HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

ita Institute for Internet Technologies and Applications

NGSON with Peer-to-Peer

Bachelorarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Frühlingssemester 2012

Autor(en):	Heinrich Muralt, Marcel Steiner
Betreuer:	Prof. Dr. Josef M. Joller
Projektpartner:	Institut für Internet-Technologien und -Anwendungen

DANKSAGUNG

Für die Unterstützung während der Bachelorarbeit möchten wir folgenden Personen einen besonderen Dank ausrichten:

PROF. DR. JOSEF M. JOLLER
für sein wertvolles und konstruktives Feedback

SANDRA FREI
für ihre kompetente Hilfe im Bereich NGSON

LYDIA MURALT, BENEDIKT KNOBEL, SIMONE SCHNEIDER
für das Korrekturlesen der Bachelorarbeit

Heinrich Muralt und Marcel Steiner: *NGSON with Peer-to-Peer*
Fortsetzungsarbeit der Studienarbeit MobileCloud
Bachelorarbeit, © Juni 2012

AUFGABENSTELLUNG

1.1 BETREUER

1.1.1 BETREUER HSR:

- Prof. Dr. Josef M. Joller, Abteilung für Informatik, HSR/FHO

1.2 AUSGANGSLAGE, PROBLEMBESCHREIBUNG

Mobile Devices (iPhones, iPads, Android basierte Geräte, zum Teil in Fahrzeuge eingebaut) sind sehr stark verbreitet und verfügen über häufig ungenutzte oder ungenügend genutzte Fähigkeiten und Kapazitäten.

Falls die mobilen Geräte Aufgaben lösen würden bzw. ihre allenfalls vorhandenen Services anbieten könnten, hätte man in fast jeder Gruppe von mobilen Geräten ein enormes Potential an Services zur Verfügung („Service Ecosystem“).

Damit mobile Services breiter nutzbar werden können, müssen wichtige Aspekte einer mobilen Kooperation und Kollaboration beachtet werden:

- Privatsphäre: Teilnehmer müssen jederzeit wissen, was auf den mobilen Geräten zurzeit aktiv vor sich geht.
- Sicherheit: Teilnehmer müssen die Gewissheit haben, dass keine unerlaubten und unerwünschten Besucher auf den mobilen Geräten aktiv sind.

Existierende Services, welche in dieselbe Richtung gehen sind:

- Viber (auf iPhone und Android): Diese Software stellt Basisdienste für die Kommunikation zur Verfügung.
- Whats Up

Unser Ziel ist weitergehend in Richtung "semantischer" Information (Topics, Social Networks, …), „Next Generation Service Overlay Networks“ und „Mobile Service Ecosystem“.

1.3 ORGANISATORISCHE RAHMENBEDINGUNG

Die Arbeit ist eine Fortsetzungsarbeit der Studienarbeit *MobileCloud*.

1.4 Technologische Rahmenbedingungen

Als externe Datendarstellung kann XML verwendet werden. Damit lässt sich die Datendarstellung plattformneutral realisieren. Sicherheitsaspekte sollen fürs erste nicht berücksichtigt werden (keine Verschlüsselung der übermittelten Daten). Dies kann zu einemspäteren Zeitpunkt leicht nachgeholt werden.

Die Lösung soll sich auf eine mobile Plattform beschränken, konkret Android. iPhone wird ausgeklammert, da diese Geräte zu proprietär sind.

1.5 Ziele der Arbeit

Hauptziel dieser Arbeit ist es zu untersuchen, inwiefern mobile Geräte miteinander vernetzt werden können, unter Ausnutzung aller verfügbaren Netzwerk-Technologien (IP basierte Netzwerk-Technologien, mobile Telefonie- basierte Technologien). Zusätzlich soll aufgezeigt werden, wie und welche Services auf dieser Plattform genutzt werden können.

Untergeordnetes Ziel ist der Aufbau eines mobilen Clusters aus miteinander kommunizierenden mobilen Geräten.

Ein weiteres Ziel ist der anschliessende Ausbau in Richtung "Service Cloud", also die Anbietung unterschiedlichster Services. Bei diesen Services kann es sich um Services handeln, welche bereits irgendwo auf einem Netzwerk-Partner vorhanden sind und benutzt werden können.
Dieser Teil der Arbeit orientiert sich am neuen Standard NGSON – Next Generation Service Overlay Networks.

1.6 Zur Durchführung

Die erfolgreiche Bearbeitung dieser Aufgabe bedingt Grundkenntnisse in der Programmierung verteilter Systeme voraus (praktische Beispiele, nicht bloss Literaturkenntnisse) sowie die Bereitschaft sich in neue und neuste Ergebnisse einzuarbeiten (MobileAnwendungen, speziell ganz neue Möglichkeiten mit Android).

Mit dem Betreuer finden in der Regel wöchentliche Besprechungen statt. Zusätzliche Besprechungen sind nach Bedarf durch die Studierenden zu veranlassen. Das Sitzungsintervall ist flexibel: falls wenig oder keine Probleme auftreten können die Sitzungsintervalle vergrössert werden; falls Probleme auftauchen, werden die Intervalle verkleinert.

Alle Besprechungen sind von den Studenten mit einer Traktandenliste vorzubereiten und die Ergebnisse sind in einem Protokoll zu dokumentieren, das dem Betreuer per E-Mail zugestellt wird.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. An Meilensteinen gemäss Projektplan sind einzelne Arbeitsresultate in vorläufigen Versionen abzugeben.

1.7 Dokumentation und Abgabe

Wegen der beabsichtigten Verwendung der Ergebnisse in weiteren Projekten wird auf Vollständigkeit sowie (sprachliche und grafische) Qualität der Dokumentation erhöhter Wert gelegt.

Die Dokumentation zur Projektplanung und -verfolgung ist gemäss den Richtlinien der Abteilung Informatik anzufertigen. Die Detailanforderungen an die Dokumentation der Recherche- und Entwicklungsergebnisse werden entsprechend dem konkreten Arbeitsplan festgelegt.

Die Dokumentation ist vollständig auf CD abzugeben oder Zugriff auf eine Ablage (SVN, GIT).

Neben der Dokumentation sind abzugeben:

- alle zum Nachvollziehen der Arbeit notwendigen Ergebnisse und Daten (Quellcode, Buildskripte, Testcode, Testdaten usw.)
- Material für eine Abschlusspräsentation (ca. 20')

1.8 Termine

Siehe Terminplan auf der HSR Seite.

HSR Terminplan	Beginn der Studienarbeit Ausgabe der Aufgabenstellung durch die Betreuer
1. Semesterwoche	Kick-off Meeting
2. Semesterwoche	Abgabe der Projektplanung (Entwurf), einschliesslich ggf. zu beschaffender Hardware.
4. Semesterwoche	Abgabe eines vorläufigen Technologierechercheberichts und eines detaillierten Vorschlags für einen Arbeitsplan. Festlegung des Projektplans mit dem Betreuer.
6. Semesterwoche	Abgabe eines Umsetzungsvorschlags für die Experimentierumgebung (Funktionsumfang, Aufwandsabschätzung) Abstimmung und Festlegung mit dem Betreuer.
7. Semesterwoche	Abgabe Anforderungs- und Domainanalyse für die Experimentierumgebung
10. Semesterwoche	Review-Meeting Softwaredesign für die Experimentierumgebung
13. Semesterwoche	Vorstellung der Implementierung (Arbeitsstand)
HSR Terminplan	Abgabe der Kurzbeschreibung für zur Kontrolle an den Betreuer
HSR Terminplan	Abgabe des Berichtes an den Betreuer (bis 12:00 Uhr)

1.9 Literaturhinweise

Generelle Referenzen:

Siehe Vorlesung über Verteilte Software Systeme .

Weitere Literatur steht bei Bedarf zur Verfügung!

1.10 Beurteilung

Eine erfolgreiche Bachelorarbeit erhält 12 ECTS-Punkten (1 ECTS Punkt entspricht einer Arbeitsleistung von ca. 25 bis 30 Stunden). Für die Modulbeschreibung der Bachelorarbeit siehe https://unterricht.hsr.ch/staticWeb/allModules/10938_M_SAI.html

Gesichtspunkt	Gewicht
1. Organisation, Durchführung	1/5
2. Berichte (Abstract, Mgmt Summary, techn. u. persönliche Berichte) sowie Gliederung, Darstellung, Sprache der gesamten Dokumentation	1/5
3. Inhalt (*)	3/5

(*) Die Unterteilung und Gewichtung von 3. Inhalt wird im Laufe dieser Arbeit mit dem Studenten vereinbart.

Im Übrigen gelten die Bestimmungen der Abt. Informatik zur Durchführung von Studienarbeiten.

Rapperswil, den

Der verantwortliche Dozent

ERKLÄRUNG

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben,
- dass wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Ort, Datum: Rapperswil, 15.06.2012

Heinrich Muralt

Ort, Datum: Rapperswil, 15.06.2012

Marcel Steiner

Inhaltsverzeichnis

ABSTRACT

Viele Apps verwenden Services für das Erfüllen ihrer Aufgaben. Mehrere dieser Services sind mehrfach vorhanden, leisten aber den gleichen Dienst. Würden diese einerseits an einem Ort zusammengeführt und andererseits für Benutzer transparent nutzbar gemacht, so könnten die vorhandenen Ressourcen viel effizienter genutzt werden. Gleichzeitig könnten die registrierten Services für die Optimierung anderer Services verwendet werden. Ebenfalls denkbar ist, dass mobile Geräte selbst eigene Services anbieten.

Ein Ansatz für die Umsetzung dieser Idee bietet der IEEE 1903 NGSON Standard. Die vorliegende Bachelorarbeit zeigt durch die Umsetzung der wichtigsten Funktionen dieses Standards, wie eine solche Verwirklichung aussehen könnte.

Im ersten Teil wurde aufbauend auf der Studienarbeit *MobileCloud* das Framework bezüglich Fehlertoleranz und Stabilität des Netzwerks verbessert. Diese Erweiterungen umfassen unter anderem einen Keep-Alive und Hole Punching Algorithmus sowie ein klares und einfaches Interface für die Implementierung zusätzlicher Services.

Im zweiten Teil wurden drei zentrale Funktionen des NGSON Standards als Serveranwendung implementiert: das Verwalten, Verketten und kontextbasierte Ausführen von Services. Diese Serveranwendung kann über eine REST-Schnittstelle angesprochen werden und nutzt exemplarisch ein paar Webservices. Zudem besteht die Möglichkeit über das *MobileCloud*-Framework mit mobilen Geräten oder anderen NGSON Servern zu kommunizieren.

Im umgesetzten Beispiel kann ein Service, welcher Bilder zum Download anbietet, die Auflösung dieser Bilder mittels Nutzung anderer Services auf die optimale Grösse skalieren und in einer ZIP-Datei zusammenfügen. Somit kann bei minimalem Datenverkehr eine optimale, dem jeweiligen Nutzer angepasste Qualität angeboten werden.

Für die einfache Verwendung des Servers und das Aufzeigen seiner Funktionalität, wurde eine Android-App entwickelt. Damit lassen sich registrierte Services abrufen und ausführen.

Management Summary

Ausgangslage

Mobile Geräte haben in den letzten Jahren einen regelrechten Boom erlebt und sind aus der heutigen Gesellschaft nicht mehr wegzudenken. Mit der starken Verbreitung ist auch das Interesse an immer neueren und umfangreicheren Apps (Programmen, Aplikationen) gestiegen. Um diesen Anforderungen gerecht zu werden, greifen die Apps immer öfter auf Services zu, welche im Internet bereits vorhanden sind. Die meisten dieser Services werden über standardisierte Schnittstellen angesprochen. Verschiedene Apps können dabei die gleichen Services verwenden. Die meisten dieser Services beschränken sich jedoch auf das Abrufen von Informationen wie zum Beispiel das aktuelle Wetter oder den Fahrplan der Bahn. Es wäre auch möglich Services wie etwa das Komprimieren von Daten oder Konvertieren von Videos anzubieten. Solche Services werden jedoch kaum zur Verfügung gestellt, da ihre isolierte Anwendung meistens keine wirklichen Vorteile bringt. Dies würde sich allerdings ändern, sobald Services verkettet und zu neuen Services zusammengefügt werden könnten. Damit könnten zum Beispiel Videos vor dem Herunterladen auf die passende Auflösung und ein unterstütztes Dateiformat konvertiert werden. Ein Vorteil wäre, dass bei minimaler Ausnützung des Netzwerkes ein optimales Ergebnis erzeugt wird. Aufbauend auf der Studienarbeit *MobileCloud* aus dem Jahre 2011 soll dies im Rahmen dieser Bachelorarbeit mit dem IEEE 1903 Standard in einem Prototyp umgesetzt werden.

Vorgehen, Technologien

In der ersten Phase des Projektes wurde das bereits bestehende *MobileCloud*-Framework erweitert. Der Fokus liegt dabei auf der Erhöhung der Fehlertoleranz, der Möglichkeit globale Clusters zu erstellen und *MobileCloud* optimal für das Anbieten von Services auf der Android-Plattform zu integrieren. Für das Erstellen globaler Cluster wurde mittels eines Prototyps untersucht, welche bekannten Methoden im Mobilfunknetz der Schweiz verwendet werden könnten.

In einer zweiten Phase wurde auf der Basis des NGSON Standards ein Java Prototyp entwickelt, welcher die wichtigsten funktionalen Definitionen des Standards implementiert. Das *MobileCloud*-Framework wurde auf die Java-Laufzeitumgebung 6 (JRE6) portiert und eingebunden. Das ermöglicht das Anfordern und Ausführen von Services, welche in der Cloud verfügbar sind. Als Schnittstelle für die Verwaltung und Verwendung der Services wird das Jersey Framework verwendet. Die registrierten Services werden in einer relationalen Datenbank abgelegt und können so jederzeit gefiltert abgerufen werden. Für das Verketten von Services wird eine XML-Struktur definiert. Damit lassen sich Services definieren, welche aus bereits vorhandenen zusammengesetzt sind. Als Client wird eine App für Android entwickelt, welche es ermöglicht die registrierten Services abzurufen und zu verwenden.

