-Server die Daten-Struktur definieren. Danach kann Crowdfunder durch Web-Anfrage die Daten in dem Server verwalten, z.B. Lösung lesen und Anreiz-Rekord eines Nutzers verändern. Durch Werkzeug von Google App Engine werden die Daten in Server direkt gezeigt. Die Benutzeroberfläche von Daten wird in der Abbildung 5-9 angezeigt.

Bei Verwendung dieser Crowdsourcing-Infrastruktur müssen Entwickler die Abstraktionen von *Crowdsourcer, Crowdfunder, Aufgaben, Lösungen und Anreiz* in einem Crowdsourcing verstehen. Diese grundlegenden Begriffe für Crowdsourcing wurden schon in dem Kapitel 2.1.4 vorgestellt.

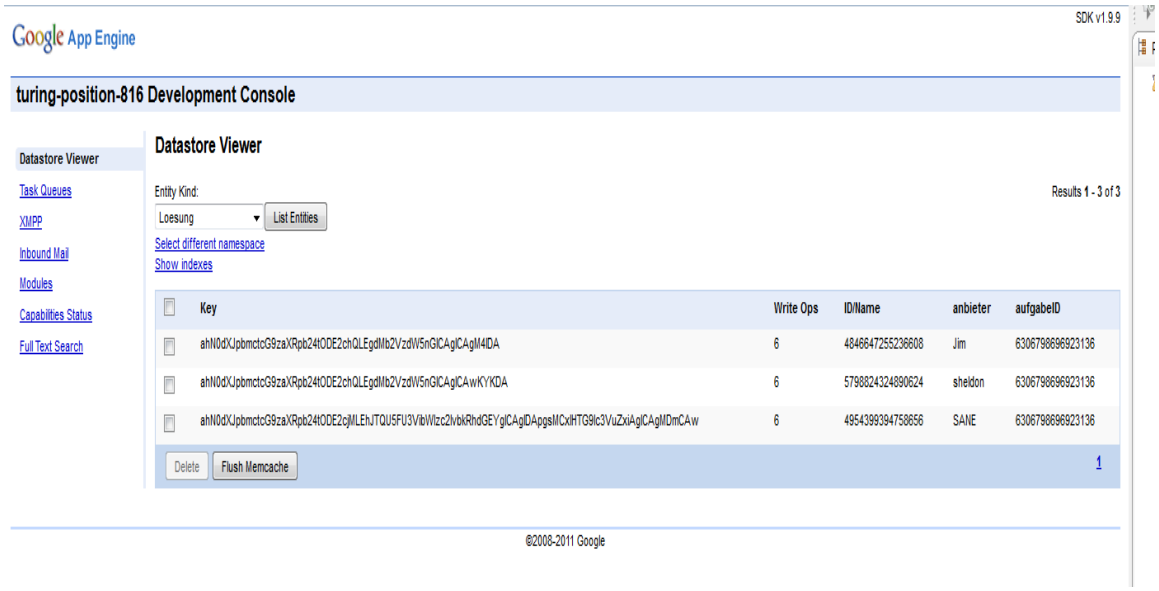


Abbildung 5-9: Ausschnitt von *Datastrore* in Google App Engine

5.5 Zusammenfassung

Zuerst wurde es in diesem Kapitel nachgewiesen, dass das Konzept dieser Doplomarbeit umgesetzt werden kann. Dabei wurden technische Besonderheiten in der Implementierung vorgestellt.

Um einfach die Crowdsourcing-Infrastruktur zu verwenden, wurde eine *Java-Packge* entwickelt. Sie wird in jedem Crowdsourcing-Derivat eingebettet und wiederverwendet. Durch diese *Java-Packge* kann die Kommunikation mit Crowdsoucing-Infrastruktur in Android System realiseirt werden. Die Crowdsourcing-Infrastruktur auf Cleintseite ist eine Android-Applikation. Diese Applikation enthält hauptsächlich eine *Intentservice*. Er kann als ein einzigartiger Service die Aufrufe aus Crowdsourcing-Derivaten verarbeiten. Danach werden Web-Anworten aus Server zurückgesandt. Dieser Service wird von Android OS angeboten, deshalb ist diese Crowdsourcing-Infrastruktur spezifisch von

Android OS abhängig. Wegen die Eigenschaften von Android OS ist die Realisierung clientseitiger Crowdsourcing-Infrastruktur einfach, z.B. die Vorteile von *Resultreseiver* und *Intentservice*. Das bedeutet auch, dass diese Infrastruktur stark von dem Android OS abhängig ist.

Die serverseitige Crowdsouring-Infrastruktur besteht aus zwei größen Teilen, sowie Proxy und Server. Der Proxy wurd durch ein vorhandene Standard-Crowdsourcing-Service sowie SANE realisiert. Mit SANE können die User-Informationen verwaltet werden. Außerdem kann Crowdsourcer auch anonym eigene Crowdsourced-Daten vorlegen. Natürlich hat SANE eigene besondere Schnittstellen, die die Web-Anfragen aus Crowdsourcern verarbeiten können. Um die Crowdsourcing-Infrastruktur zu benutzen, muss Entwickler die Schnittstelle von SANE verstehen. Neben SANE können Crowdsourer direkt eigene Crowdsourced-Daten an den Server schicken.

Der Server in Crowdsourcing-Infrastruktur basiert auf RESTful. Jede Daten-Ressource wird durch URI präsentiert. Die Daten in Server werden durch JDO gespeichert und verwaltet. Bei Entwicklung eines neuen Crowdsourcing-Derivats brauchen die Entwickler nur Attribute eines Objekts zu definieren. Bei jeder Operation kann die Daten-Konsistenz garantiert werden. Das vereinfacht die Entwicklung eines Crowdsourcing-Derivats. Auf Serverseite wird jede Web-Anfrage durch Jersey-Biliothek verarbeitet. Die Server-Ausgaben sowie Daten-Objekte werden in JSON-Format in Web-Antworten eingepackt. Die Kommunikation zwischen Client und Server benutzt HTTP-Protokoll.

Die Kommunikation zwischen Crowdsourcing-Derivaten und –Infrastruktur in Android Geräten ist asynchron, weil das Absenden einer Service-Anfrage und das Empfängen einer Service-Antwort in voneinander getrennten Prozesse zugeordnet werden. Die Kommunikation zwischen Client und Server ist syschorn, weil *Crowdsourcing-Sevice* nach Absenden einer Web-Anfrage eine dazu gehöhrte Web-Antwort wartet.

Eine Evaluierung, inwieweit die Anforderungen dieser generellen Crowdsourcing-Infrastruktur in der Implementierung erfüllt werden kann, ist im nächsten Kapitel zu entnehmen. Dabei wird diese Crowdsourcing-Infrastruktur auch auf Leistungstauglichkeit getestet.

6 Evaluation

In diesem Kapitel werden die Anforderungen für die Crowdsourcing-Infrastruktur zuerst überprüft. Danach wird die Leistung dieser Crowdsourcing-Infrastruktur analysiert, z.B. ob mit dieser Infrastruktur die RAM-Ressourcen oder Daten zur Übertragung in mobilen Geräten gespart werden kann. Schließlich wird die Modularisierung dieser Crowdsourcing-Infrastruktur untersucht.

6.1 Überprüfung von den Anforderungen für Crowdsourcing-Infrastruktur

In der Kapitel 3 werden die Anforderungen für eine generelle Crowdsourcing-Infrastruktur aufgezahlt und analysiert. In diesem Abschnitt werden diese Anforderungen überprüft.

Anforderung 1.a: Crowdsourcing-Tools kann als eine Bibliothek in jedem Crowdsourcing-Derivat wieder verwendet werden.

Ergebnisse: Erledigt.

Kommentar: Die Crowdsourcing-Library wurde in JAVA programmiert. Sie kann einfach in jeder Android-Applikation kopiert werden. Zusätzlich gibt es keine Android-Komponenten, wie *Service* oder *Activity*, deshalb braucht sie keine Informationen, in Android-Applikation zu Konfiguration. In dieser Library wird eine externe JAVA-Library GSON verwendet. Deshalb muss diese Library zusätzlich in eine Android-Applikation hinzugefügt.

Anforderung 1.b: Durch Crowdsourcing-Library werden unterschiedlichen Anfragen für Crowdsourcing-Infrastruktur-Service generiert und gesandt werden

Ergebnisse: Erledigt.