ERGEBNISSE

Entstanden ist ein Prototyp NGSON-Node, welcher die wichtigsten Funktionen, die im NGSON Standard definiert sind, implementiert:

- Verwalten von Services (*Service Management*)
- Verkettung von Services (*Service Composition*)
- Kontext gestütztes Ausführen von Services(*Context Awareness*)

Um den funktionalen Umfang aufzuzeigen, wurden drei Services erstellt. Einer für das Suchen von Fotos im Internet, ein Zweiter für das Skalieren von Fotos und ein Dritter für das Zusammenfassen mehrerer Dateien in eine Archiv-Datei. All diese Services wurden anschliessend als verketteter Service zusammengehängt. Für den Benutzer nicht sichtbar, führt dieser Service alle drei Aufrufe durch und liefert als Resultat in Form von skalierten Bildern in einer Archiv-Datei zurück.

Für die Verwendung der registrierten Services wurde eine Android App entwickelt, die auf Tablet-Computern und Smartphones optimiert ist. Damit können die Services, welche im NGSON-Node registriert sind, abgerufen und ausgeführt werden. Durch die unterschiedlichen Geräte kann aufgezeigt werden wie die Services für das Endgerät optimierte Resultate liefern.

Die entstandene Software zeigt auf, welche Ziele der Standard anstrebt und wie diese umgesetzt werden könnten. Der Prototyp könnte in Folgeprojekten weiter ausgebaut und somit für die Öffentlichkeit nutzbar gemacht werden.

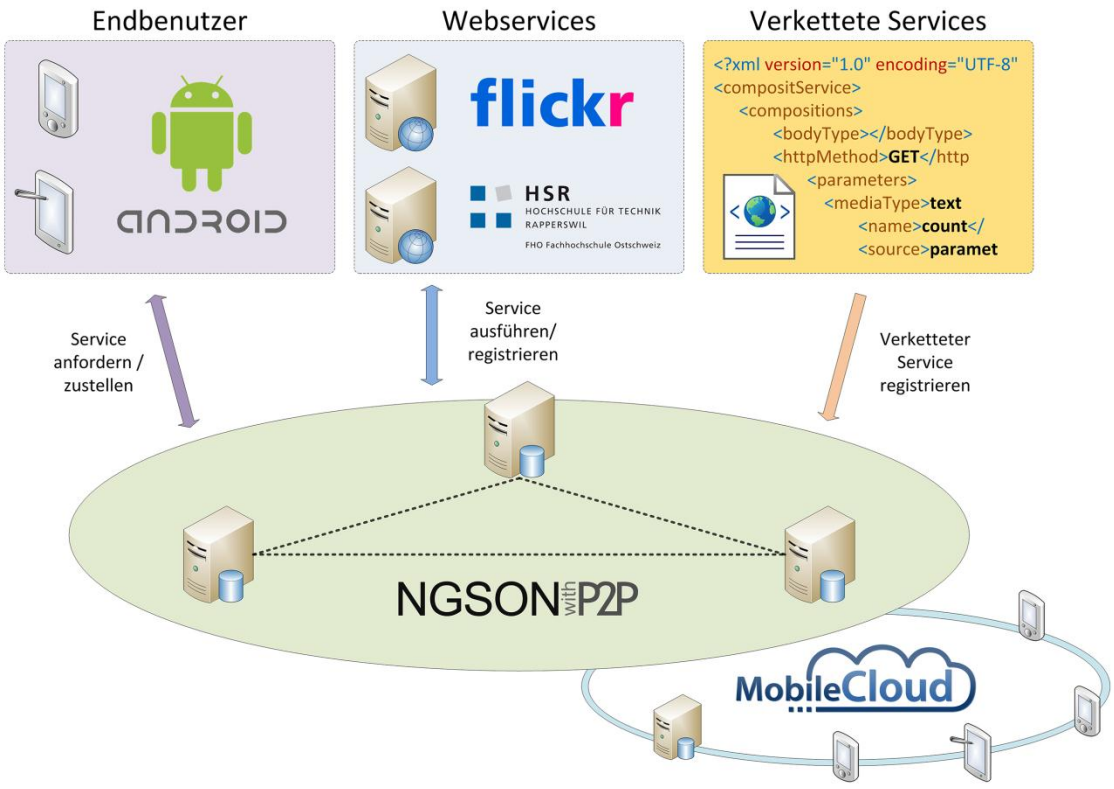


ABBILDUNG 1 ÜBERSICHT DER NGSON WITH PEER-TO-PEER KOMPONENTEN

1 Einleitung

Die Verbreitung mobiler Geräte wie Handys und Tablet-Computer nimmt weltweit stetig zu. Die Leistung und Fähigkeiten dieser Geräte haben über die Jahre zugenommen. Zudem sind sie handlicher und zu durchgestylten Modeaccessoires geworden. Als tägliche Begleiter änderte sich auch deren Einsatzgebiet. Durch die ständige Verbindung zum Internet wurde das einfache Telefon zu einem multifunktionalen, mobilen Büro, auf welchem man immer erreichbar ist und sogar unterwegs noch schnell kleine Arbeiten erledigen kann.

Mit diesen Änderungen stieg auch der Wunsch nach immer mehr und komplexeren mobilen Applikationen, sogenannte Apps. Diese Nachfrage wird mit dem Anbieten von verschiedensten Apps über eine Zentralle Stelle, den App Store, gestillt. Von Büroanwendungen bis zu komplexen Spielen wird heute ein breites Spektrum angeboten. Eine einzelne App bietet jedoch nur einzelne Services an und kann nur vom Entwickler erweitert werden.

In dieser Bachelorarbeit wird aufgezeigt, dass dies nicht so sein muss. Dafür wurde der NGSON[1] Standard der Organisation IEEE[2] verwendet, um Services zu verwalten und nutzen. Für die Verwendung dieser Services wurde eine App auf der Android Plattform erstellt.

1.1 Einführung

Durch die Möglichkeit mit den mobilen Geräten auf das Internet zuzugreifen, erweitert sich auch deren Einsatzgebiet. Mit diversen Apps werden heutzutage verschiedenste Services wie zum Beispiel das Betrachten von Videos, Verwalten von E-Mails oder Abrufen der Wetterprognose angeboten. Viele dieser Services werden bereits im Internet bereitgestellt, können jedoch ohne zusätzliche App nicht genutzt werden. Ein Beispiel, bei dem diese Einschränkung nicht besteht, ist der Webbrowser, über welcher verschiedene Services benutzt werden können. Auch ist es für jeden möglich seinen eigenen Service anzubieten. Das ist allerdings nur für Services möglich, welche auf HTML/CSS und Javascript basieren.

Es gibt üblicherweise viele ähnliche Services. Welcher schlussendlich für die aktuelle Situation am geeignetsten ist und wie dieser optimal genutzt werden kann, ist ohne Fachwissen oft nur schwer einschätzbar. Services, die ein auf Benutzer angepasstes Resultat liefern oder sogar personalisiert werden können, heben sind von anderen Services ab und haben daher einen Vorteil.

Ein weiterer grosser Vorteil würde die Verknüpfung verschiedener Services bringen. Ein Videoservice könnte die Videodaten vor dem Senden mittels anderer Services auf ein für den Benutzer optimiertes Format konvertieren. Dabei würde bei minimaler Ausnutzung des Netzwerkes dem Benutzer ein Video mit optimaler Qualität angeboten werden.

[1] Next Generation Service Overlay Network (NGSON) ist ein Standard der Organisation IEEE. Darin wird die funktionale Architektur von NGSON beschrieben [14].
[2] Institute of Electrical and Electronics Engineers (IEEE) ist ein weltweiter Berufsverband von Ingenieuren aus den Bereichen Elektrotechnik und Informatik mit Sitz in New York.

Dieses Dokument beschreibt nach einem kurzen Überblick über die verwendeten Begriffe den technischen Teil der Arbeit. Der Fokus liegt dabei bei der Umsetzung und den Ergebnissen. Im zweiten Teil wird das gesamte Projekt ausgewertet und einen kurzen Blick in die Zukunft gewagt.

1.2 Ziel

Das primäre Ziel dieser Bachelorarbeit war es aufbauend auf der Studienarbeit zu untersuchen, wieweit mobile Geräte unter Ausnutzung aller verfügbaren IP-basierten Verbindungen miteinander vernetzt werden können. Der Aufbau eines Cluster war dabei ein sekundäres Ziel. Dabei mussten folgende Punkte besonders beachtet werden:

- **Fehlertoleranz**: Ein Netzwerk ist beachtlich dynamisch und wird von vielen Faktoren beeinflusst. Damit in einem Cluster gearbeitet werden kann, braucht es eine hohe Fehlertoleranz.
- **Globaler Cluster**: Um die Verwendbarkeit von *MobileCloud* zu erhöhen, muss es die Möglichkeit geben, über die lokalen Cluster aus zu kommunizieren.
- **Nutzung von Services:** Es muss ohne grossen Aufwand möglich sein vorhandene Services zu nutzen und neue anzubieten.

Weiter musste untersucht werden, wie und welche Services auf den Geräten angeboten werden können. Dabei konnte es sich um Services handeln, welche bereits im Internet vorhanden waren oder zusätzlich erstellt wurden. Dieser Teil sollte sich am NGSON Standard orientieren. Dabei mussten folgende Punkte besonders beachtet werden:

- **Service Cloud**: Services müssen zentral verwaltet werden können, gleichzeitig aber auch so vielen Leuten wie möglich transparent zugänglich gemacht werden.
- **NGSON**: NGSON ist ein Standard, welcher das Anbieten von Services standardisiert. Dieser soll so weit umgesetzt werden, dass die Vorteile und Funktionsweise aufgezeigt werden können. Dabei soll der Fokus auf folgenden Punkten liegen:
    - Verwalten von Services (*Service Management*)
    - Verkettung von Services (*Service Composition*)
    - Kontext gestütztes Ausführen von Services(*Context Awareness*)

1.3 VORGEHEN

In einem ersten Schritt wurde das bereits bestehende *MobileCloud*-Framework überarbeitet und erweitert. Durch eine automatische Reorganisation des Netzwerkes konnte die Fehlertoleranz erhöht werden. Das Interface für alle Handler wurde nochmals überarbeitet und für die zukünftige Verwendung als Schnittstelle zu den Services optimiert.
Des Weiteren wurde mittels eines Prototyps untersucht, ob und wie die lokalen Cluster global miteinander verbunden werden können. Dabei lag der Fokus auf die Vernetzung über die Mobilfunknetze.

In einem zweiten Schritt wurde der NGSON Standard untersucht und die grundsätzliche Funktionsweise mit einem Prototyp aufgezeigt. Dafür konnte das bereits bestehende *MobileCloud*-Framework auf Java portiert und als ServiceCloud im NGSON-Node verwendet werden. Umgesetzt wurden zusätzlich zwei der wichtigsten Funktionen von NGSON, das Kontextbewusstsein sowie das Zusammenhängen von Services.

2 Allgemeine Begriffsdefinition

2.1 MobileCloud

MobileCloud (kursiv) ist der Name des Vorprojekts und bezieht sich auf dieses Projekt.

Mit MobileCloud ist die gesamte *Cloud* bzw. das gesamte P2P-Netwerk gemeint. Dieses Netzwerk setzt sich aus verschiedenen mobilen Geräten (Nodes) zusammen, welche durch die MobileCloud Applikation Zugang zum gesamten Netzwerk erhalten.

Die MobileCloud Applikation ist die Applikation, welche auf den mobilen Geräten installiert wird und damit den Zugang zum P2P-Netzwerk aufbaut und verwaltet.

2.2 Node

Ein Node repräsentiert ein mobiles Gerät (Peer) in der MobileCloud. Ein Node hat eine Adresse, mit welcher er über das Netzwerk angesprochen werden kann. Jedes Gerät hat einen Mainnode, welcher das Gerät selbst repräsentiert.

Falls auf einem Gerät mehrere Instanzen von MobileCloud laufen kann es sein, dass dieses Gerät von mehreren Mainnodes repräsentiert wird.

2.3 Nachbarnodes

Die Nachbarnodes sind eine Untermenge aller Nodes im gesamten P2P-Netztwerk. Darin enthalten ist eine maximale Anzahl von Nodes, dessen Adressen der Eigenen am ähnlichsten sind.

2.4 Node-Adresse

Ein wichtiger Teil von MobileCloud sind die Node-Adressen. Diese beinhalten die IP, den Port sowie eine eineindeutige ID des Geräts. In der MobileCloud werden die Geräte nur über die ID angesprochen. Diese wird mit einer HASH-Funktion anhand der Telefonnummer berechnet. Die Adresse verbindet danach die ID mit der IP und dem Port. Mit diesen Informationen ist das Gerät über das Netzwerk ansprechbar.
Die HASH-Funktion wird im *Kapitel 4.3.7.1 Hash-Funktion* genauer beschrieben.

2.5 Leaf-Set

Das Leaf-Set ist die Hauptdatenstruktur eines Nodes. Es ist eine Liste, in welcher die Nachbarnodes in geordneter Reihenfolge abgespeichert werden. Durch dieses Verhalten entsteht die Ringstruktur im Netzwerk.

2.6 NGSON-Node

Ein NGSON-Node hat im Gegensatz zum einfachen Node nicht direkt etwas mit dem Client in der MobileCloud zu tun. Der NGSON-Node ist ein Server, welcher den IEEE 1903 NGSON Standard unterstützt.

Ein solcher Node kann verschiedene Ressourcen unterstützen wie zum Beispiel das Verwalten oder Weiterleiten von Services.

Verwendet ein NGSON-Node die ServiceCloud, repräsentiert er gleichzeitig ein Node im P2P-Netzwerk.

Weitere Informationen über den NGSON Standard sind dem *Kapitel 4.4.1 Einführung* zu entnehmen.

2.7 ServiceCloud

ServiceCloud beschreibt das auf Java portierte *MobileCloud*-Framework. Die Funktionsweise ist identisch, bis auf Kleinigkeiten wie das Loggen oder das JSON[3] Framework.