Kommentar: In *Crowdsourcing-Library* gibt es eine *CSIntent.java* Klasse. Dadurch können unterschiedlichen Service-Anfragen erstellt werden. Alle Parameter für Crowdsourcing-Infrastruktur werden darin eingepackt. Grundsätzlich wird eine Service-Anfrage durch nur einzige *Request_ID* identifiziert. Bei dem Absenden einer Anfrage wird die Android Schnittstelle sowie *StartService()* benutzt. Diese Schnittstelle ist eine *Abstract-Methode* in *Activity-Klasse* in Android OS, d.h. nur in einer *Activity-Klasse* die Service-Anfragen gesandt werden können, z.B. *StartService(csIntetent.getIntent())*.In diesem Fall werden die Inhalte im *Intent-Objekt* durch *csIntent.getIntent.getExtras()* gelesen.

Anforderung 1.c: Unterschiedlichen Typen und Formate von Crowdsourced-Daten unterstützen

Ergebnisse: Teilweise.

Kommentar: Technisch ist es schwer, alle unterschiedliche Daten-Typen und Formate zu unterstützen. In *Crowdsourcing-Library* wird nur Typ *String* für Crowdsourced-Daten unterstützt. Trotzdem ist *String* ein allgemeiner Datentyp. Andere Datentypen, z.B. Audio- oder Video-Daten, können in *String* konvertiert werden. Deshalb kann diese Crowdsourcing-Infrastruktur in unterschiedlichen Crowdsourcing-Derivaten verwendet werden. In *Crowdsourcing-Library* werden nur Formate sowie *LinkedhashMap* und

JSON von Crowdsourced-Daten unterstützt. Format *LinkhashMap* passt die Schnittstellen bei SANE an, weil alle Parameter dafür paarweise in eine HTTP-Web-Anfrage formatiert werden. Format *JSON* passt direkt den Server an, weil Server auf RESTful basiert. Durch Jesey-Bibliothek können JSON-Objekte schnell in Java-Objekt konvertiert werden.

Anforderung 1.d: Durch Crowdsourcing-Library werden die Ergebnisse aus Crowdsourcing-Infrastruktur-Service erfasst und erkennt.

Ergebnisse: Erledigt.

Kommentar: Durch Schnittstelle *onReceiveResult()* in *ResultReceiver* Klasse in Android kann die Ergebnisse aus Crowdsourcing-Service automatisch erfasst werden. Um unterschiedliche Service-Antworten zu erkennen, kann eine Nummer in einer Service-Anfrage vordefiniert werden. Diese Nummer wird in einer Service-Antwort eingepackt und zurückgesandt. Durch diese Nummer können Service-Antworten unterschieden werden.

Anforderung 1.e: Durch Crowdsourcing-Library werden individuelle Schnittstellen für Verarbeitung der Service-Ergebnisse angeboten.

Ergebnisse: Erledigt.

Kommentar: Es gibt eine Schnittstelle Klasse *CSReceiverInterface.java* in der *Crowdsourcing-Library*. Durch „*implements*“ dieser Schnittstelle können mehrere Klassen definiert werden, wo die Ergebnisse aus Crowdsourcing-Service verarbeitet werden sollten. Durch *ResultReceiver.set(CSReceiverInterface)* kann eine individuelle Klasse setzen. Natürlich kann die individuelle Klasse durch *ResultReceiver.set(CSReceiverInterface)* dynamisch verändert werden.

Anforderung 2.a: Crowdsourcing-Infrastruktur sollte von Crowdsourcing-Derivaten in einem mobilen Gerät getrennt werden.

Ergebnisse: Erledigt.

Kommentar: Die *„CSInfrastruktur (abkürzt für: **C**rowd**s**ourcing **Infrastruktur**)“* ist alleine Android-Applikation. Es kann in Android-Mobile-Geräten installiert werden. Nach einmaliger Installierung kann er alle Crowdsourcing-Derivaten bedienen.

Anforderung 2.b: Crowdsourcing-Infrastruktur sollte als einen Service spielen.

Ergebnisse: Erledigt.

Kommentar: In dieser Android-Applikation gibt es hauptsächlich eine *CSIntentService.java* Klasse. Diese Klasse erbt die Klasse *IntentService* von Android OS. Klasse *CSIntentService.java* kann die Anfragen aus unterschiedlichen Crowdsourcing-Derivaten bedienen.

Anforderung 2.c: Einzige Instanz von Crowdsourcing-Infrastruktur auf Clientseite in Laufzeit generieren.

Ergebnisse: Erledigt.

Kommentar: Das wird von dem Android OS garantiert. Es gibt nur eine Instanz eines *IntentService* Objekts während mehreren Aufrufen.

Anforderung 2.d: Mehrere Aufrufe aus unterschiedlichen Crowdsourcing-Derivaten verwalten.

Ergebnisse: Erledigt.

Kommentar: Während *IntentService*-Aufrufen wird ein *Default-Worker-Thread* erstellt. Dieser Thread ist allein und wird von dem „Haupt-Activity" einer Android-Applikation abgetrennt. Dieser Thread hat eine private *Work-Queque*. Diese Schlange kann all ankommende Anfragen speichern und nur eine Anfrage an den *Intentservice* bei einer Verarbeitung übertragen.

Anforderung 2.e: Automatisch kann Crowdsourcing-Infrastruktur selbst schließen.

Ergebnisse: Erledigt.

Kommentar: *Intentservice* in Android OS kann selbst ausschalten, wenn es keine Aufgaben in der privaten *Work-Queque* mehr gibt.

Anforderung 2.f: Ein Crowdsourcing-Derivat muss direkt den Crowdsourcing-Service aufrufen. Der Service muss die Ergebnisse direkt an dieses Derivat zurücksenden.

Ergebnisse: Teilweise.

Kommentar: Durch Crowdsourcing-Library wird eine Service-Anfrage direkt an den Crowdsourcing-Service gesandt. Aber der Crowdsourcing-Service sendet eine Service-Antwort nicht direkt sondern mit *Resultreceiver.send()* an dieses Crowdsourcing-Derivat zurück. Durch diese Funktion *Resultreceiver.send()* kann Android OS automatisch die Schnittstelle *onReceiveResult()* in Crowdsourcing-Library aufrufen.

Anforderung 3.a: Die serverseitige Crowdsourcing-Infrastruktur besteht aus losen Komponenten.

Ergebnisse: Erledigt.

Kommentar: Die Serverseitige Crowdsourcing-Infrastruktur besteht aus zwei großen Teilen sowie Server und Proxy. Crowdsourcer kann direkt oder durch Proxy SANE mit einem Server kommunizieren. Aber SANE muss mit einem Server zusammenarbeiten, weil die Crowdsourced-Daten in Server gespeichert werden müssen.

Der Crowdsourcing-Server besteht auch aus losen Komponenten, sowie einigen *Packages*. Jedes *Package* enthält einzige Daten-Ressource und ihre Operation. Zusätzlich gibt es auch die Schnittstellen für Web-Anfragen. Abhängig von Anforderungen können unterschiedlichen *Packages* mit einander flexible kombinieren und zusammenarbeiten.

Anforderung 3.b: Unterschiedlichen Daten-Arten unterstützen.

Ergebnisse: Teilweise.

Kommentar: Durch Modellierung einer Crowdsourcing-Definition gibt es nur drei grundlegende Daten-Arten sowie *Aufgabe, Lösung, AnreizRekord* in vorhandener Crowdsourcing Infrastruktur. Typischerweise gibt es keine Daten-Objekte von Crowdsourcer und Crowdfunder.

Anforderung 3.c: Grundlegende Operationen von Daten-Objekten realisieren.

Ergebnisse: Erledigt.

Kommentar: Die grundlegende Operationen sowie CRUD von Daten-Objekten sowie *Aufgabe, Lösung, AnreizRekord* werden erfolgreich realisiert. Durch JDO wird die Daten-Konsistenz bei Operationen garantiert.

Anforderung 3.d: Unterschiedliche Daten-Objekte können in unterschiedlichen Derivate dynamisch definiert werden.

Ergebnisse: Erledigt.

Kommentar: Die Daten-Objekte auf Serverseite werden durch JDO präsentiert und gespeichert. Entwickler eines neuen Crowdsourcing-Derivats können die Attribute in diesen drei grundlegenden Daten-Arten verändern und *cascade* definieren, z.B. das Attribut sowie *author* in *Aufgabe* Klasse kann auch eine Klasse sein.

Anforderung 3.e: Crowdsourcer verwalten.

Ergebnisse: Nein.

Kommentar: Die Crowdsourcing-Infrastruktur benutzt einen Standard-Crowdsourcing-Service sowie SANE. SANE kann die Crowdsourcer verwalten, z.B. User registrieren. Aber die Kommunikation mit der SANE wird noch nicht erfolgreich erledigt.

Die Nutzerverwaltung sollte von dem SANE geschafft werden. Das bedeutet, dass die Nutzerverwaltung nur in der Kommunikation-Mode 1 existiert.

Anforderung 3.f: Crowdsourcer kann anonym Crowdsourced-Daten vorlegen.

Ergebnisse: Nein.

Kommentar: SANE sendet nur Crowdsourced-Daten mit einer *SubmissionID* an den Server. Darin gibt es keine Informationen von Crowdsourcern. Aber die Kommunikation mit der SANE wird noch nicht erfolgreich erledigt. Deshalb kann die Crowdsourcing-Infrastruktur diese Funktion nicht realisieren.

Der Nutzeranonymus sollte von dem SANE geschafft werden. Das bedeutet, dass die Nutzeranonymus nur in der Kommunikation-Mode 1 existiert.

Anforderung 4.a: Kommunikation zwischen Crowdsourcing-Derivat und –Service ist asynchron.

Ergebnisse: Erledigt.

Kommentar: Die Kommunikation zwischen Crowdsourcing-Derivat und –Service ist asynchron. Durch *StartService()* ruft Crowdsourcing-Derivat den Service auf. Durch *onReceiveResult()* wird Crowdsourcing-Derivat aufgerufen.

Anforderung 4.b: Kommunikation zwischen Client und Server ist synchron.

Ergebnisse: Erledigt.

Kommentar: Die Kommunikation zwischen Client und SANE und die Kommunikation zwischen SANE und Server sind ursprünglich synchron. Die Kommunikation zwischen Client und Server ist auch synchron. Nur nach Bekommen einer Web-Antwort kann *Crowdsourcing-Service* eine neue Web-Anfragen senden.

Anforderung 4.c: Allgemeines Web-Protokoll zwischen Client und Server verwenden.

Ergebnisse: Erledigt.

Kommentar: Das Web-Protokoll für Kommunikation zwischen Client und Server benutzt HTTP-Protokoll. HTTP-Protokoll ist ein bekanntes und allgemeines Web-Protokoll. Das Android OS unterstützt das HTTP-Protokoll default, d.h. es keine zusätzlichen Bibliotheken eines Web-Protokolls während Entwicklung von Android Anwendungen braucht.

6.2 Leistung der Crowdsourcing-Infrastruktur überprüfen

Im letzten Abschnitt wurden die Anforderungen für die Crowdsourcing-Infrastruktur überprüft. Bis jetzt gibt es eine generelle Crowdsourcing-Infrastruktur mit einigen besonderen Eigenschaften, z.B. einzigartiger Service in Android OS, auf RESTful-Web-Service basierter Server, mit JDO aufgebaute Daten-Struktur. Diese generelle Crowdsourcing-Infrastruktur kann in einem neuen Crowdsourcing-Derivat wieder verwendet, deshalb kann sie die Entwicklung eines neuen Crowdsourcing-Derivats vereinfachen.

Eine weitere Motivation für diese Crowdsourcing-Infrastruktur in mobilen Geräten ist, deren System-Ressourcen zu sparen. Es gibt zwei Eigenschaften in dieser Motivation. Erste, diese Motivation konzentriert nur auf der Clientseite, sowie mobile Geräte. Zweite, ein Untersuchen-Modell für Evaluierung von System-Ressourcen ist notwendig, z.B. was in mobilen Geräten und wie evaluiert werden sollte.

6.2.1 Leistung-Evaluierung-Modell

Erste Frage, welche Systemressourcen in mobilen Geräten sollte evaluiert werden? Die Herausforderungen in Mobile Crowdsourcing sowie begrenzte System-Ressourcen in mobilen Geräten können am Nächsten zusammengefasst werden.

- Batterie Ressourcen
- Speicher z.B. RAM

In dieser Diplomarbeit wird nur die RAM-Ressource evaluiert.

Zur Beurteilung über RAM-Ressource-Sparen ist eine Kontrast-Crowdsourcing-Derivat ohne Crowdsourcing-Infrastruktur notwendig. Das Kontrast-Crowdsourcing-Derivat enthält *Crowdsourcing-Library* und *Crowdsourcing-Service*, d.h. der ganze *Crowdsourcing-Library* und *Crowdsourcing-Service* der Crowdsourcing-Infrastruktur in Kontrast-Crowdsourcing-Derivat transplantiert werden. Die Verwenden-Situation eines Crowdsourcing-Derivats ist, die serverseitigen Crowdsourcing-Aufgaben nachzufragen. Die Anzahl von verwendeten RAM-Ressourcen einer Applikation in dem Emulator wird während Laufzeit aufgezeichnet. Um die unterschiedlichen Verwenden-Situationen zu simulieren, werden RAM-Ressourcen bei unterschiedlichen Malen von Nachfragen gemessen.

6.2.2 Umgebung für Systemressourcen-Evaluierung

Die Crowdsourcing-Infrastruktur basiert auf Android OS. Die Evaluierung wird in einem AVD (abkürzt für: **A**ndroid **V**irtual **D**evice) gemacht. In der Tabelle 6-1 werden die Parameter von AVD aufgezählt. Die GUI von Kontrast-Crowdsourcing-Derivat ist gleich wie in der Abbildung 5-8.

Parameter-Name	Wert
Device	Nexus S(4.0 ´´, 480 * 800: hdpi)
Target	Google APIs(Google Inc.) – API Level 19
CPU/ABI	ARM(armeabi-v7a)
Memory Options	RAM:343MB VM Heap: 32MB
Internal Storage	200MB

Tabelle 6-1: Parameter von AVD in Evaluierung

6.2.3 Test-Vorgang

Der Test-Vorgang ist, dass nach Druck des Button „*Get Aufgaben*“ Crowdsourcing-Anfragen mit einem bestimmten Anzahl kontinuierlich an den Server gesandt werden. Während Laufzeit wird die RAM-Verwendung-Situation in Logcat[29] angezeigt. Ein Rekord von Logcat ist ähnlich wie „*GC_FOR_ALLOC freed 133K, 7% free 3055K/3260K, paused 46ms, total 54ms*„. Die Zahl 3055 ist die Anzahl vom zugeordneten RAM einer Android-Applikation. Die 3260 präsentiert die „Heap Size“ von RAM einer Android-Applikation. Die zugeordnete RAM-Ressource verändert sich häufig, weil die Android OS kontinuierlich die RAM-Ressource einer Applikation freisetzt und recycelt. Deswegen wird der Durchschnittswert von allen über RAM-Situation in Logcat gezeigten Rekorden gerechnet und angenommen. Die Probenahme sind 50, 100,150 und 200 male Nachfragen von Crowdsourcing-Aufgaben.

[29] The Android logging system provides a mechanism for collecting and viewing system debug output. Logs from various applications and portions of the system are collected in a series of circular buffers, which then can be viewed and filtered by the logcat command. Address: http://developer.android.com/tools/help/logcat.html [Zugriff am 01.02.2015]

6.2.4 Ergebnisse Analysieren

Der Test-Vorgang wird in drei Android-Anwendungen ausführt, sowie ein Crowdsourcing-Derivat mit *Crowdsourcing-Service*, ein Kontrast-Crowdsourcing-Derivat und *Crowdsourcing-Service*. Die Messergebnisse werden in der Abbildung 6-1 und Abbildung 6-2 angezeigt. Die vertikale Achse präsentiert die Anzahl von RAM Ressourcen in AVD. Die horizontale Achse präsentiert die Anzahl von Crowdsourcing-Nachfragen.