Die ServiceCloud wird im NGSON-Node als Schnittstelle zum 2P2-Netzwerk verwendet.

2.8 Service

Als Services werden Dienste bezeichnet, welche in irgendeiner Art und Weise angeboten und genutzt werden können. Services können Informationen oder auch erweiterte Funktionalität anbieten.

Als MobileCloud Service werden die Services genannt, welche in der MobileCloud zu Verfügung gestellt werden. Zum Beispiel das Versenden von Textnachrichten.

Android Service ist eine spezielle Android Komponente (oder einen Teil einer Applikation), welche im Hintergrund läuft und anderen Applikationen verschiedene Services zu Verfügung stellt [1].

Webservices sind Services, welche im Internet angeboten werden. Die meisten können über die REST[4] oder standardisierte SOAP[5] Schnittstelle verwendet werden.

[3] Die JavaScript Object Notation (JSON) ist ein kompaktes Datenformat, welches für Mensch und Maschine einfach lesbar ist.
[4] Representational State Transfer (REST) bezeichnet eine Softwarearchitektur für Webanwendungen.
[5] Simple Object Access Protocol (SOAP) ist ein Netzwerkprotokoll, mit welchem Daten zwischen Systemen ausgetauscht werden können.

2.9 Legende

Einige Beispiele werden mit Grafiken beschrieben. In den Abbildungen werden folgende Symbole und Zeichen verwendet:

Symbol	Namen	Beschreibung
	**Mainnode**	Ein Node, welcher das aktuell betrachtete Gerät repräsentiert. Der Node hat die ID xx.
	**Node**	Ein Node, welcher nicht im *Leaf-Set* des *MainNodes* gespeichert ist. Der Node hat die ID zz.
	**Nachbarnode**	Ein Node, welcher im *Leaf-Set* des *MainNodes* gespeichert ist. Der Node hat die ID yy.
	**fehlerhafter Node**	Ein Node, welcher einen inkonsistenten Zustand auslöst, weil er noch in *Leaf-Sets* gespeichert jedoch nicht mehr in der MobileCloud ist oder weil er nicht mit den aktuellen Daten gespeichert wurde.
	**Daten**	Daten, welche versendet werden. Das können Verwaltungsdaten oder auch Bilder, Text, Sprache, usw. sein.
	Struktur	Grundlegende Struktur der zusammenhängenden Nodes.
	**Weg**	Route von Daten.
	Referenz	Gespeicherte Adresse auf einen Node.
	**Fehlerhafte Referenz**	Fehlerhafte gespeicherte Adresse auf einen Node.

Tabelle 1 Symboltabelle

Solange nichts anderes angegeben wird, ist die maximale Anzahlt Nodes pro Leaf-Set-Seite drei.

3 Ausgangslage Studienarbeit

Ein Ansatz für die Serviceanbietung wurde in der Studienarbeit *MobileCloud* aus dem Jahr 2011 [2] untersucht. Das Ziel dieser Arbeit war zu untersuchen, ob und wie mobile Geräte unter Ausnutzung der neuesten Technologien miteinander vernetz werden können. In einem zweiten Schritt sollte dann aufgezeigt werden, wie verschiedene Services in diesen Netzwerken angeboten und verwendet werden können.

Das Resultat der Studienarbeit war das Framework *MobileCloud,* welches als P2P-Netzwerk dessen wichtigsten Funktionalitäten implementiert und als Beispiel ein Service zum Versenden von Textnachrichten enthielt. Das ganze Framework wurde als Android Applikation umgesetzt.

Das Framework unterstützte folgende Funktionalitäten:

- Betreten des Netzwerkes
- Verlassen des Netzwerkes
- Weiterleiten von Nachrichten
- Versenden/Empfangen von Nachrichten

Fehlertoleranz, Sicherheit oder Privatsphäre wurden nicht beachtet, weil der Umfang der Arbeit sonst zu gross gewesen wäre. Auch beschränkte sich das Erstellen von Clustern auf ein lokales Netzwerk.

Der Abschluss der Studienarbeit wurde auf dem Repository mit dem Tag `0.2.1` gekennzeichnet. Alles Anschliessende wurde im Rahmen der Bachelorarbeit entwickelt.

4 Ergebnisse

4.1 Entscheidungen

In diesem Kapitel werden die Entscheidungen, welche bezüglich Technologien oder Algorithmen im Verlaufe des Projektes gefällt worden sind, kurz begründet. Viele dieser Entscheide sind in den jeweiligen Kapiteln noch genauer erläutert.

4.1.1 Keep-Alive

Für die verteilte Fehlerbehandlung wurde untersucht, welche Möglichkeit am geeignetsten ist, um Fehler zu erkennen und zu behandeln.

Verteilte Fehlerbehandlung	
Möglichkeiten:	Standard Keep-Alive, Cooperative Keep-Alive
Entscheid:	Cooperative Keep-Alive
Begründung:	Die Vorteile des Cooperative Keep-Alive werden im *Kapitel 4.3.6.3 Keep-Alive* erläutert.

Tabelle 2 Entscheid Verteilte Fehlerbehandlung

4.1.2 Network Address Translation

Für die Erstellung von globalen Clustern wurde untersucht, welche Möglichkeiten für den Verbindungsaufbau durch NATs bestehen und welche geeignet sind und umgesetzt werden können.

Network Address Translation	
Möglichkeiten:	Hole Punching, ICE, STUN, TURN
Entscheid:	Hole Punching
Begründung:	Die Entscheidung wird im *Kapitel 4.3.5 NAT-Traversierung* erläutert.

Tabelle 3 Entscheid NAT

4.1.3 Webservice

Im Zusammenhang mit NGSON wurde untersucht, welche Möglichkeiten für die Client-Server-Kommunikation bestehen und geeignet sind.

Web service	
Möglichkeiten:	REST, SOAP
Entscheid:	REST
Begründung:	REST ist eine leichtgewichtige Architektur, dessen Verwendung für Benutzer einfach ist. In vielen Programmiersprachen wie zum Beispiel JavaScript, Java oder Ruby ist eine Unterstützung für die Umsetzung zu finden. Alle Mechanismen von REST sind einfach konzipiert und basieren auf dem bereits bestehenden HTTP Protokoll. Dadurch ist es resistent gegen Firewalls, da üblicherweise der verwendete Port 80 offen ist. Im Gegensatz dazu ist SOAP ein sehr umfangreiches und komplizierteres Protokoll, welches nicht auf einfache Art und Weise von verschiedenen Programmiersprachen verwendet werden kann. Die fehlenden Funktionalitäten von REST, wie zum Beispiel Sicherheit oder Session Management, werden vom NGSON übernommen.

Tabelle 4 Entscheid Web service

4.1.4 Rest-Framework

Damit die Server-Implementation mit REST nicht von Grund auf entwickelt werden muss, wurde ein Framework gesucht, welches dies bereits implementiert.

Rest-Framework	
Möglichkeiten:	Jersey, Restlet
Entscheid:	Jersey
Begründung:	Jersey ist die Referenzimplementation von JAX-RS[6] in Java. Jersey wird von einer grossen Community unterstützt, durch welche man schnelle Unterstützung bekommt, sobald Probleme auftreten. Mit Jersey konnte innert kürzester Zeit ein Webservice erstellt und genutzt werden.

Tabelle 5 Entscheid Rest-Framework Server

Beim Client wurde ebenfalls überprüft und entschieden, welche Klassenbibliotheken bei der Umsetzung des REST-Client verwendet werden können.

Rest-Framework	
Möglichkeiten:	Apache HTTP Client, Google HTTP Client , Jersey Client
Entscheid:	Apache HTTP Client
Begründung:	Der Entscheid wird in *Kapitel 4.4.2.6 Client* begründet.

Tabelle 6 Rest-Framework Client

[6] Java API for RESTful Web Services (JAX-RS) ist eine Spezifikation einer Programmierschnittstelle in der Sprache JAVA für REST-Webservices.

4.1.5 Datenbank

Für die Speicherung der Services auf dem NGSON-Node wurde untersucht, welche Datenbankserver verwendet werden könnten.

Datenbank	
Möglichkeiten:	PostgreSQL, MySQL
Entscheid:	PostgreSQL
Begründung:	PostgreSQL wurde bereits in vorherigen Arbeiten verwendet. Durch dessen Verwendung konnte die Einarbeitungszeit in eine neue Technologie auf ein Minimum reduziert werden.

Tabelle 7 Entscheid Datenbank

4.2 Technologiewahl

In den folgenden Kapiteln werden die grundlegenden Technologien beschrieben, welche in dieser Arbeit eingesetzt werden.

4.2.1 Android,Cloud Computing und Peer-to-Peer Topologie

Aufbauend auf der Studienarbeit werden Android, Cloud Computing und Peer-to-Peer weiterhin eingesetzt. Diese drei Technologien sind bereits in der Vorarbeit beschrieben [2].

4.2.2 Service Overlay Network

Ein Overlay Network ist ein Netzwerk, welches auf bestehenden Netzwerken aufbaut. Es bildet dabei eine eigene Topologie und nutzt oft eine eigene von den unteren Netzwerken unabhängige, Adressierung und ein eigenes Routing.

Ein Service Overlay Network ist ein Overlay Network, welches bestimmte Services bietet.

4.2.3 NGSON (Next Generation Service Overlay Networks)

NGSON ist ein Standard des Verbandes IEEE. Diese Arbeit beruht auf dem im Jahr 2011 veröffentlichten IEEE 1903 Dokument. Es spezifiziert eine funktionale Architektur, welche die Möglichkeit bietet kontextbewusste, dynamisch anpassungsfähige und selbstorganisierende Service-Netzwerke zu realisieren.

Der NGSON Standard ist im *Kapitel 4.4.1 Einführung* genauer beschrieben.

4.2.4 REST, RESTful

REST ist ein nicht standardisierter Architekturstil, welcher beschreibt, wie Webservices realisiert werden können. Er wurde von Roy Thomas Fielding in seiner Dissertation *Architectural Styles and the Design of Network-based Software Architectures* [3] beschrieben und wird mittlerweile weit verbreitet eingesetzt.

Grundsätzlich sieht das Modell vor, dass Clients mittels HTTP-Protokoll Ressourcen auf Server zugreifen können. Jede dieser Ressource kann über eine eindeutige URL adressiert werden. Als Ressource-Typen können alle möglichen Internet Media-Typen (image/jpeg, text/html, …) verwenden werden. Das HTTP-Protokoll definiert Methoden, welche auf diesen Ressourcen ausgeführt werden können:

- GET
- PUT
- POST
- DELETE
- HEAD
- etc…

Weitere Merkmale des Modells sind:

- Client-Server Architektur, wobei der Client aktiv und der Server passiv ist.
- Der Server verfolgt zu keiner Zeit den Clientstatus (*stateless*). Bei jeder Client-Anfrage werden immer alle nötigen Daten mitgesendet.

- Clients können Antworten des Servers zwischenspeichern (*caching*), wobei der Server die Antworten als zwischenspeicherbar oder nicht markiert.
- Es können Repräsentationen der Ressourcen als Dokument angefordert werden.

REST kommt bei der Kommunikation zwischen NGSON Nodes und Clients zur Anwendung. Dank dem Einsatz von REST können bereits bestehende Technologien und Webservices genutzt werden. REST unterstützt eine gute Skalierbarkeit. Dies zeigt die Vielzahl an bereits vorhandene REST-Webservices.

Eine Anwendung ist **RESTful**, wenn sie ein REST-Server, also eine zur REST konforme Schnittstelle, oder ein REST-Client implementiert.

4.2.4.1 WADL (Web Application Description Language)

Mit WADL können HTTP-basierte Anwendungen bzw. REST-Webservices beschrieben werden [4]. Die Darstellung ist XML-basiert. Es kann mit dem WSDL (*Web Services Description Language*) für SOAP-basierte Webservices verglichen werden.

Im WADL können die einzelnen Ressourcen beschrieben werden. Die auf diesen Ressourcen aufrufbaren Methoden können mit Typ, Parameter, Eingabedaten, Rückgabedaten und weiteren Informationen genau definiert werden.

Es gibt Tools und Java-Klassenbibliotheken, die WADL generieren können. Zudem lässt sich einfach Client-Code, der die im WADL beschriebenen Ressourcen nutzt, generieren. Ebenfalls möglich ist das dynamische Zusammensetzen eines REST-Aufrufs während der Laufzeit, wie dies in diesem Projekt gemacht wird.

4.2.5 Jersey

Jersey ist eine Referenz-Implementation der Java API für RESTful Webservices (*Java API for RESTful Web Services, JAX-RS*) [5] und steht unter der CDDL 1.1 und GPL 2 Lizenz. Jersey wird von Oracle entwickelt.

Mit Hilfe von Jersey können RESTful Anwendungen in Java entwickelt werden – Server sowie auch Clients. Es gibt eine vollständige Dokumentation und durch die breite Nutzung gibt es viele Codebeispiele und Unterstützung in Foren.

4.2.6 PostgreSQL

PostgreSQL ist ein objektrelationales Datenbankmanagementsystem und wird von einer Open-Source-Community (weiter-)entwickelt. Es ist mit dem SQL-Standard mehrheitlich konform, wobei es auch spezifische Funktionalitäten gibt. Die Software wird unter der PostgreSQL Lizenz, einer Open-Source Lizenz, vertrieben.

PostgreSQL wird auf dem NGSON Node eingesetzt.

4.3 MobileCloud

In diesem Kapitel werden die Erneuerungen und Erweiterungen vom *MobileCloud*-Framework beschrieben. Alle weiteren Informationen zum *MobileCloud*-Framework sind im technischen Bericht des Vorprojekts beschrieben. [2]

4.3.1 Design und Architektur

4.3.1.1 Aufbau des Projekts

MobileCloud besteht aus drei Projekten:

MobileCloud
Ist das Hauptprojekt, sprich die Android Applikation. Dieses beinhaltet den gesamten Code, welcher für das Erstellen von MobileCloud nötig ist, inklusive des Android GUI's.