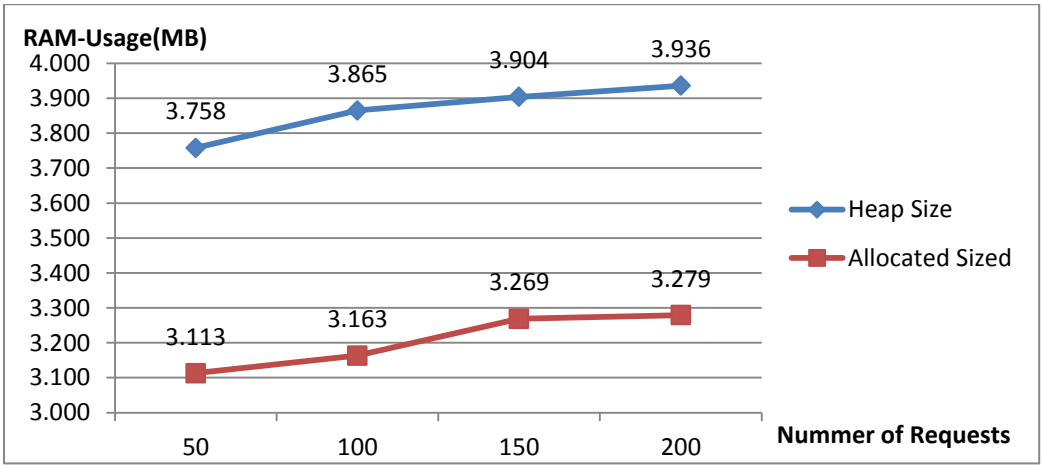


Abbildung 6-1: RAM-Situation von dem Kontrast-Crowdsourcing-Derivat.

Diese Abbildung spiegelt eine grundlegende Wirklichkeit wider. Mit dem Erhöhen von Nachfragen-Anzahl erhöht sich die zugeordnete RAM-Ressource. Wenn die Anzahl von Nachfragen mehr als 150 ist, ist der Anstieg von der zugeordneten RAM-Ressource nicht zu offensichtlich.

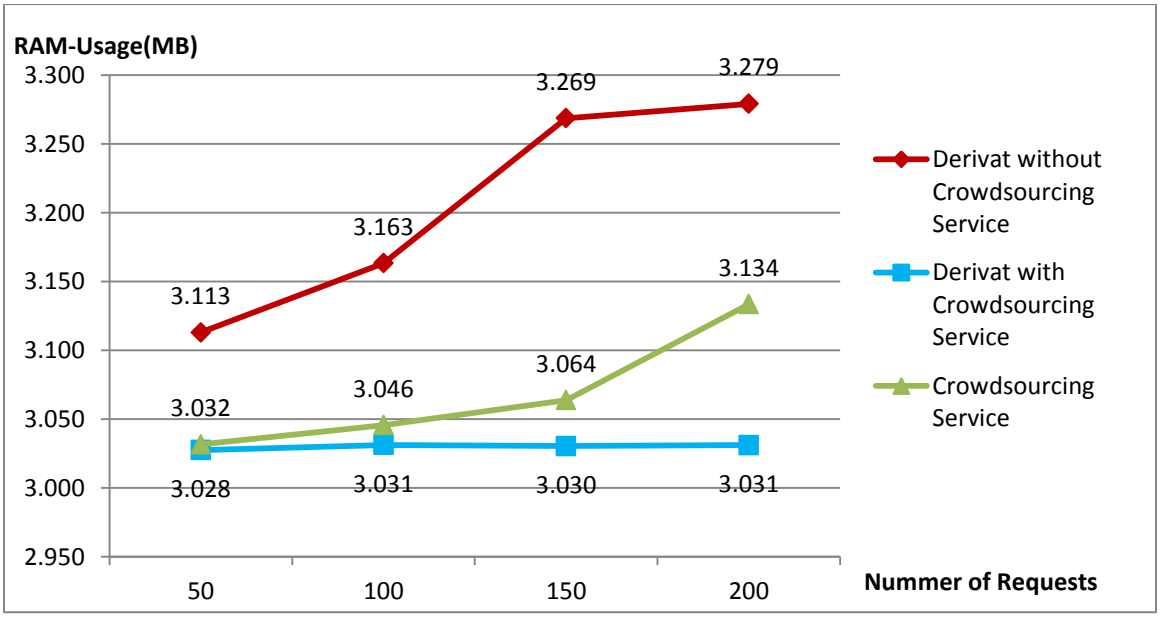


Abbildung 6-2: RAM-Verwendung-Situation von dem Kontrast-Crowdsourcing-Derivat(Rot), Crowdsourcing-Derivat(Blue) und Crowdsourcing Service(Hellgrün).

In dieser Abbildung werden die RAM-Verwendung-Situationen verglichen. Blau-Linie benutzt weniger RAM-Ressourcen als Rot-Linie. Außerdem bleibt die Anzahl der zugeordneten RAM-Ressourcen von Blau-Linie stabil, weil das Crowdsourcing-Derivat nur für das Senden von *Service-Anfragen* und Akzeptiren von *Service-Ergebnissen* verantwortlich ist. Der *Crowdsourcing Service* macht die Web-Kommunikation mit dem Server. Dieser Prozess kostet mehr RAM-Ressourcen. Genau aus diesem Grunde erhöhen sich die zugeordneten RAM-Ressourcen in *Crowdsourcing-Service*.

Aus dieser Abbildung wird es bewiesen, dass ein Crowdsourcing-Derivat mit Crowdsourcing-Infrastruktur die RAM-Ressource in Android-Geräten sparen kann. Aber ist die Anzahl der RAM-Ressourceneinsparung zu gering, z.B. ungefähr 0,2 MB RAM-Ressource. Zusätzlich braucht Crowdsourcing-Infrastruktur selbst noch einige RAM-Ressourcen, sowie zugeordnete RAM-Ressourcen für Crowdsourcing-Service. Es kann gerechnet werden. Angenommen, der Crowdsourcing-Service 3,0 MB RAM braucht, jedes Crowdsourcing-Derivat mit diesem Crowdsourcing-Service 0,2 MB RAM sparen kann, wenn die Anzahl von Crowdsourcing-Derivaten mehr in einem Android-Gerät als 15 wäre, könnte RAM-Ressourcen von den ganzen Crowdsourcing-Derivaten gespart werden. Natürlich ist dieses Evaluierung-Ergebnis nur für Android OS gültig.

6.3 Daten übertragen Evaluation

Neben begrenzten System-Ressourcen in mobilen Geräten gibt es noch eine Begrenzung für Crowdsourcing-Derivate in mobilen Geräten, sowie teure Daten-Übertragen-Kosten. Deshalb wird die Daten übertragungsmenge evaluiert.

In dieser generellen Crowdsourcing-Infrastruktur wird die Crowdsourced-Daten in Datentyp *String* in Datenformat *JSON* übertragen. Natürlich können die Daten in anderen Datenformaten übertragen werden, z.B. XML oder Paarweise-Web-Parameter. Die Anzahl von Daten zur Übertragung wird in XML- und JSON-Datenformat statisch verglichen.

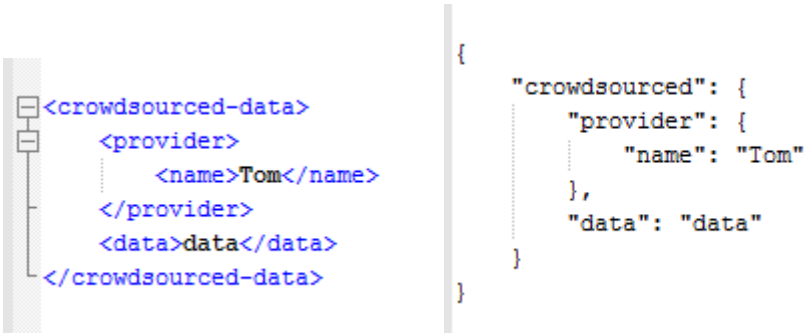


Abbildung 6-3: gleiche Daten in XML- und JSON-Datenformat, XML Datenformat(links), JSON Datenformat(rechts)

Aus der Abbildung 6-3 werden die Informationen in XML zusammen 103 Zeichen benutzt. Im Vergleichen dazu werden die gleichen Informationen in JSON zusammen 63 Zeichen benutzt. Wenn diese Informationen an den Server gesandt werden, kann JSON 40 Zeichen weniger als XML sparen. Deshalb kann die generelle Crowdsourcing-Infrastruktur die Daten übertragungsmenge zu einem bestimmten Grad sparen.

6.4 Modularisierung Evaluieren

Durch Modularisierung kann die Wiederverwendbarkeit einer Crowdsourcing-Infrastruktur erhöht werden. Am Nächsten wird die Modularisierung in dieser generellen Crowdsourcing-Infrastruktur evaluiert.

Im Kapitel Grundlagen wurden die Grundbegriffe von Modularisierung schon vorgestellt. Die Modul-Einheit in JAVA Plattform ist *Package*, d.h. ein *Package* ein Modul ist. Abhängig von dem Modularisierung-Prinzip P1 werden die Abhängigkeiten zwischen Modulen gemessen. Um Abhängigkeit-Diagramme zu erhalten, wird ein Werkzeug sowie CodePro[30] in Eclipse benutzt. Die Funktion sowie „*dependecies-analyze*“ in CodePro kann die Abhängigkeiten zwischen Projekten, Packages und Klassen in einer grafischen Abbildung anzeigen. Die Abhängigkeiten zwischen den Elementen in Diagrammen werden mit gerichteten Linien dargestellt. Die Nummer in einer Linie bedeutet die Anzahl von Abhängigkeiten. Die Elemente in einem Abhängigkeit-Graph werden drei Gruppen geteilt. Um einfach zu erkennen, wird jede Gruppe mit einer bestimmten Farbe markiert. Die Eigenschaften jeder Gruppe werden in Tabelle 6-2 dargestellt.

Name	Color	Description
internal elements	black	Elements that were either directly selected when the analysis was performed, or elements that are contained in directly selected elements. e.g, if you were to select the package java.util, both the package java.util and the class java.util.ArrayList would be internal elements.
external elements	gray	Elements that were neither selected nor contained in selected elements, but that are referenced by them. e.g., if you were to select the package java.util, the interface java.lang.Cloneable would be an external element in the type-level graph because the class java.util.ArrayList implements it and it is not within the java.util package.
subset of the internal elements	red	Those that are elements of a strongly connected component. A strongly connected component is a set of elements in which it is possible to reach all of the elements by following dependencies, starting from any element.

Tabelle 6-2: Eigenschaften von Elemente Gruppen in Analyse Dependencies in CodePro [17]

Nach der Beschreibung gibt es einen Zyklus in roten Elementen in einem Abhängigkeit-Graph. Die Modularisierung Prinzip P1 ist, dass die Module die losen Abhängigkeiten miteinander haben sollten. Nach diesem Prinzip sollten die Module die Zyklus-Abhängigkeiten und natürliche auch die bidirektionalen Abhängigkeiten vermeiden, d.h. keine *Packages* mit Rot Farbe in einem Abhängigkeit-Diagramm existieren sollten. Die Abbildung 6-4 zeigt die Package-Abhängigkeiten in *Crowdsourcing-Service*. Die Abbildung 6-5 zeigt die Package-Abhängigkeiten in serverseitige Crowdsourcing-Infrastruktur.

[30] https://developers.google.com/java-dev-tools/download-codepro [Zugriffszeit: 25.11.2014]

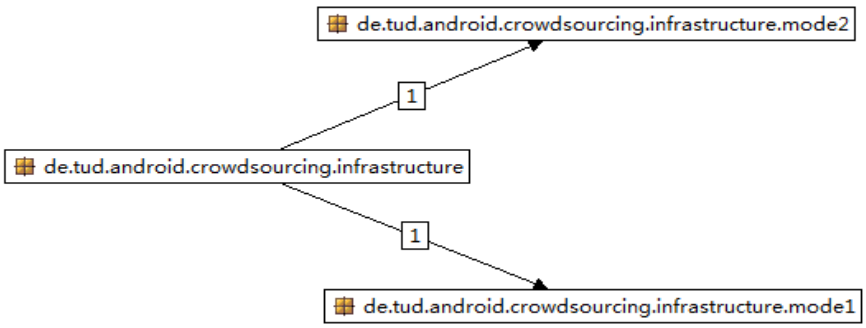


Abbildung 6-4: Abhängigkeiten von *Crowdsourcing-Service*

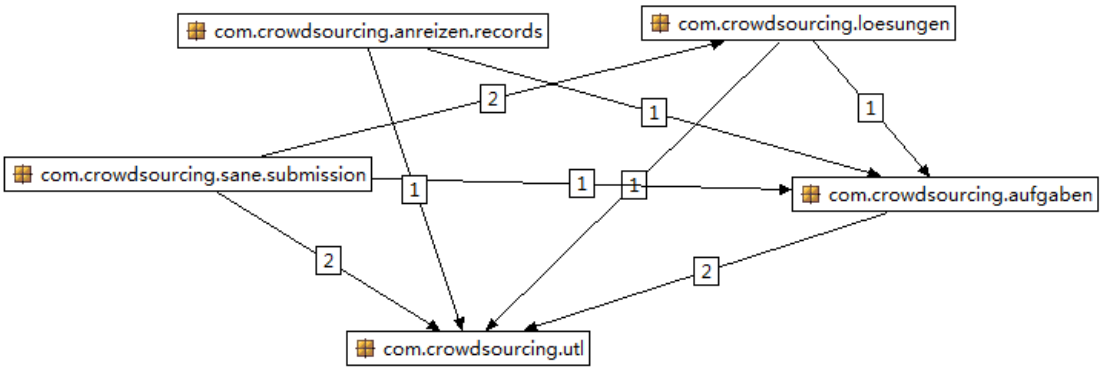


Abbildung 6-5: Abhängigkeiten in serverseitiger Crowdsourcing-Infrastruktur

Aus den beiden Abbildungen können zwei Punkte zusammengefasst,

- Es gibt keine stark verbundene Module sowie *Packages* in dieser generellen Crowdsourcing-Infrastruktur[31].
- Die Abhängigkeiten zwischen Modulen sind gering.

Der *Crowdsourcing-Service* macht hauptsächlich die synchrone Web-Kommunikation zwischen Client und Server. Die Architektur von *Crowdsourcing-Service* ist nicht komplex. Serverseitige Crowdsourcing-Infrastruktur verwendet RESTful-Technologien. Ein *Package* präsentiert eine völlige Ressource. Darin gibt es nicht nur Daten-Objekte, sondern auch die Operationen dieser Daten-Objekte. Außerdem gibt es noch die Schnittstelle, die die ankommenden Web-Anfragen verarbeiten können. Deshalb präsentiert die Abhängigkeiten zwischen unterschiedlichen *Packages* nur die Beziehungen zwischen Daten-Objekten. Weil die in Abbildung 5-5 aus Seite 63 angezeigte Daten-Objekt-Struktur einfach ist, sind die Abhängigkeiten zwischen Modulen gering.

[31] Es gibt kein Abhängigkeit-Diagramm über *Crowdsourcing-Library*, weil es nur ein *Package* darin gibt. Es gibt kein Abhängigkeit-Diagramm über den Proxy SANE, weil SANE als eine *Black Box* in Crowdsourcing-Infrastruktur verwendet wird.

6.5 Zusammenfassung

In diesem Kapitel werden die Anforderungen für eine generelle Crowdsourcing-Infrastruktur überprüft. Die Überprüfen-Ergebnisse werden in Tabelle 6-3 auf Seite 84 zusammengefasst. Die meisten Anforderungen werden erfüllt. Die Probleme passieren bei Kommunikation mit Proxy SANE. Trotzdem können die verbleibenden Komponenten gut funktionieren.

Nach der Überprüfung von Anforderungen wird die Leistung dieser Crowdsourcing-Infrastruktur getestet. Der Test wird in einem Android Emulator ausgeführt. In dem Test wurden die verwendeten RAM-Ressourcen gemessen. Die gemessenen Ergebnisse zeigen, dass die von Crowdsourcing-Infrastruktur gesparte RAM-Ressource nicht signifikant und zu gering ist. Wenn die RAM-Ressource von Crowdsourcing-Infrastruktur sowie *Crowdsourcing-Service* einschließlich berechnet wird, kann die gesparte RAM-Ressource in einem Crowdsourcing-Derivat ausglichen werden. Natürlich ist dieses Ergebnis nur für Android OS gültig. Um ein allgemeines Ergebnis zu bekommen, z.B. RAM in mobilen Geräten sparen, sollte diese Crowdsourcing-Infrastruktur in mehreren mobilen Systemen evaluiert werden. Aber eine Crowdsourcing-Infrastruktur, die auf dem Konzept dieser Diplomarbeit basiert, in einem bestimmten mobilen Operation-System ist notwendig, z.B. eine in Apple iOS implementierte Crowdsourcing-Infrastruktur.