MobileCloudJavaClient

Der MobileCloudServer wurde in MobileCloudJavaClient unbenannt. Diese Konsolen-Anwendung dient vorwiegend zu Testzwecken als Node im Peer-To-Peer Netzwerk. Des Weiteren kann der Client als Entry Node auf einem beliebigen Rechner mit Java 6 eingesetzt werden.
Dieses Projekt benötigt das MobileCloud-Projekt.

MobileCloudTest
Das Projekt beinhaltet alle Unit Tests für das Projekt MobileCloud. Die Unit Tests werden bei Android immer in einem separaten Projekt erstellt.
Dieses Projekt benötigt das MobileCloud-Projekt.

4.3.1.2 Logische Sicht

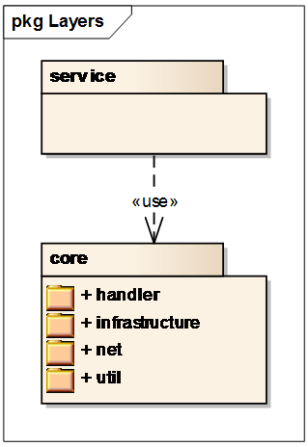


Abbildung 2 Schichten von MobileCloud

Wie bereits im Vorprojekt definiert, gibt es 2 Schichten [2]. Die Verantwortlichkeiten der Schichten wurde leicht abgeändert, damit eine bessere Trennung zwischen den *MobileCloud* Grundfunktionen und den *MobileCloud* Services gemacht werden kann. Zu den Grundfunktionen zählen die gesamte Netzwerkkommunikation sowie die Verwaltung der MobieCloud.

Service

Diese Schicht enthält die Handler, welche die MobileCloud Services umsetzen, und die zur Umsetzung dieser Services benötigten Klassen. Bei einer Erweiterung der MobileCloud Services soll nur diese Schicht angepasst werden müssen.

Wegen einer Fokusänderung bei der Verwaltung und Anbietung der Services, wurde diese Schicht noch nicht implementiert.

Core

Die Core Schicht enthält wie bis anhin die netzwerkrelevanten Informationen. Sie ist für die Datenübertragung verantwortlich. Hinzu kommen die Handler zur Verwaltung der MobileCloud und die dazu benötigten Klassen. Diese Schicht soll möglichst wenigen Änderungen unterlegen sein.

4.3.1.3 Beschreibung der Pakete

Die genauen Beschreibungen der Klassen und Methoden befinden sich im aktualisierten JavaDoc. In diesem Kapitel folgt eine kurze Beschreibung zu den neu hinzugekommenen und gelöschten Klassen.

Paket core.net

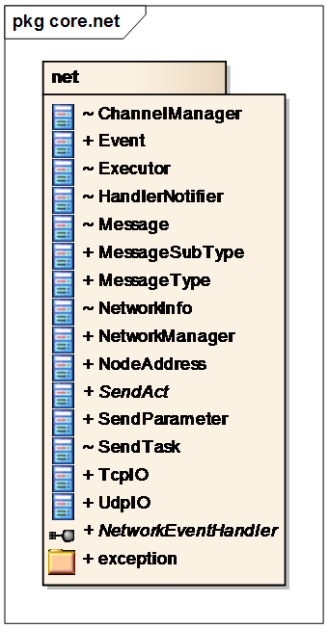


Abbildung 3 Packet core.net

Neu

CHANNELMANAGER

Diese Klasse verwaltet offene TCP-Verbindungen. Sie kontrolliert die Anzahl der offenen Verbindungen und macht eine Zuordnung zwischen Node-Adressen und TCP-Verbindungen.

HANDLERNOTIFIER

Die Klasse *HandlerNotifier* implementiert die Benachrichtigung der entsprechenden Handler bei einem Nachrichteneingang.

SendAct

Diese Klasse dient als Callback, um über den Erfolg bzw. Misserfolg der Schreib- bzw. Sendeoperation informiert zu werden.

SendParameter

Diese Klasse kapselt alle vom Netzwerk benötigten Parameter, um Daten an einen bestimmten Node zu senden.

SendTask

Diese Klasse definiert einen Schreib- bzw. Sendeauftrag für die Ein-/Ausgabe-Klassen (*TcpIO* und *UdpIO*).

TcpIO

Dies ist die Ein-/Ausgabe Klasse für TCP, welche Daten von einem TCP-Port empfangen und senden kann.

UdpIO

Dies ist die Ein-/Ausgabe Klasse für UDP, welche Daten von einem UDP-Port empfangen und senden kann.

Gelöscht

UDPNetworkListener, UDPNetworkSender

Diese Klassen wurden durch die UDP Ein-/Ausgabe Klasse *UdpIO* ersetzt.

Paket core.handler

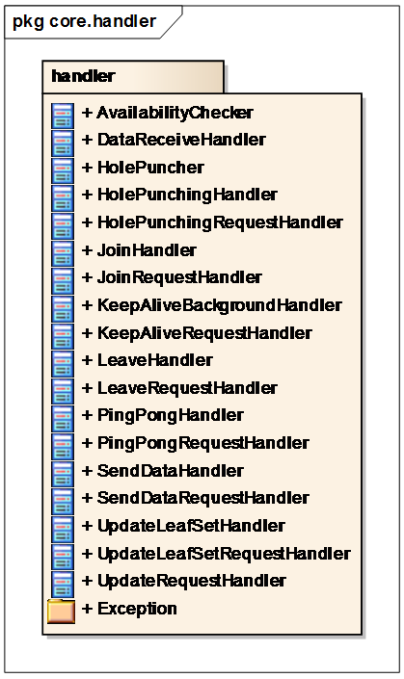


Abbildung 4 Packet core.handler

Neu

AvailabilityChecker, PingPongHandler, PingPongRequestHandler

Mit Hilfe dieser Klassen kann überprüft werden, ob ein bestimmter Node erreichbar ist.

HolePuncher, HolePunchingHandler, HolePunchingRequestHandler

Diese Klassen kümmern sich um das UDP Hole Punching. Genaueres dazu ist im *Kapitel 4.3.5 NAT-Traversierung* zu finden.

KeepAliveBackgroundHandler, KeepAliveRequestHandler

Diese Klassen dienen zur automatischen Erkennung von Ausfällen bzw. unerreichbaren Nodes und leiten entsprechende Massnahmen zur Aktualisierung des Leaf-Sets ein.

Gelöscht

DataReceiveHandler, SendDataHandler, SendDataRequestHandler

Diese Klassen werden nicht mehr benötigt, da sie als Beispiel-Dienst in der Vorarbeit dienten.

EventHandlerCallback, NullEventHandlerCallback, EventHandlerState

Diese Klassen bzw. Interfaces wurden in das Paket *core.util* verschoben.

core.util

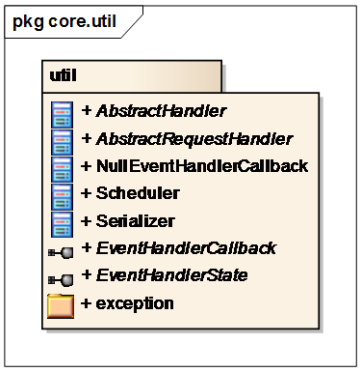


Abbildung 5 Packet core.util

Neu

Scheduler

Diese Klasse ermöglicht das Ausführen von Code zu einem bestimmten Zeitpunkt bzw. in bestimmten regelmässigen Abständen. Dazu muss sich der Code innerhalb der *run*-Methode eines *Runnable*-Objects befinden.

Serializer

Diese Klasse stellt Methoden zur Serialisierung und Deserialisierung von Node-Adressen zur Verfügung.

4.3.2 Multithreading

4.3.2.1 Übersicht

Anhand dieser Übersicht soll aufgezeigt werden, welche Teile der *core*-Schicht vom Mutlithreading betroffen sind. Grundsätzlich gelten dieselben Bedingungen wie im Vorprojekt [2].

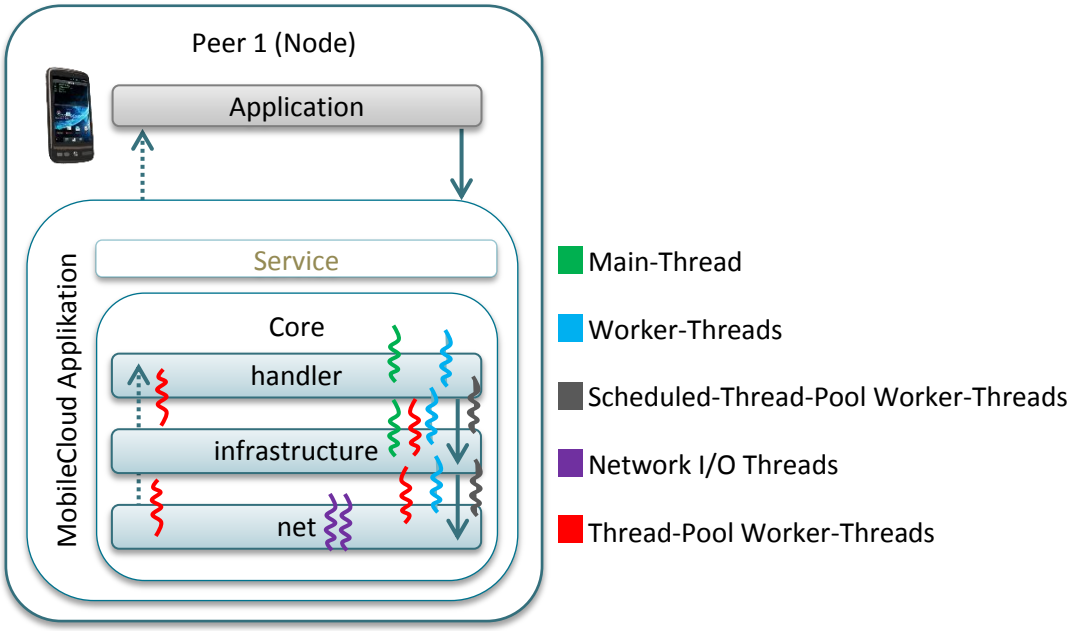


Abbildung 6 Threads innerhalb der MobileCloud Applikation

Beim Multithreading wurden Änderungen beim Netzwerk und den Handlern in der *core*-Schicht vorgenommen. Wie die *Abbildung 6* zeigt, sind zwei neue Thread-Typen hinzugekommen. Diese werden weiter unten im *Kapitel Beschreibung der neuen Thread-Typen* beschrieben. Ferner kann der Main-Thread unter bestimmten Umständen bis in die *core*-Schicht ins *infrastructure*-Paket gelangen, da das *handler*- und das *infrastructure*-Paket in diese Schicht verschoben wurden. Diese Verschiebung musste, wie in *Kapitel 4.3.1.2 Logische Sicht* beschrieben, wegen der Trennung zwischen Service- und Verwaltungshandlern. Der Main-Thread, die Worker-Threads und die Thread-Pool Worker-Threads sind weiterhin vorhanden und im technischen Bericht von MobileCloud genau beschrieben [2].

Beschreibung der neuen Thread-Typen

Scheduled-Pool Worker-Threads

Die Scheduled-Thread-Pool Worker-Threads werden von den Handlern für die Code-Ausführung nach Ablauf einer bestimmten Zeit oder in regelmässigen Zeitabständen benutzt. Dies ermöglicht es, Operationen nach einem Timeout abzubrechen oder wiederholende Operationen nebenläufig ausführen zu lassen.

Diese Threads werden von einem Thread-Pool verwaltet. Sobald ein Handler über die *Scheduler*-Klasse zum ersten Mal eine zeitlich versetzte Operation startet, wird der Thread-Pool angelegt. Beim Beenden des Handlers wird der Thread-Pool automatisch wiederum über die *Scheduler*-Klasse beendet.

Network I/O Threads

Beim Netzwerk kommen neu zwei I/O Threads zum Einsatz. Sie werden beim expliziten Start des Netzwerks automatisch mitgestartet und laufen bis sie durch ein Interrupt beim Stoppen des Netzwerks beendet werden. Diese Threads führen die Schreib- und Leseoperationen auf den Sockets aus. Sie ersetzten den bisherigen Network Listener Thread. Es gibt je einen Thread für die UDP- und einen für die TCP-Kommunikation. So wird sichergestellt, dass die UDP- und TCP-Kommunikation unabhängig voneinander bestehen können.