Obwohl diese Crowdsourcing-Infrastruktur nicht viel RAM in einem mobilen Geräte sparen kann, kann sie durch eigene modularisierte Komponenten die Entwicklung eines Crowdsourcing-Derivats vereinfachen und wiederverwendet werden.

Zur Evaluation von Daten-Übertragen-Sparen wird die Daten-Anzahl in XML und JSON statisch verglichen. Im Vergleichen zu XML kann Crowdsourcing-Infrastruktur die Datenübertragungsmenge in JSON sparen.

Zum Ende der Evaluation wird die Modularisierung in der Crowdsourcing-Infrastruktur evaluiert. Diese Evaluierung basiert auf die Analysierung von Packages-Abhängigkeiten. Durch Werkzeug „CoedPro“ werden die Packages-Abhängigkeiten in einem Diagramm gezeigt. Die Ergebnisse zeigen, dass es keine stark verbundene Module sowie *Packages* in Crowdsourcing-Infrastruktur gibt. Deshalb wird die Crowdsourcing-Infrastruktur gut Modularisiert.

In dem nächsten Kapitel wird Auswerten und Ausblich dieser Crowdsourcing-Infrastruktur vorgestellt.

Nr.	Anforderung für Crowdsourcing-Infrastruktur	Situation
1.a	Als eine Bibliothek in jedem Crowdsourcing-Derivat wieder verwendet werden.	erledig
1.b	Dadurch unterschiedlichen Anfragen für Crowdsourcing-Infrastruktur-Service generiert, gesandt werden.	erledig
1.c	Unterschiedliche Typen und Formate von Crowdsourced-Daten unterstützen.	Teilweise
1.d	Dadurch die Ergebnisse von Crowdsourcing-Infrastruktur-Service erfasst und erkennt werden.	erledig
1.e	Davon individuelle Schnittstellen für Verarbeitung der Service-Ergebnisse angeboten werden.	erledig
2.a	Crowdsourcing-Infrastruktur von Crowdsourcing-Derivaten in einem mobilen Gerät getrennt sollte.	erledig
2.b	Crowdsourcing-Infrastruktur als einen Service spielen.	erledig
2.c	Einzige Instanz von Crowdsourcing-Infrastruktur in Laufzeit in mobilen Geräten garantieren.	erledig
2.d	Mehrere Aufrufe aus unterschiedlichen Crowdsourcing-Derivaten verwalten.	erledig
2.e	Automatisch Crowdsourcing-Infrastruktur selbst schließen kann, falls es keine Aufrufe von Crowdsourcing-Derivaten mehr gibt.	erledig
2.f	Ein Crowdsourcing-Derivat muss direkt den Crowdsourcing-Service aufrufen. Der Service muss die Ergebnisse direkt an dieses Derivat zurücksenden.	Teilweise
3.a	Aus losen Komponenten bestehen.	Erledigt
3.b	Unterschiedlichen Daten-Arten unterstützen.	Teilweise
3.c	Grundlegende Operationen von Daten-Objekten realisieren.	Erledigt
3.d	Unterschiedliche Daten-Objekte in unterschiedlichen Derivate dynamisch definiert werden könnten.	erledig
3.e	Crowdsourcer verwalten.	Nicht erledig
3.f	Crowdsourcer kann anonym Crowdsourced-Daten vorlegen.	Nicht erledig
4.a	Kommunikation zwischen Crowdsourcing-Derivat und –Service asynchron sein.	erledig
4.b	Kommunikation zwischen Client und Server synchron sein.	erledig
4.c	Allgemeines Web-Protokoll zwischen Client und Server verwenden.	erledig

Tabelle 6-3: Anforderungen Überprüfen

7 Zusammenfassung und Ausblick

In diesem Kapitel wird die ganze Diplomarbeit zusammengefasst. Danach werden die möglichen zukünftigen Arbeiten vorgestellt.

7.1 Zusammenfassung

Um die Ressourcen in mobilen Geräten zu sparen und um die Entwicklung von Crowdsourcing-Derivaten zu vereinfachen, wird eine generelle Crowdsourcing-Infrastruktur in dieser Diplomarbeit entwickelt und bekommen.

Die ganze Crowdsourcing-Infrastruktur basiert auf eine Crowdsourcing Definition 10 und wird in Android OS implementiert. Die ganze Crowdsourcing-Infrastruktur besteht aus 4 Teilen, sowie Crowdsourcing-Library, -Service, SANE und Crowdsourcing-Server. Crowdsourcing-Library wird in jedem Crowdsourcing-Derivat eingebettet. Damit kann ein Crowdsourcing-Derivat den *Crowdsourcing-Service* benutzen. Crowdsourcing-Service ist eine alleine Android-Anwendung und für die Web-Kommunikation zwischen Client und Server verantwortlich. Nach einmaliger Installierung von Crowdsourcing-Service kann er alle Crowdsourcing-Derivate bedienen. Proxy sowie SANE ist für Nutzerverwaltung verantwortlich. Mit SANE können Crowdsourcer anonym eigene Crowdsourced-Daten abgeben. Natürlich können sie auch direkt an einen Crowdfunding-Server schicken. Die serverseitige Crowdsourcing-Infrastruktur basiert auf das RESTful-Pattern. Durch Jersey wird ein URI beschrieben. Jedes Daten-Objekt in Crowdsourcing wird durch JDO als eine Ressource unter einem URI gespeichert und verwaltet. Die vorhabende Crowdsourcing-Infrastruktur enthält hautsächlich drei Daten-Objekte, sowie *Aufgabe, Lösung* und *Anreiz-Rekord*. Aus diesen drei grundlegenden Daten-Objekten können weitere Daten-Objekte und Funktionen entstanden werden. Obwohl die SANE nicht erfolgreich in dieser Crowdsourcing-Infrastruktur integriert wurde, wurden schon die Schnittstellen und das Daten-Objekt sowie SANE-Submission für SANE entwickelt und verbleiben. In der ganzen Crowdsourcing-Infrastruktur wird das allgemeine HTTP-Protokoll verwendet. Die Daten zur Übertragung werden in JSON formatiert. Die Eigenschaften sowie Vorteile und Nachteile dieser Crowdsourcing-Infrastruktur werden in der Tabelle 7-1 und Tabelle 7-2 auf Seite 90 aufgezählt.

In der Evaluation wird zuerst die Leistung der clientseitigen Crowdsourcing-Infrastruktur untersucht. Durch Analysierung von Ergebnissen wird es gefunden. Der Crowdsourcing-Service kann die RAM-Ressourcen in einem Android-Emulator sparen, aber die Anzahl von gesparten Ressourcen ist zu gering. Danach wird ein Punkt durch statischen Vergleich zwischen JSON und XML bekommen, dass Daten-Übertragen mit JSON den Netzverkehr sparen kann. Die Analysierung von Abhängigkeiten zwischen *Packages* in der Crowdsourcing-Infrastruktur zeigt, dass diese Infrastruktur schon gut modularisiert wurde, d.h. die Module sowie *Packages* in Crowdsourcing-Infrastruktur effektiv wiederverwendet werden kann.

7.2 Ausblick

Wegen der Diversität von Crowdsourcing-Derivaten kann die Crowdsourcing-Infrastruktur dieser Diplomarbeit die Anforderungen aller Crowdsourcing-Derivaten nicht völlig erfüllen. Aber auf dieser Basis sowie Crowdsourcing-Infrastruktur können Entwickler weiter eigene Funktionen oder Applikation-Schnittstellen in einem neuen Crowdsourcing-Derivat entwickeln, z.B. abhängig von einem Nutzer-Namen die

„*Lösung-Objekte*“ lesen. Das spiegelt auch die Erweiterbarkeit dieser Crowdsourcing-Infrastruktur wider.

Es gibt mehrere Erweiterung- und Verbesserungsmöglichkeiten für diese Crowdsourcing-Infrastruktur. Im Folgenden werden einige Möglichkeiten aufgezählt.

Möglichkeit 1: Die Funktionen in *Crowdsourcing-Servi*ce erweitern, z.B. unterschiedlichen Kommunikation-Moden entwickeln.

In dieser Crowdsourcing-Infrastruktur wurden nur zwei Kommunikation-Moden sowie SANE-Mode und Direkt-Server-Mode unterstützt. Diese beiden Moden basieren nur auf HTTP-Protokoll. Natürlich könnte eine neue Mode hinzugefügt werden, z.B. Mode 3 auf FTP-Web-Protokoll basiert.

Möglichkeit 2: Die Potenzial von SANE weiter unterstützen**.**

In dieser Crowdsourcing-Infrastruktur wurden nur die Funktionen sowie Nutzerverwaltung und anonyme Submission verwendet. Die SANE bietet noch mehr Funktionen, z.B. Crowdsourced-Daten signieren und nach Nutzer-Geoinformationen die nächsten SANE-Proxy finden. Um diese Möglichkeit zu realisieren, soll das Modul sowie Package „*crowdsourcing.infrastructure.mode1*“ entwickelt werden.

Möglichkeit 3: Mehr Crowdsourcing-Funktionen entwickeln, sowie Schnittstellen erweitern.

In dieser Crowdsourcing-Infrastruktur werden die grundlegenden Daten-Operationen angeboten, z.B. eine Lösung einer Aufgabe abgeben und löschen. Diese Operationen konzentrieren sich nur auf die begrenzten grundlegenden Daten-Objekte. Offensichtlich sind diese Funktionen nicht ausreichend für die Entwicklung eines neuen Crowdsourcing-Derivats, z.B. in einem Derivat kann Crowdsourcer die *Lösungen-Objekte* unter einem Nutzer-Namen anfragen. Die Realisierung dieser Möglichkeit ist einfach. Ein Attribut in *Lösung-Daten-Objekt* sollte Nutzer-Name sein. Dann wird eine Schnittstelle für diese Funktion programmiert. Schließlich wird diese Operation einem URI z.B. *http://server/crowdsourcing/lösung/anfragen/username* zugewiesen.

Möglichkeit 4: Mehr Daten-Objekte hinzufügen.

Eine Crowdsourcing-Funktion ist von Daten-Objekten abhängig, z.B. eine Funktion sowie „*Ranking-User-Contributions*“ braucht das Ranking-Data-Objekt. In dieser Crowdsourcing-Infrastruktur gibt es nur drei grundlegende Data-Objekten, sowie *Aufgabe, Lösung* und *Anreiz-Rekord*. Um diese Operation zu realisieren, muss eine neue Data-Ressource sowie Ranking-Data-Objekt modelliert und in diese Infrastruktur hinzugefügt werden. Natürlich sind die dazu gehörenden Operationen notwendig, z.B. CRUD.

Möglichkeit 5: Autorisieren- und Authentifizieren-Komponente auf Serverseite hinzufügen.

Die vorhandene Crowdsourcing-Infrastruktur hat keine Komponenten für Autorisierung und Authentifizierung. Jede Person kann die Daten unter einem URI verwalten.

Möglichkeit 6: Leistung von clientseitiger Crowdsourcing-Infrastruktur verbessern.

Weil die clientseitige Crowdsourcing-Infrastruktur in einem mobilen Gerät läuft, ist sehr notwendig, Systemressourcen zu sparen. Während der Entwicklung dieser Crowdsourcing-Infrastruktur wurde nur die Realisierung von dem Konzept und den Funktionen konzentriert. Es gab keine Optimierung von Leistungen. Deshalb gibt es noch Anforderungen oder Motivationen sowie Leistung-Optimierung dieser Crowdsourcing-Infrastruktur.

Möglichkeit 7: in mehreren mobilen Betriebssystemen unterstützen.

Die vorhandene Crowdsourcing-Infrastruktur wurde nur in Android OS implementiert. Die Implementierung benutzte mehr von Android OS abgehängte Schnittstellen.

Nach einigen Verbesserungen und Erweiterungen wäre diese Crowdsourcing-Infrastruktur eine Infrastruktur nicht mehr, sondern wäre ein Crowdsourcing-Framework. Durch dieses Framework könnte ein Crowdsourcing-Derivat mehr einfacher entwickelt werden.

Komponente	Vorteile
Crowdsourcing-Library	• Individuelle Schnittstellen für das Akzeptieren von Service-Ergebnissen definieren. • Service-Anfragen dynamisch erstellen und unterschiedlichen Service-Antworten erkennen. • Asynchrone Kommunikation mit Crowdsourcing-Service.
Crowdsourcing-Service	• einzigartiger Service. Damit können Ressourcen in mobilen Geräten sparen. • Crowdsourcing-Derivate brauchen nicht den Service zu verwalten, weil Service selbst schließen und die Ankommenden Anfragen verwalten kann.
SANE	• Standard Web-Service.
Crowdsourcing-Server	• Modularisierte Module einfach wieder zu verwenden. • Erweiterbare Daten-Objekte. Man kann die Attribute in einem Daten-Objekt verändern. • Kaskade Operationen, z.B. Löschen.

Tabelle 7-1: Vorteile der Crowdsourcing-Infrastruktur

Komponente	Nachteile
Crowdsourcing-Library	• Manuell Service-Anfragen senden. Deshalb sind die Service-Anfragen sowie *Intents* sichtbar. Das zerstört die Modularisierung-Prinzip P2 „*Information Hiding*“
Crowdsourcing-Service	• Service-Antwort verzögern, wenn es viele Aufgaben in der „Worker-Queue“ im *IntentService* gibt, werden die danach ankommenden Aufgaben verzögert. • Die *Service-Anfragen* können nicht parallel verarbeitet werden. • Wenn Service ausfällt, gehen alle Anfragen verloren.
Crowdsourcing-Server	• Begrenzte Daten-Objekte. Diese Daten können nicht eine komplizierte Crowdsourcing weiter unterstützen, • Operationen konzentrieren nur auf einfache Daten-Operationen, sowie CRUD. • Normale HTTP Protokoll verwenden. Das bringt ein Sicherheit-Risiko. • Es gibt keine Autorisation- und Authentication-Komponenten.

Tabelle 7-2: Nachteile der Crowdsourcing-Infrastruktur

Literatur

[1] Gigwalk, „Gigwalk - Home,“ Gigwalk, 2012. [Online]. Available: http://www.gigwalk.com/. [Zugriff am 12 2014].

[2] FOSSGIS e.V., „OpenStreetMap - Deutschland,“ FOSSGIS e.V., [Online]. Available: http://www.openstreetmap.de/index.html. [Zugriff am 12 2014].

[3] P. Sloane, A Guide to Open Innovation and Crowdsourcing Expert tips and advice, Great Britian and the United States: Kogan Page Limited, 2011, pp. 15-15.

[4] D. Geiger, M. Rosemann and E. Fielt, "Crowdsourcing Information Systems - A Systems Theory Perspective," *ACIS 2011 Proceedings,* no. 978-1-74210-239-9, 2011.

[5] M. Stevens and E. D'Hondt, "Crowdsourcing of Pollution Data using Smartphones," 2010.

[6] T. C. Hara, „Decentralised Approach for a Reusable Crowdsourcing Platform Utilising Standard web servers“.

[7] K. Knoernschild, Java Application Architecture, United States of America: Pearson Education,Inc, 2012, pp. 17-82.

[8] W. Stevens, G. Myers und L. Constantine, „Strunctured design,“ *IBM Systems Journal,* Bd. 13, pp. 115-139, 1974.

[9] D. Parnas, „Information Sistribution Aspects of Design Methodology,“ in *In IFIP Congress(1),* 1971.

[10] M. Conway, „How do Committees Invent?,“ *Datamation Journal,* pp. 28-31, 1968.

[11] R. Martin, Agile Sofware Developemnt, principles, patterns, and practices., 1st Edition Hrsg., Prentice Hall, 2002.

[12] A. Adepetu, K. A. Ahmed, Y. A. Abd, A. A. Zaabi and D. Svetinovic, "CrowdREquire:A Requirements Engineering Crowdsouricng Plattform," in *2012 AAAJ Spring Symposium Series*, 2012, March.