4.3.2.2 Thread-Sicherheit

Standard-Szenario

Das Standard-Szenario sieht durch die Änderungen bei den Threads nun wie folgt aus:

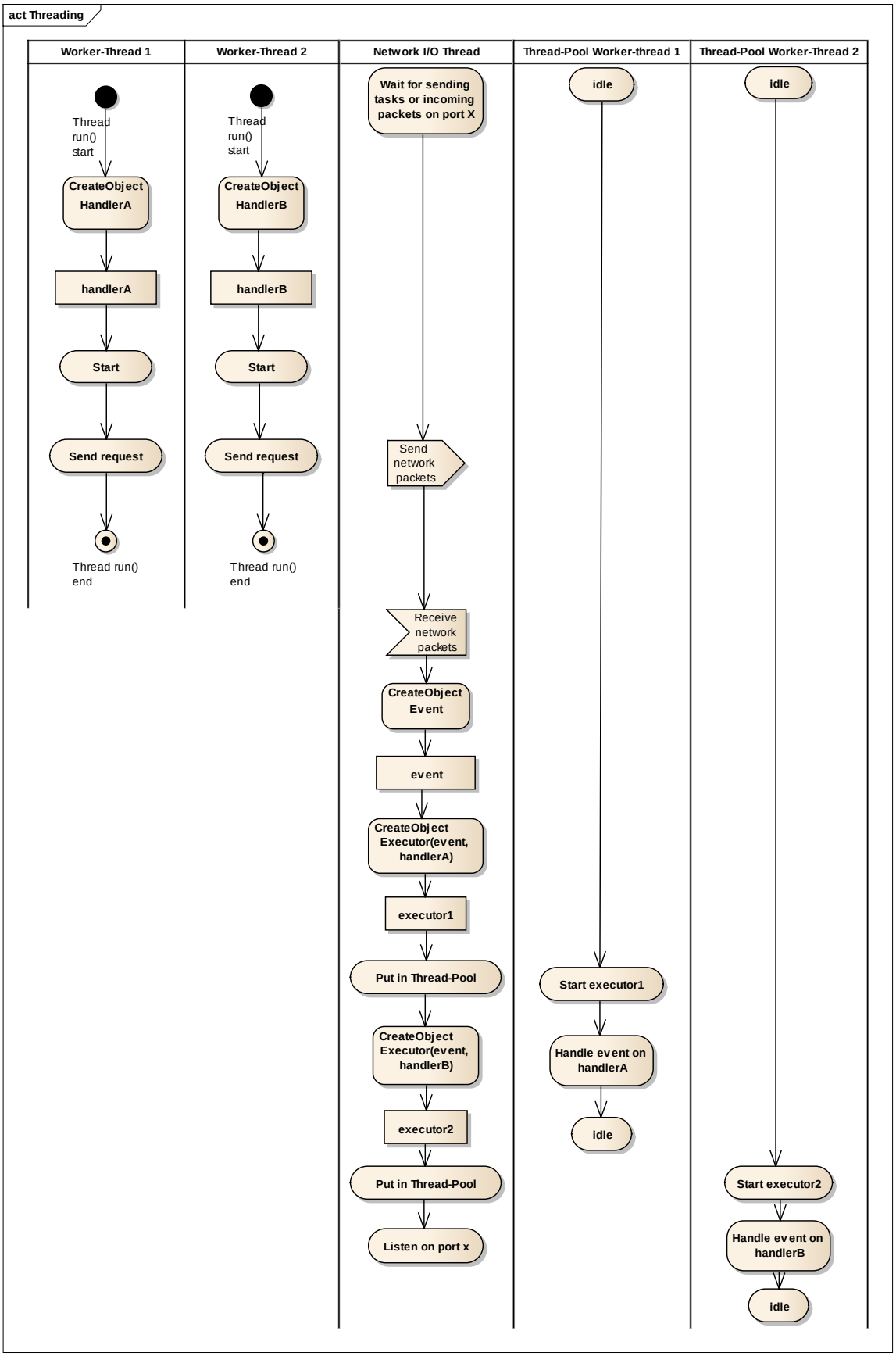


ABBILDUNG 7 THREADING BEISPIEL

Ein grosser Unterschied zu vorher sind die I/O Threads, welche nun auch das Senden der Nachrichten übernehmen. In diesem Beispiel sendet der UDP I/O Thread zuerst die Anfragen und empfängt anschliessend die Antworten.

Von Threads gemeinsam genutzte Objekte

Die vom geänderten Multithreading betroffenen Klassen werden in den nächsten Unterkapiteln beschrieben. Für alle bisherigen Klassen bzw. Objekte gelten weiterhin die Beschreibungen des Vorprojekts [2].

Handler

Eine wichtige und zu beachtende Erneuerung bei den Handlern sind die Scheduled-Thread-Pool Worker-Threads. Nutzt man den Scheduled-Thread-Pool, ist zu berücksichtigen, dass der von diesen Threads ausgeführte Code keine Zustandsänderungen an den Handlerobjekten durchführen darf. Denn diese Objekte könnten gleichzeitig von den Thread-Pool Worker-Threads benutzt und geändert werden. Falls doch am Handlerobjekt-Zustand etwas geändert wird, müssen diese Änderungen synchronisiert werden. Da die Thread-Pool Worker-Threads unter sich mit dem Monitor-Objekt der Handler synchronisiert werden, kann eine Synchronisation mit den Scheduled-Thread-Pool Worker Threads ebenfalls über das Monitor-Objekt erfolgen.

TcpIO und UdpIO

Objekte dieser beiden Klassen werden von den Worker-Threads, Thread-Pool Worker-Threads und den Network I/O Threads benutzt. Die Worker-Threads und Thread-Pool Worker-Threads hinterlegen Sendeaufträge in eine durch Locks gesicherte Queue. Die Network I/O Threads arbeiten diese Aufträge nacheinander ab. Folgende Abbildung soll dies veranschaulichen:

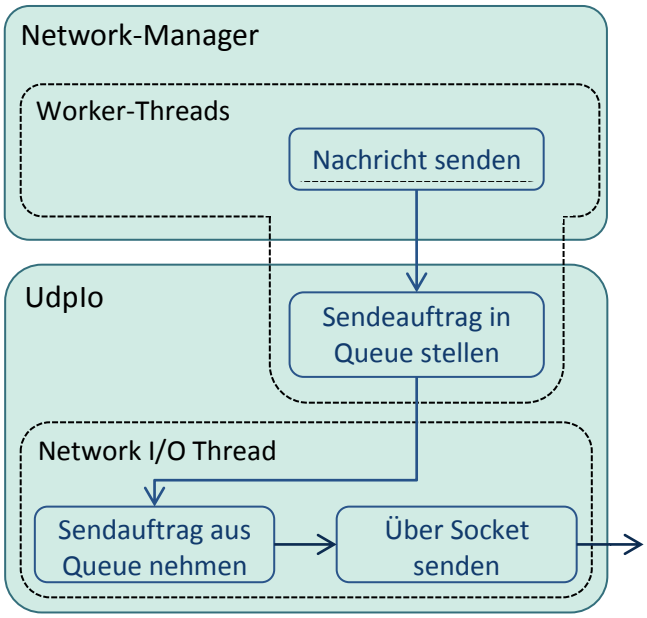


Abbildung 8 Nachricht senden

4.3.3 Netzwerk

4.3.3.1 TCP

Das Netzwerk wurde mit der Möglichkeit, Daten über TCP zu versenden, erweitert. Dazu musste unteranderem das SPMP angepasst werden. Diese Änderung wird im *Kapitel 4.3.4 SPMP* genauer beschrieben. Der *NetworkManager* bietet nun eine *sendOverTcp*-Methode an, mit Hilfe welcher Daten über TCP versendet werden können.

4.3.3.2 Erfolgsbenachrichtigung

Damit, wenn nötig, die oberen Schichten über das erfolgreiche oder misslungene Senden einer Nachricht informiert werden, kann neu ein Callback zu diesem Zweck hinterlegt werden. Dieser wird beim Aufruf einer Sende-Methoden (TCP oder UDP) beim *NetworkManager* mitgegeben.

4.3.3.3 Fragmentierung

Nachrichten, welche zu gross zum Versenden sind, werden bei UDP in mehrere Netzwerkpakete unterteilt und gesendet. Anschliessend werden diese auf Empfängerseite wieder zu einer Nachricht zusammengefügt. Ansonsten würden die UDP-Pakete auf Netzwerkebene fragmentiert und höchstwahrscheinlich unterwegs verworfen werden. Dazu wurde, wie im *Kapitel 4.3.4 SPMP* beschreiben, das eigene Protokoll erweitert.

4.3.4 SPMP

SPMP (*Simple Peer-to-Peer Management Protocol*) wurde in der Studienarbeit *MobileCloud* entwickelt [2] und in dieser Arbeit erweitert. Um bestimmte Neuerungen wie Fragmentierung und TCP zu unterstützen, musste der Header-Aufbau des SPMP angepasst werden.

4.3.4.1 Protokoll-Stack

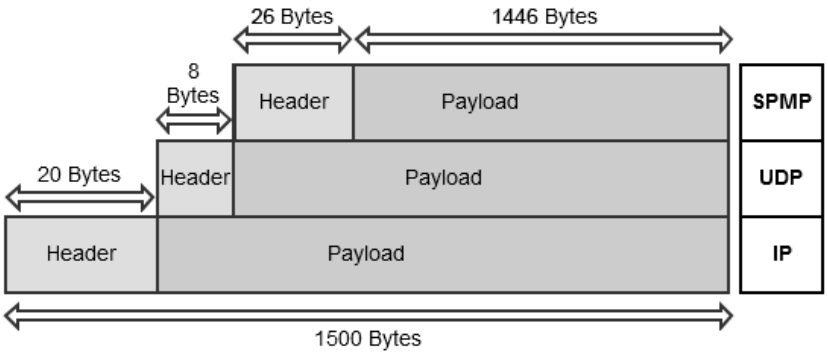


Tabelle 8 Protokoll-Stack bei UDP

Der Header wurde um 9 Bytes länger. Somit steht bei UDP und 1500 Bytes MTU[7] noch 1446 Bytes für die Nutzdaten des SPMP zur Verfügung. Bei TCP sind es 1434 Bytes. Der Header-Aufbau ist im *Kapitel 4.3.4.2 Nachrichtenaufbau* zu finden.

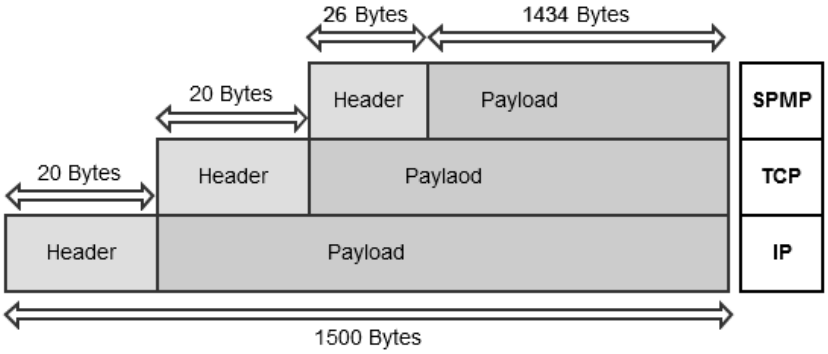


Tabelle 9 Protokoll-Stack bei TCP

4.3.4.2 Nachrichtenaufbau

Bits								
8	7	6	5	4	3	2	1	
Message sub-type				Version				Octet 1
Source address								Octet 2-8
Destination address								Octet 9-15
Message type								Octet 16+17

[7] Die Maximum Transmission Unit (MTU) beschreibt die maximale Paketgröße eines Protokolls.

Message ID	Octet 18-21
Payload length	Octet 22-25
Packet number	Octet 26
Payload	Octet 27 -1445

Folgende Änderungen wurden gemacht:

Source-, Destination Address

Die Absender- und Empfängeradressen wurden von fünf auf sieben Bytes verlängert. Dies musste wegen der Hash-Generierung erweitert werden. Nur so können alle möglichen Hash-Werte mit den zur Verfügung stehenden Bytes dargestellt werden.

Payload length

Dieses Feld ist neu und gibt die Länge der Nutzdaten an. Es wird für die Fragmentierung bzw. Defragmentierung benötigt. Es sind somit bis zu 2^{32} Bytes Nutzdaten pro Nachricht möglich.

Packet number

Dieses Feld ist neu und gibt die Paket-Nummer an. Es wird für das Zusammensetzen einer fragmentierten Nachricht benutzt. Es sind bis zu 2^{8} Pakete möglich.

Die anderen Felder sind gleich geblieben [2].

4.3.4.3 Festgelegte Mitteilungstypen

Bei den Mitteilungstypen sind vier neue dazugekommen (1-4).

Wert	Kürzel	Beschreibung
0	TEST	Wird für Testzwecke während der Entwicklung verwendet.
1	NETWORK	Wird für Nachrichten auf Netzwerk-Ebene verwendet.
2	NAT_TRAVERSAL	Wird für NAT-Traversierungsnachrichten verwendet. Siehe *Kapitel 4.3.5 NAT-Traversierung* für weitere Details.
3	KEEP_ALIVE	Wird für Keep-Alive-Signale verwendet.
4	PING_PONG	Wird für das Pingen von Nodes verwendet.
5	JOIN	Wird für die Abhandlung eines Node-Beitritts zur MobileCloud verwendet.
6	LEAVE	Wird beim Verlassen eines Nodes verwendet.
7	UPDATE	Ein neuer Node ist vorhanden, oder hat seine Adresse geändert und kann in die Adressliste aufgenommen werden.
8	NODELIST_REQUEST	Ein Adressbereich wird angefordert.

Tabelle 10 Festgelegte Mitteilungstypen

Die Mitteilungssubtypen haben sich nicht geändert. Siehe Vorprojekt [2].

4.3.4.4 Fehler bei UDP

Wie in der Vorgängerarbeit erwähnt, kann bei UDP folgendes Problem auftreten [2]:

- Ein Node wartet immerzu auf eine Antwort, wenn die Anfrage- oder Antwort-Nachricht verloren geht.

Mit Hilfe von Timeouts wird dieses Problem verhindert. Jeder Handler kann selber festlegen, wann eine Netzwerk-Operation unterbrochen werden muss. Dies hängt stark mit dem Typ der Operation zusammen. Die Timeout-Zeit kann zum Beispiel davon abhängen, ob der Benutzer warten muss oder die Nachricht über viele Nodes weitergleitet wird. Das Definieren von optimalen Timeouts mittels Simulationen und Tests waren jedoch nicht Bestandteil dieser Arbeit.

4.3.5 NAT-Traversierung

4.3.5.1 Begriffe

Adresse

Als Adresse wird die Kombination aus der IP und dem Port bezeichnet.

NAT

Network Address Translation ist ein bei Routern eingesetztes Verfahren, um einem gesamten Netzwerk eine IP zuweisen zu können. Dies ermöglicht die Wiederverwendung von IPv4-Adressen in unterschiedlichen Netzwerken.

Private Adresse

Jeder Rechner innerhalb einer NAT erhält eine eigene IP-Adresse. Diese ist jedoch nur im aktuellen lokalen/privaten Netzwerk gültig. Alle Adressen (IP und Port) mit dieser privaten IP sind ebenso privat.

Öffentliche Adresse

Dies ist die Adresse, über welche ein in einem NAT befindlicher Rechner erreichbar ist. Sie besteht aus der IP des NAT-Routers und ein vom Router gewählter Port. Der Router führt eine Zuordnungstabelle zwischen gewählten Ports und privaten Adressen. So kann er die eingehenden IP-Pakete an die entsprechende private Adresse weiterleiten.