[13] X. Peng, M. Ali Babar and C. Erbert, "Collaborative Software Development Plattforms for Crowdsourcing," *Software,IEEE,* no. 0740-7459 , pp. 30 - 36, Mar.-Apr. 2014.

[14] Google Project Hosting, "Google-Gson," Google Inc., 01 2001. [Online]. Available:

http://code.google.com/p/google-gson/. [Accessed 30 01 2015].

[15] Google, „what is Google App Engine?,“ Google, [Online]. Available: https://cloud.google.com/appengine/docs/whatisgoogleappengine. [Zugriff am 01 2015].

[16] Oracle Corporation, „Jersey-RESTful Web Services in Java,“ Oracle Corporation, [Online]. Available: https://jersey.java.net/. [Zugriff am 01 2015].

[17] G. Developers, „Dependencies Analyse,“ Google, 07 März 2012. [Online]. Available: https://developers.google.com/java-dev-tools/codepro/doc/features/dependencies/dependencies. [Zugriff am 01 11 2014].

[18] G. Chatzimilioudis, U. o. C. N. Dept.of Comput.Sci., A. Konstantinidis, C. Laoudias and D. Zeinalipour-Yazti, "Crowdsourcing with Smartphones," *Internet Computing, IEEE,* no. 1089-7801 , pp. 36-44, OCT 2012.

[19] T. Springer, „Poster des Project Mapbiquitous,“ 2014.

[20] T. C. Hara, *Towards A Reliable Architecture For Crowdsourcing In Contex Of The MapBiquitous Project*, 2012.

[21] G. P. Hosting, „google-gson,“ Google, [Online]. Available: http://code.google.com/p/google-gson/. [Zugriff am 11 2014].

[22] The Apache Software Foundation, „Apache HttpComponents - HttpClient for Android,“ Apache HttpComponent, 2005. [Online]. Available: https://hc.apache.org/httpcomponents-client-4.3.x/android-port.html. [Zugriff am 01 12 2014].

[23] Developer, Android, „Application Fundamentals,“ [Online]. Available: http://developer.android.com/guide/components/fundamentals.html. [Zugriff am 01 11 2014].

[24] Android Developer, „AsyncTask | Android Developers,“ [Online]. Available: http://developer.android.com/reference/android/os/AsyncTask.html. [Zugriff am 12 2014].

[25] Google, „Using JDO 2.3 with App Engine,“ Google, 09 2014. [Online]. Available: https://cloud.google.com/appengine/docs/java/datastore/jdo/overview. [Zugriff am 10 01 2015].

A.Anhang

A.1 Klassen Diagramme in Crowdsourcing Infrastruktur

Die Klassen-UML-Diagrammen werden durch ObjektAid 1.1.5 generiert. ObjektAid[32] ist ein Werkzeug für UML-Diagramme in Eclipse.

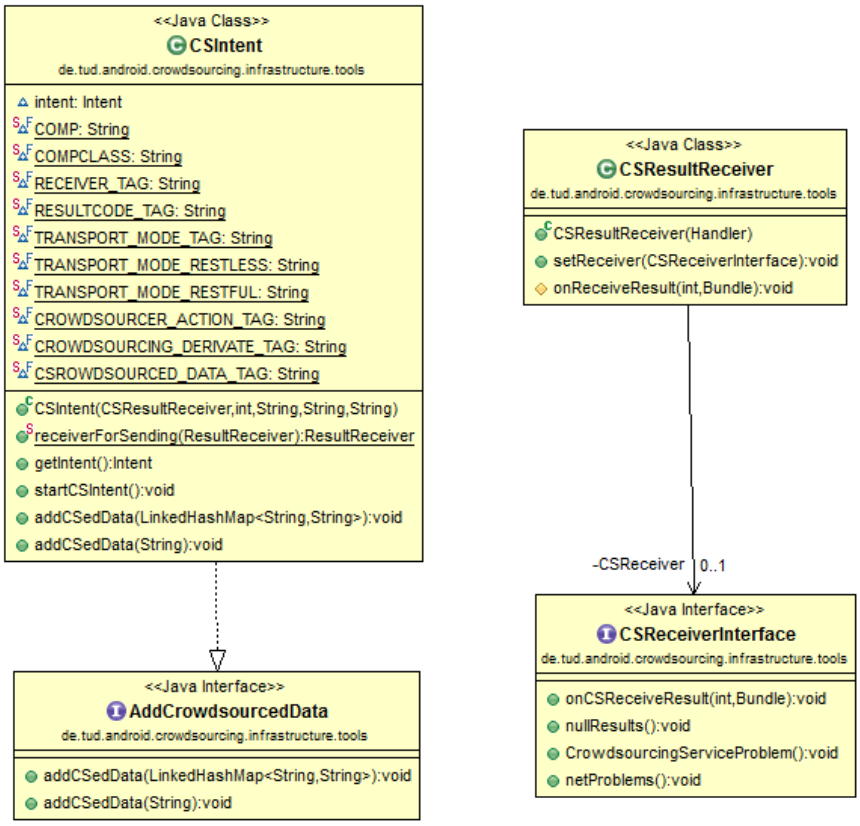


Abbildung A-1: Screenshot für Klassen-UML-Diagramme von Crowdsourcing-Library

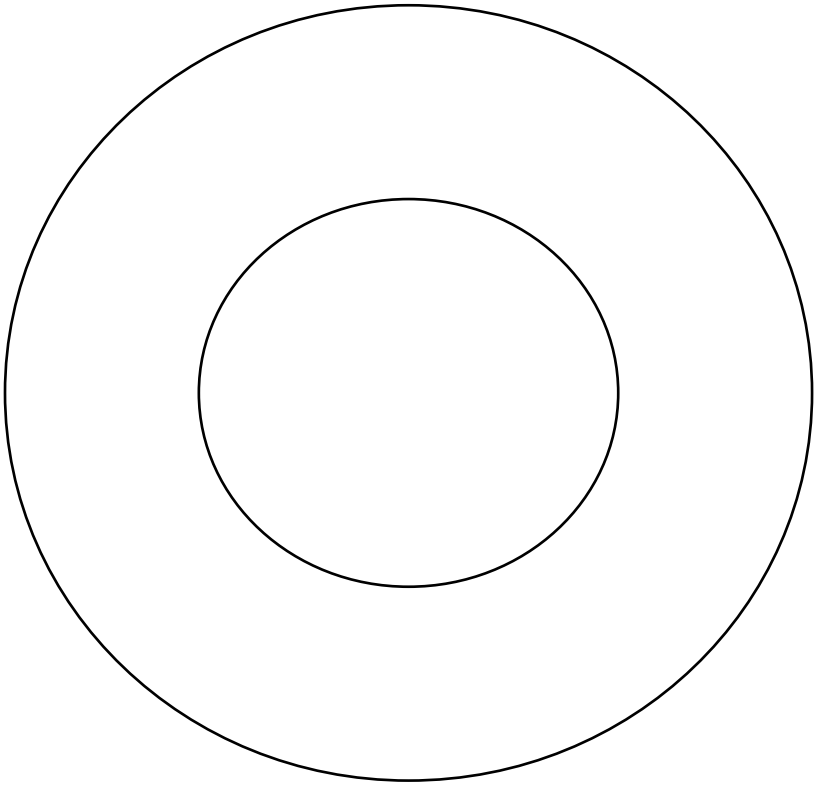
[32] http://www.objectaid.com/home [Zugriffszeit:02.01.2015]

A.2 Datei auf dem CD

Dateiverzeichnis:

Diplomarbeit.pdf

Programm



Danksagung

An dieser Stelle möchte ich all jenen danken, die ihre fachliche und persönliche Unterstützung zum Gelingen dieser Diplomarbeit beigetragen haben.

Mein Dank gilt zunächst meinem Betreuer Tenshi Hara und Thomas Springer. Sie sind sehr nett und hilfreich. Sie haben mir viele wichtige und sinnvolle Vorschläge gegeben. Diese Vorschläge beziehen sich nicht nur auf Fachkenntnis sondern auch die Forschung-Erfahrungen. Ohne ihre Hilfe kann ich nicht diese Diplomarbeit erledigen.

Des Weiteren möchte ich mich für die jahrelange Unterstützung meiner Eltern und Freundin Dan Li bedanken. Sie haben mich viel unterstützt. Mein Freund Lei Zhang hat mir geholfen, meine Diplomarbeit zu überarbeiten.