Traversierung

Wenn ein NAT-Router eingehende Pakete nicht an private Adressen weiterleitet, wird oft ein Verfahren verwendet, der den Router dazu zwingt, die Pakete trotzdem weiterzuleiten. Dies wird als Traversierung bezeichnet.

4.3.5.2 Problembeschreibung

Wie in der Vorgängerarbeit beschrieben, sind NATs[8] ein Problem beim Verbindungsaufbau zwischen zwei Nodes. Befinden sich die Nodes in unterschiedlichen privaten Netzwerken, können für den Datenaustausch nicht die privaten Adressen verwendet werden. In diesem Fall sind die Nodes über ihre öffentlichen Adressen, welche von den NATs auf die privaten Adressen abgebildet

[8] NAT = Network Address Translation

werden, erreichbar. Dies gilt jedoch nicht für alle Fälle. Je nach NAT-Typ werden die eingehenden IP-Pakete blockiert. Es gibt folglich zwei Probleme zu lösen. Erstens muss die korrekte öffentlich Adresse, über welche der gewünschte Node erreichbar ist, herausgefunden werden. Zweitens muss bewerkstelligt werden, dass die NATs die eingehenden IP-Pakete nicht blockieren.

Um diesem Problem entgegenzuwirken, wurden in der Vorgängerarbeit verschiedene Ansätze bzw. Verfahren erwähnt. Dies sind unteranderem:

- STUN [6]
- TURN [7]
- Hole Punching [8]
- ICE [9]

Das Ziel in dieser Arbeit ist es, eine direkte Verbindung zwischen zwei mobile Geräte herzustellen. Auch im Fall, dass sich die Geräte in unterschiedliche Mobilfunknetze befinden.

4.3.5.3 Entscheid für ein Traversierungsverfahren

STUN

STUN (*Session Traversal Utilities for NAT*) ist ein RFC und beschreibt ein Protokoll, welches von Rechnern in privaten Netzwerken dazu verwendet werden kann, Informationen über eigene Adressumsetzung und NATs zu erhalten. Dabei wird ein öffentlicher Server, sogenannter STUN Server, als Hilfsmittel zur Erkennung der NAT-Typen verwendet. Dieser Standard definiert nicht direkt ein Verfahren für die NAT-Traversierung, sondern kann als Hilfsmittel dafür eigensetzt werden.

TURN

Bei TURN (*Traversal Using Relays around NAT*) verläuft die Kommunikation zwischen zwei Rechnern über einen öffentlichen Vermittlungsserver, ein sogenannter TURN Server. Alle Daten werden über diesen Server von und zu den Rechnern weitergeleitet.

Hole Punching

Hole Punching ist ein einfaches und effektives Verfahren. Man unterscheidet zwischen UDP, TCP und ICMP Hole Punching. Bei UDP Hole Punching liegt die Erfolgsquote bei rund 80% [10]. Dies gilt jedoch nur beim Einsatz mit gewöhnlichen NATs, wie sie in privaten Haushalten vorkommen. Wie der Name bereits sagt, geht es darum „Löcher“ in die NATs zu stanzen. Sendet ein hinter einer NAT befindlicher Rechner A eine Anfrage an einen öffentlichen Server, muss die Antwort des Servers von der NAT zum Rechner A weitergeleitet werden. Dies nutzt man bei diesem Ansatz aus. Weitere Informationen dazu befinden sich im *Kapitel 4.3.5.4 Hole Punching*.

ICE

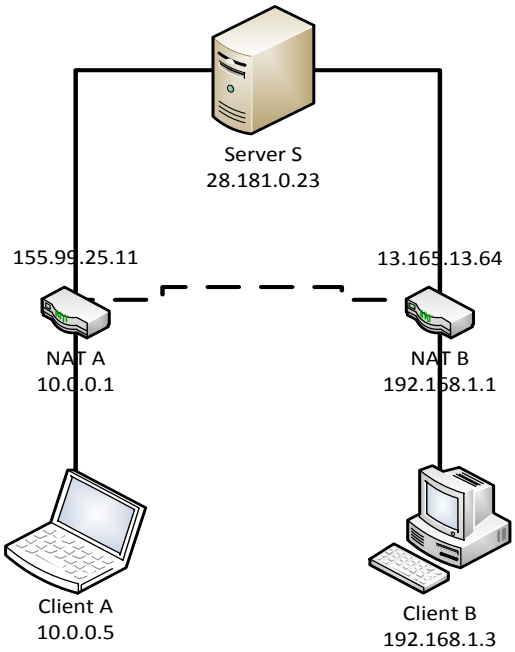
ICE (*Interactive Connectivity Establishment*) ist ein RFC und umfasst mehrere Protokolle bzw. Ansätze wie STUN, TURN und Hole Punching.

Schlussfolgerung

Da das Ziel eine direkte Verbindung zwischen zwei mobile Geräte ist, wurde TURN ausgeschlossen. STUN ist ein umfangreiches Protokoll und liefert vorwiegend Netzwerkinformationen. Es beschreibt keine Traversierung. Daher wurde STUN ebenfalls vorerst ausgeschlossen. Es wurde entschieden den Hole Punching Ansatz zu testen. Da die gesamte Kommunikation zur Verwaltung der MobileCloud über UDP basiert, wurde UDP Hole Punching genauer untersucht.

4.3.5.4 Hole Punching

Das Grundprinzip von Hole Punching lässt sich wie folgt beschreiben:



1. A sendet eine Anfrage an den Vermittlungsserver S.
2. S sendet A die öffentliche Adresse von B. Des Weiteren leitet S die Anfrage inklusive der öffentlichen Adresse von A an B weiter. Nun kennen A und B gegenseitig ihre Adressen.
3. A und B senden sich gegenseitig IP-Pakete. Dabei warten sie auf ein IP-Paket des Gegenübers.
4. Die NAT von A registriert die ausgehenden Pakete von A an B. Ab dem ersten ausgehenden Paket leitet die NAT im Idealfall die eingehenden Nachrichten von B an A weiter. Dasselbe geschieht bei NAT von B. Ab dem ersten ausgehenden Paket von B an A, leitet NAT B die Pakete von A an B weiter. Im Idealfall besteht nun eine Verbindung zwischen A und B.

Dieser Ansatz kann mit verschiedenen IP-basierten Protokollen umgesetzt werden. Wobei es am einfachsten mit zustandslosen Protokollen wie UDP und ICMP funktioniert.

Voraussetzungen:

- Die beteiligten privaten Rechner müssen vor dem Starten dem Server bekannt sein. Das heisst, der Server muss die privaten sowie auch die öffentlichen Adressen der beteiligten Rechner kennen.
- Der Vermittlungsserver kann jederzeit Daten an die „registrierten“ Rechner senden.
- Die beteiligten NATs sind vom Typ *Full Cone*, *Restricted Cone* oder *Port Restricted Cone*. [2]

Versuche

Um UDP Hole Punching auf Mobilfunknetzwerken testen zu können, wurden ein Android Client und ein Java Vermittlungsserver entwickelt. Als Grundlage wurde die bereits vorhandenen Netzwerkkomponenten vom *MobileCloud* Projekt benutzt. Dies aus dem Grund, dass bei einer Implementation bei *MobileCloud* ebenfalls diese Komponenten verwendet werden würde.

Beschreibung

Die beiden Clients melden sich beim Vermittlungsserver an. Dieser speichert deren öffentliche Adressen. Client A sendet eine Verbindungsanfrage an den Server. Dieser leitet die Anfrage an Client B inklusive öffentliche Adresse von A weiter. Zudem sendet er die öffentliche Adresse von B an A. Sobald Client B die Anfrage erhalten hat, sendet dieser im Abstand von einer Sekunde 10 UDP Pakete an die öffentliche Adresse von A. Sobald Client A vom Server die Antwort mit der öffentlichen Adresse von B erhalten hat, sendet dieser UDP Pakete an B. Ein Verbindungsaufbau ist dann erfolgreich, wenn innerhalb von 10 Sekunden beide Clients mindestens ein UDP Paket des Gegenübers empfangen haben.

Resultat der Tests

Netz A	Netz B	Resultat
Lokal ohne NAT dazwischen		OK
Gewöhnliches Heimanwender-NAT	Swisscom	OK
Swisscom	Sunrise	Nicht OK Es kann keine Verbindung hergestellt werden. Kein einziges UDP-Paket wird auf einer der Seiten von der anderen Seite empfangen.
Swisscom	Orange	Nicht OK Es kann keine Verbindung hergestellt werden. Kein einziges UDP-Paket wird auf einer der Seiten von der anderen Seite empfangen.
Sunrise	Orange	Nicht OK Es kann keine Verbindung hergestellt werden. Kein einziges UDP-Paket wird auf einer der Seiten von der anderen Seite empfangen.

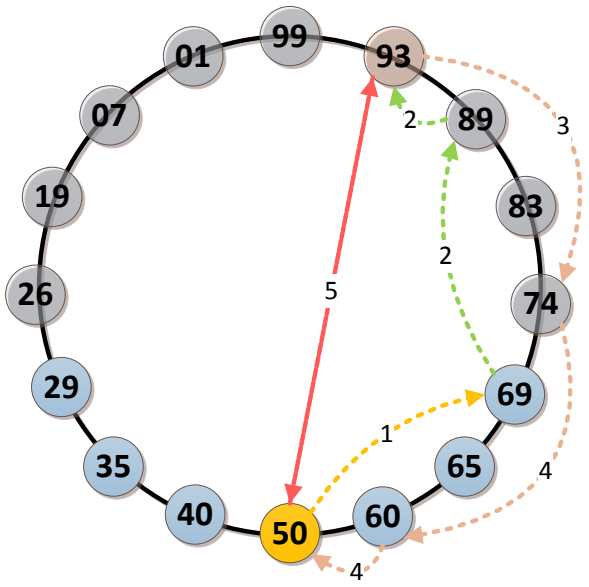
Tabelle 11Testergebins zur NAT-Traversierung

Die ersten zwei Tests wurden zum Testen des Algorithmus durchgeführt. Sie zeigen, dass jener funktionieren würde, jedoch wegen den verwendeten NATs bei den Mobilfunkbetreibern scheitert.

Umsetzung

Die Umsetzung soll ohne zentralen Vermittlungsserver auskommen. Jeder Node soll die Rolle des Vermittlungsserver übernehmen können. Die Voraussetzung, damit dies funktioniert ist, dass der Vermittlungsnode eine Verbindung (direkt oder indirekt) zu beiden beteiligten Parteien hat. Mit dieser Anforderung gibt es folgende Umsetzungsmöglichkeiten:

Variante 1 – Anfrage durch den Ring routen



Node 50 möchte eine direkte Verbindung mit Node 93. Das Leaf-Set hat eine Grösse von 3 Nodes pro Seite.

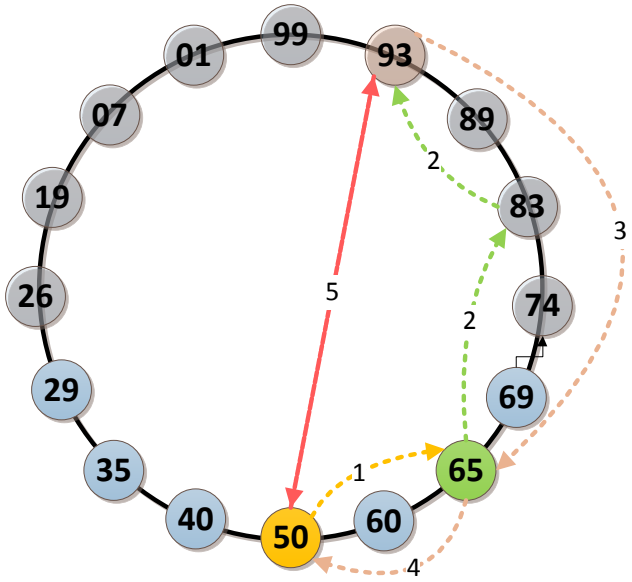
1. Node 50 sendet seinem äussersten Nachbarn (69) die Hole Punching Anfrage. Node 69 fügt die öffentliche Adresse von 50 der Anfrage hinzu und leitet diese weiter.
2. Die Anfrage wird im Ring bis zum Ziel, hier 93, weitergeleitet.
3. Node 93 erhält die Anfrage und startet das Hole Punching mit der öffentlichen Adresse des Nodes 50. Gleichzeitig sendet er seine Antwort an seinen äussersten Nachbarn 74. Dieser fügt die öffentliche Adresse von 93 der Antwort hinzu und leitet diese weiter.
4. Die Antwort wird bis zum Ziel, hier 50, weitergeleitet.
5. Node 50 entnimmt der Antwort die öffentliche Adresse des Nodes 93 und startet ebenfalls mit dem Hole Punching.

Voraussetzungen:

- Damit Node 69 die öffentliche Adresse von 50 kennt, darf sich dieser nicht hinter derselben NAT wie 50 befinden.
- Damit Node 74 die öffentliche Adresse von 93 kennt, darf sich dieser nicht hinter derselben NAT wie 93 befinden.
- Node 93 muss mindestens solange Pakete senden bis Node 50 die Antwort erhalten und sein erstes Paket an 93 gesendet hat. Dies müsste mittels Timeout oder einer Bestätigungsnachricht von 50 an 93 über den Ring koordiniert werden.
- Jeder Node muss stets seine im Leaf-Set gespeicherten Nachbarn erreichen können. Das heisst, bei jeder Änderung des Leaf-Sets muss überprüft werden, ob die neuen Nachbarn erreichbar

sind. Wenn nicht, muss Hole Punching angewendet werden. Hat man noch keine Verbindung zu irgendeinem Nachbar, kann der Public Node als Vermittler eingesetzt werden.

Variante 2 – Public Node als Vermittlungsserver



1. Node 50 sendet dem Public Node 65 die Hole Punching Anfrage. Der Public Node, hier 65, fügt der Anfrage die öffentliche Adresse von 50 hinzu und leitet diese weiter. Der Public Node könnte auch die Anfrage direkt an 93 senden, doch kann man nicht davon ausgehen, dass die NAT des Nodes 93 eingehende Nachrichten vom Public Node zulässt.
2. Die Anfrage wird im Ring bis zum Ziel, hier 93, weitergeleitet.
3. Node 93 erhält die Anfrage und startet das Hole Punching mit der öffentlichen Adresse des Nodes 50. Gleichzeitig sendet er seine Antwort direkt dem Public Node.
4. Der Public Node fügt der Antwort die öffentliche Adresse von 93 an und leitet diese dem Node 50 direkt weiter.
5. Node 50 entnimmt der Antwort die öffentliche Adresse des Nodes 93 und startet ebenfalls mit dem Hole Punching.

Voraussetzungen:

- Node 93 muss mindestens solange Pakete senden bis Node 50 die Antwort erhalten und sein erstes Paket an 93 gesendet hat. Dies müsste mittels Timeout oder einer Bestätigungsnachricht von 50 an 93 über den Ring koordiniert werden.

Entscheid

In MobileCloud wurde Variante 2 implementiert, weil diese stabiler ist. Befinden sich alle Nachbarn eines Nodes A hinter der gleichen NAT, funktioniert Variante 1 nicht. Damit ein externer Node B eine Verbindung zu A aufbauen kann, muss einer der Nachbarn von A dessen öffentliche Adresse an B weiterleiten. Die Nachbarn kennen jedoch nur die lokale Adresse. Bei Variante 2 vermittelt nicht einer der Nachbarn sondern immer der Public Node die öffentliche Adresse. Dieser kennt sie stets und kann sie an B weiterleiten.

4.3.6 Fehlerbehandlung

4.3.6.1 Grundlagen Verteilte Fehlerbehandlung

Die Fehlerbehandlung in verteilten Systemen ist nicht ganz einfach. Möchte sichergestellt werden, dass das System komplett fehlerfrei funktioniert, müsste jeder denkbare Fehler gefunden und behandelt werden. Gerade bei einem verteilten System, welches von vielen äusseren Faktoren beeinflusst wird, ist dies kaum möglich.

Damit die Fehlerbehandlung eingegrenzt werden kann, wird in *MobileCloud* auf das CAP-Theorem von Seth Gilbert und Nancy Lynch zurückgegriffen [11].

Im CAP-Theorem werden verteilte Systeme in drei Eigenschaften unterteilt:

• **Konsistenz** (*Consistency*): Alle sehen zur selben Zeit dieselben Daten.

• **Verfügbarkeit**(*Availability*): Alle Anfragen werden vom System in minimaler Zeit beantwortet.

• **Partitionstoleranz**(*Partition tolerance*): Das System funktioniert auch, wenn das Netzwerk partitioniert ist, einzelne Partitionen nicht mehr erreichbar sind oder Nachrichten verloren gehen.

In diesem Theorem wurde aufgezeigt, dass es in einem verteilten System nicht möglich ist, alle drei Eigenschaften gleichzeitig zu garantieren. Wird zum Beispiel ein System auf mehrere Partitionen aufgeteilt und sollte gleichzeitig die Konsistenz sichergestellt werden, sinkt die Antwortzeit des Systems. Die Antwortzeit vergrössert sich mit der Anzahl von Partitionen des Systems, weil zuerst alle Partitionen synchronisiert werden müssen.

Für die MobileCloud bedeutet dies, dass durch den hohen Grad der Partitionierung sowie der Anforderung an eine hohe Verfügbarkeit die Konsistenz nicht garantiert werden kann. Somit müssen alle Daten, welche in der MobileCloud gespeichert sind als inkonsistent betrachtet werden.

Durch die oben erwähnte, nicht sichergestellte Konsistenz, muss auch die Ringstruktur von MobileCloud immer als inkonsistent angesehen werden. Das kann bedeuten, dass Nodes, welche im Leaf-Set referenziert werden nicht aktuell sind.

Durch den Keep-Alive Algorithmus werden die Leaf-Sets aller Nodes in regelmässigen Zeitabständen auf Konsistenz überprüft. Werden Unregelmässigkeiten entdeckt, wird bei allen inkonsistenten Leaf-Sets eine Aktualisierung durchgeführt. Somit wird die Konsistenz nach einer gewissen Zeit wiederhegestellt. Der genaue Vorgang von Keep-Alive ist im *Kapitel 4.3.6.3 Keep-Alive* beschrieben.

4.3.6.2 Fehlerbehandlung in MobileCloud

Ein stabiles und benutzerfreundliches P2P-Netzwerk muss eine hohe Fehlertoleranz aufweisen. Es wird öfters vorkommen, dass sich Nodes nicht wie angenommen verhalten oder Abläufe durch äussere Einflüsse gestört werden. Zusätzlich muss beachtet werden, dass Fehler eines einzelnen Nodes oft auch Auswirkungen auf andere Nodes haben.

Besser als Fehler zu beheben ist es, dessen Eintreten zu verhindern. Es wird aber immer Fälle geben, in welchen dies nicht möglich ist. Dann muss der Fehler gefunden und anschliessend behandelt oder zumindest abgeschwächt werden. Um die Auswirkungen eines Fehlers möglichst

gering zu halten, muss dieser schnellst möglichst nach dem Auftreten entdeckt und behandelt werden.

In MobileCloud gibt es drei grundlegende Fehlerquellen: Algorithmen, Datenstrukturen und dieInfrastruktur. Diese werden in diesem Kapitel genauer betrachtet und in einige Fehlerfälle zerlegt. Die einzelnen Fehler werden dann genauer betrachtet und Möglichkeiten zur Erkennung, Verhinderung, Minderung und/oder Behandlung betrachtet.

Algorithmen

Die Algorithmen sind ein zentraler Punkt in den P2P-Netzwerken. Sie definieren die einzelnen Abläufe, wie zum Beispiel das Betreten des Netzwerks oder das Versenden von Daten. Diese Algorithmen können in zwei Gruppen unterteilt werden: in zustandsbehaftete oder zustandslose.

Der genaue Ablauf der Algorithmen ist im *Kapitel 4.3.7 Algorithmen* beschrieben.

Die **zustandslosen** Algorithmen bestehen aus zwei Operationen, einer Anfrage und einer Antwort. Bei diesen Operationen kann es vorkommen, dass eine Anfrage oder Antwort verloren geht. Da während der ganzen Operation kein Zustandswechsel stattfindet, kann beim Ausbleiben einer Antwort einfach nochmals eine Anfrage gesendet werden.

Fehler	A1 - Keine Antwort
Beschreibung	Nach dem Versenden einer Anfrage bleibt die erwartete Antwort aus.
Feststellen	Nach dem Versenden einer Anfrage trifft in einer festgelegten Zeit keine Antwort ein.
Vermindern	Nochmaliges Senden der Anfrage. Maximal drei Mal.
Behandlung	Den Empfänger als nicht erreichbar einstufen.

Tabelle 12 A1 - Keine Antwort

Die **zustandsbehafteten** Algorithmen sind etwas komplizierter. Bei diesen muss sichergestellt werden, dass sich das System nach Abschluss der Operation in einem konsistenten Zustand befindet, unabhängig davon, ob die Operation erfolgreich oder nichterfolgreich war. Tritt beim Ablauf eines Algorithmus ein Fehler auf, gibt es zwei Möglichkeiten: Die Operation kann trotzdem abgeschlossen und das System in einen neuen Zustand überführt werden oder die Operation wird rückgängig gemacht und das System in den ursprünglichen Zustand zurückgesetzt.

Im Folgenden werden die zustandsbehafteten Algorithmen genauer angeschaut:

Betreten einer Cloud (join)

Fehler	A2 - Kein Entrynode
Beschreibung	Es ist kein bekannter aktiver Node für das Eintreten in das Netzwerk verfügbar.
Feststellen	Kein der bekannten Nodes ist beim Eintreten erreichbar. Dies kann durch den Ping-Pong Algorithmus überprüft werden.
Vermindern	Abspeichern von mehreren bekannten Nodes. So kann die Wahrscheinlichkeit reduziert werden, dass keiner erreichbar ist. Zusätzlich die gespeicherten Adressen jedes Mal beim Verlassen aktualisieren.
Behandlung	Annahme, dass kein aktiver Node vorhanden ist. Deshalb ein neues, eigenes P2P-Netzwerk erstellen, bei welchem sich wiederum andere Nodes anmelden können.

Tabelle 13 A2 - Kein Entrynode

Fehler	A3 - Login nicht atomar
Beschreibung	Zwei Nodes N1 und N2 melden sich zur selben Zeit beim Netzwerk an. Die Nodes müssten so platziert werden, dass sie sich gegenseitig im Leaf-Set haben. Fordert N2 das Leaf-Set von N1 an, jedoch bevor N1 sich bei allen anderen Nodes angemeldet hat, würden schlussendlich N1 und N2 sich nicht kennen.
Feststellen / Behandlung	Die Feststellung sowie die Behandlung dieses Fehlers, übernimmt der Keep-Alive Algorithmus (*Kapitel 4.3.6.3 Keep-Alive*).

TABELLE 14 A3 - LOGIN NICHT ATOMAR

Verlassen einer Cloud (leave)

Fehler	A4 – verlorene Logout Nachricht
Beschreibung	Durch das Fehlen einer Logout-Nachricht ist die Node-Adresse des nicht mehr vorhandenen Nodes noch in den Leaf-Sets dessen ehemaliger Nachbarn vorhanden.
Feststellen / Behandlung	Die Feststellung sowie die Behandlung dieses Fehlers übernimmt der Keep-Alive Algorithmus (*Kapitel 4.3.6.3 Keep-Alive*).

TABELLE 15 A4 – VERLORENES LOGOUT

Daten Senden (sendData)

Fehler	A5 – Node ist nicht verfügbar
Beschreibung	Eine Anfrage zum Senden von Daten wird durch das Netzwerk geleitet, der Empfänger ist allerdings nicht im Netzwerk.
Feststellen	Die Nachbarnodes des potentiellen Empfängers können in ihrem Leaf-Set feststellen, ob der Empfänger erreichbar ist oder nicht. Ist die Empfänger Adresse nicht im Leaf-Set vorhanden, dann ist der Empfänger nicht erreichbar.
Behandlung	Es wird eine Nachricht an den Absender zurückgesendet mit der Meldung, dass der Empfänger nicht erreichbar ist.

TABELLE 16 A5 – NODE IST NICHT VERFÜGBAR

DATENSTRUKTUR

Die Datenstruktur wird verwendet, um Adressen von anderen Nodes zu speichern und dadurch eine geordnete Netzwerkstruktur zu erhalten. In MobileCloud ist es das Leaf-Set, welches die Adressen der Nachbarnodes speichert. Durch das stetige Hinzufügen und Entfernen von Nodes, kann es passieren, dass das Leaf-Set in einen inkonsistenten Zustand gerät.

Im Folgenden werden die möglichen inkonsistenten Zustände genauer angeschaut:

Fehler	A6 – Node existiert nicht mehr
Beschreibung	Eine Node-Adresse ist im Leaf-Set eingetragen, jedoch im Netzwerk nicht oder nicht mehr vorhanden.
Feststellen / Behandlung	*Die Feststellung sowie die Behandlung dieses Fehlers übernimmt der Keep-Alive Algorithmus (Kapitel 4.3.6.3 Keep-Alive).*

TABELLE 17 A6 – NODE EXISTIERT NICHT MEHR

Fehler	A7 – Node ist nicht eingetragen
Beschreibung	Eine Node-Adresse im Leaf-Set ist im Netzwerk vorhanden, jedoch nicht im Leaf-Set eingetragen.
Feststellen / Behandlung	Die Feststellung sowie die Behandlung dieses Fehlers übernimmt der Keep-Alive Algorithmus (*Kapitel 4.3.6.3 Keep-Alive*).

TABELLE 18 A7 – NODE IST NICHT EINGETRAGEN

Fehler	A8 – Node ist an der falschen Stelle eingetragen
Beschreibung	Die Anordnung der Nodes im Leaf-Set stimmt nicht.
Feststellen / Behandlung	Die Feststellung sowie die Behandlung dieses Fehlers übernimmt der Keep-Alive Algorithmus (*Kapitel 4.3.6.3 Keep-Alive*)

TABELLE 19 A8 – NODE IST AN DER FALSCHEN STELLE EINGETRAGEN

INFRASTRUKTUR

Es kann vorkommen, dass die Infrastruktur nicht wie geplant funktioniert. Da die genaue Infrastruktur jedoch nicht bekannt ist und auch dynamisch wechseln kann, ist es kaum möglich Fehler zu verhindern. Somit beschränkt sich die Fehlertoleranz im Bereich Infrastruktur auf die Erkennung und Behandlung von Fehlern.

Fehler	A9 – Netzwerkverbindung unterbricht
Beschreibung	Durch das Unterbrechen der Netzwerkverbindung, verliert ein Node die Verbindung zum P2P-Netzwerk.
Feststellen / Behandlung	Die Feststellung sowie die Behandlung dieses Fehlers übernimmt der Keep-Alive Algorithmus (*Kapitel 4.3.6.3 Keep-Alive*).

TABELLE 20 A9 – NETZWERKVERBINDUNG UNTERBRICHT

4.3.6.3 KEEP-ALIVE

Keep-Alive Mechanismen werden verwendet, um Netzwerkverbindungen am Leben zu erhalten oder sich zu vergewissern, dass Netzwerketeilnehmer noch erreichbar sind. Keep-Alive Signale sind spezifische Pakete, welche im Netzwerkprotokoll definiert sind. Sie werden in regelmässigen Zeitabständen ausgesendet und erwarten in einer vorgegebenen Zeit eine Antwort. Bleibt die Antwort mehrfach hintereinander aus, kann davon ausgegangen werden, dass der Netzwerkteilnehmer nicht mehr erreichbar ist. Ist eine spezielle Behandlung dieses Ereignisses nötig, kann diese anschliessend abgearbeitet werden.

Würde in einem P2P-Netzwerk für jeden bekannten Node dieses Verfahren einsetzt werden, ergäbe das alleine vom Keep-Alive Algorithmus einen riesigen Datenverkehr. In den ersten P2P-Netwerken wurde dies noch so implementiert. Das hatte jedoch zur Folge, dass zum Beispiel bei frühen Gnutella Versionen bis zu 50% des Netzwerkverkehrs von Keep-Alive verursacht wurde [12].

Ivan Dedinski, Alexander Hofmann und Bernhard beschrieben in *Cooperative Keep-Alives: An Efficient Outage Detection Algorithm for P2P Overlay Networks* [12] den Cooperative Keep-Alive (CKA) Algorithmus, welcher die Menge von Nachrichten markant reduziert.
In diesem Algorithmus übernimmt jeder Node die Verantwortung für einige andere Nodes. Sobald einer dieser Nodes nicht mehr erreichbar ist, müssen alle Nodes, welche diese Information benötigen, darüber informiert werden. Durch die zusätzlich benötigten Daten, wie zum Beispiel, welche Nodes bei einem Ausfall informiert werden müssen, steigt die Menge der Daten in den einzelnen Keep-Alive Paketen leicht an. Die Summe der zu versendenden Pakete reduziert sich jedoch markant. In der folgenden Tabelle wird der CKA mit dem Standard Keep-Alive Algorithmus (SKA) gegenübergestellt:

Eigenschaften	SKA	CKA
Speicher	$O(d)$	max. $O(d^2)$, min. $O(d)$
Netzwerkverkehr	$O(N \cdot d)$	Schlimmsten Fall $O(N \cdot d)$ Normalfall $O(N)$
Verzögerung	Weniger als K Sekunden	Mit hoher Wahrscheinlichkeit weniger als K. Exponentiell sinkende Wahrscheinlichkeit für hohe Verzögerungen.

TABELLE 21 ZUSAMMENFASSUNG VON SKA UND CKA

d: durchschnittlicher Grad der Nodes
N: Anzahl Nodes
K: Keep-Alive Intervall [12]

In MobileCloud wird ein an CKA angelehnter Algorithmus verwendet. Durch die starke Netzwerkstruktur kann die Summe der Pakete sowie deren Grösse nochmals leicht reduziert werden.

Jeder Node R ist in MobileCloud nur für seinen direkten rechten Nachbarn N verantwortlich. Ist dieser nicht mehr erreichbar, müssen alle anderen Nachbarn informiert werden, welche N in ihren Leaf-Sets haben. Das sind genau diese Nodes, welche *N* im Leaf-Set hatte. Da *R* und *N* Nachbarn sind, kennt *R* alle Nachbarnodes von *N* bis auf den ganz rechts aussen. Durch das Fluten der Logout-Nachricht, kann nun *R* diese Nachricht anstelle von *N* absenden. Sie wird dann weitergeleitet bis kein Node mehr Interessen an dieser Nachricht hat. Der detaillierte Ablauf der Algorithmen ist im *Kapitel 4.3.7 Algorithmen* beschrieben.

4.3.7 Algorithmen

In diesem Kapitel werden weitere verwendete Algorithmen beschrieben.

4.3.7.1 Hash-Funktion

In P2P-Netzwerken wird oft ein Hash als ID eines Nodes verwendet. Dies erlaubt ein effizientes Suchen von Daten in Netzwerken, oft mittels einer verteilen Hashtabelle. Dabei werden Teilinformationen von Dateien gehasht und die Dateien bei den Nodes gespeichert, dessen ID am nächsten bei der gehashten Information ist. Beim Suchen einer Datei im Netzwerk kann anschliessend einfach durch angeben der benötigten Information ein Hash erzeugt werden. Mit diesem berechneten Hash kann danach der Node gesucht werden, welcher die Datei gespeichert hat. Wie dieser Vorgang genau funktioniert und welche verschiedenen Möglichkeiten es gibt, kann in Peer-to-Peer-Netzwerke Algorithmen und Methoden [13] nachgelesen werden.

Die Hash-Funktion wird in MobileCloud für die Generierung der ID eines Nodes benötigt. Diese wird aus der MSISDN[9] berechnet. Wichtig dabei ist, dass die Informationen über das Land und den Funknetzbetreiber nicht verloren gehen. Diese werden für die Anordnung der Nodes im Ring benötigt. Auf dem Ring sind die Nodes in aufsteigender Reihenfolge ihrer ID's angeordnet, was eine Gruppierung der Nodes mit der gleichen Vorwahl ergibt. So werden die Nodes, welche den gleichen Mobilfunkanbieter haben, nebeneinander platziert. Durch diese Platzierung steigt die Wahrscheinlichkeit, dass Nodes, welche oft miteinander kommunizieren müssen auch physikalisch in der Nähe sind.

Wie im SPMP[10]definiert ist, stehen für die ID sieben Bytes zur Verfügung. Damit können circa $7.2 \cdot 10^{16}$ verschiedene Werte codiert werden. Die maximale Länge einer MSISDN beträgt 15 Zeichen, welche somit alle mit den vorhandenen sieben Bytes abgebildet werden können. Weitere Informationen über SPMP sind im *Kapitel 4.3.4 SPMP* aufgelistet.

Die Generierung des Hash Wertes aus einer MSISDN wird auf eine einfache Art und Weise umgesetzt. Die MSISDN wir in einem numerischen Wert abgebildet. Dieser Wert wird danach in einem Bytearray in Big-Endian[11] Reihenfolge abgespeichert. Das bedeutet, dass das höchstwertige Byte beim Index 0 gespeichert wird. Diese Reihenfolge muss beim Vergleichen der ID's beachtet werden.

[9] Mobile Subscriber Integrated Services Digital Network Nummer (MSISDN) ordnet einem Mobiltelefon eindeutig er Nummer zu. Diese Nummer wird oft auch einfach Telefonnummer genannt.
[10] Simple Peer-to-Peer Management Protocol (SPMP), ist ein Protokoll für das Administrieren von P2P Netzwerken und wurde in der Studienarbeit MobileCloud definiert.
[11] Big-Endian definiert eine Speicherorganisation für einfache Zahlenwerte, bei welcher das höchstwertige Byte zuerst abgespeichert wird.

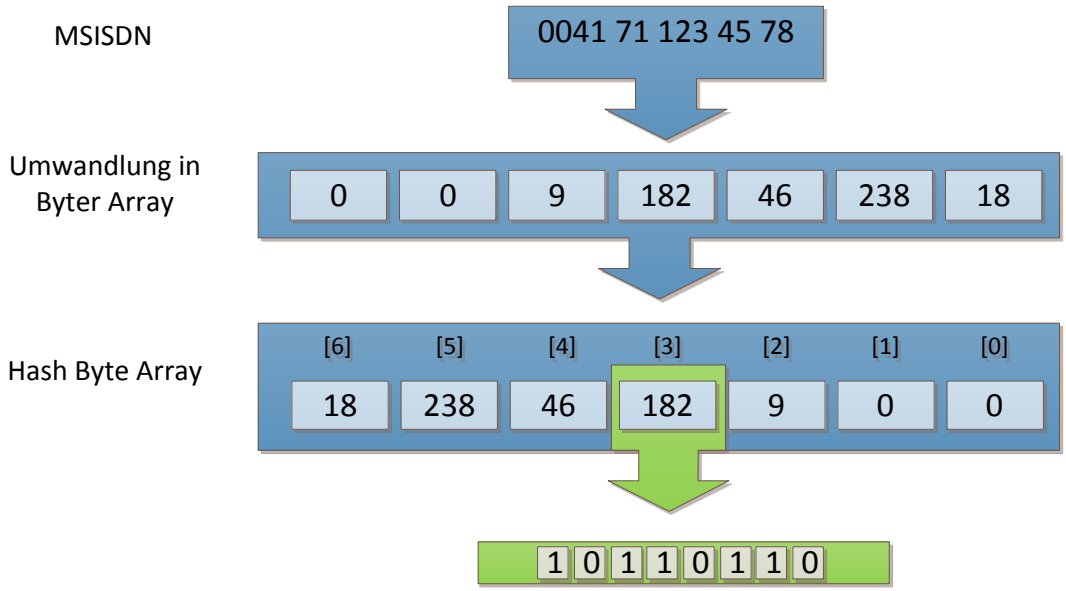


ABBILDUNG 9 GENERIERUNG HASH ID

Für Geräte, welche MobileCloud verwenden möchten, aber keine MSISDN haben, kann auch eine Nummer, welche noch nicht belegt ist frei gewählt werden. Um Überschneidungen auszuschliessen, müsste im produktiven Betrieb eine zentrale Stelle geschaffen werden, welche diese Nummern vergibt, um so doppelt vergebene Nummern so auszuschliessen.

4.3.7.2 SERVICES ALGORITHMEN

Alle Funktionen werden als MobileCloud Services betrachtet. Dazu gehören auch Verwaltungsfunktionen wie etwa das Betreten oder das Verlassen der MobileCloud. Dies erlaubt die nachträgliche Erweiterung mit geringem Aufwand von zusätzlichen MobileCloud Services. Auch können so schnellere Algorithmen implementiert sowie bestehende erweitert werden. Es ermöglicht weiter, dass spezifische Routingmechanismen für einzelne Services implementiert werden können.

MobileCloud Services bestehen normalerweise aus zwei Handler, einem für die Anfrage (*handler*) und einem für die Antwort (*request handler*). Es kann aber vorkommen, dass einzelne MobileCloud Services nur mit einem Handler auskommen oder einen bereits bestehenden verwenden.

Die grundlegendsten Algorithmen wurden während der Studienarbeit implementiert und dokumentiert. In diesem Kapitel werden nur noch Services beschreiben, welche angepasst wurden oder neu hinzugekommen sind.

Genauere Informationen zu den Handlern sind aus der Studienarbeit [2] oder in den UML Diagrammen zu entnehmen.

In MobileCloud gibt es folgende Services und Handler:

Name	Kurzbeschreibung	Verwendete Handler
join	Der MobileCloud beitreten.	JoinHandler
leave	Die MobileCloud verlassen.	LeaveHandler

TABELLE 22 MOBILECLOUD SERVICES

Name	Kurzbeschreibung	Verwendete Handler
JoinHandler	Bearbeitet den Service join.	UpdateHandler, UpdateLeafSetHandler
UpdateHandler	Bearbeitet einen neu hinzugekommenen Node .	
UpdateLeafSetHandler	Bearbeitet die Aktualisierung des LeafSets.	
LeaveHandler	Bearbeitet den Service leave.	UpdateLeafSetHandler
PingPongHandler	Überprüft die Verfügbarkeit eines Nodes.	
KeepAliveHandler	Überprüft periodisch die Verfügbarkeit des rechten Nachbarnodes.	PingPongHandler, UpdateLeafSetHandler, LeaveHandler
HolePunchingHandler	Führt die NAT-Traversierung durch.	HolePuncher

Tabelle 23 MobileCloud Handler

	Bestandteil der Studienarbeit
	Wurde in der Bachelor Arbeit erweitert.
	Wurde in der Bachelor Arbeit neu erstellt.

Tabelle 24 Legende Handler

UpdateLeafSetHandler

Beim UpdateLeafSetHandler wurde zwei Änderung vorgenommen:

- Es wird nicht mehr der nächste sondern der entfernteste Nachbar für das Leaf-Set angefragt.
- Bei einer Anfrage wird das gesamte Leaf-Set zurückgesendet.

Durch diese zwei Änderungen ist es möglich, die Aktualisierung in einem Schritt zu erledigen, auch wenn alle Nodes auf einer Seite bis auf den Äusserten nicht mehr erreichbar sind.

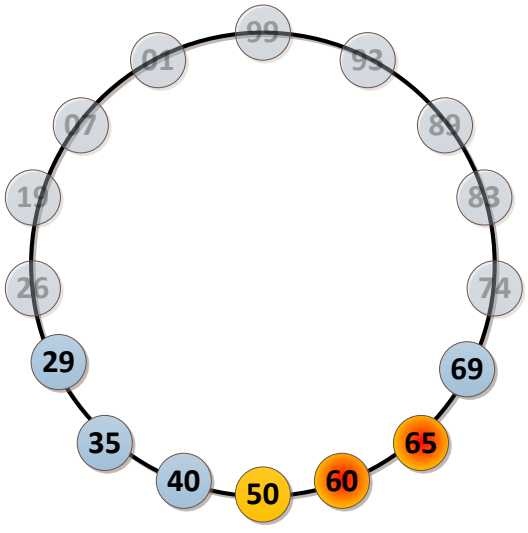


Abbildung 10 Vor der Aktualisierung

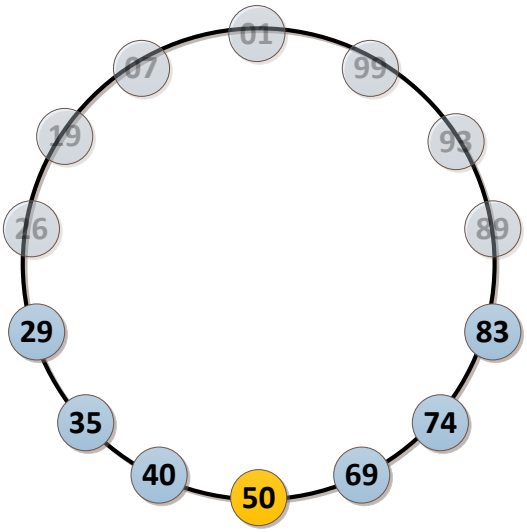


Abbildung 11 Nach der Aktualisierung

Wie in der *Abbildung 10*Abbildung 12 veranschaulicht, ist der einzige noch aktive Node auf der rechten Seite derjenige mit der ID 69. Bei einem Update werden dem Node 50 die Adressen {50,60,65,74,83,89} gesendet. Mit diesen Adressen ist es ihm möglich das Leaf-Set in einen konsistenten Zustand zu überführen. Dieses wird in *Abbildung 11* dargestellt.

LeaveHandler

Beim Verlassen des Netzwerks wird die Nachricht nur noch an die zwei direkten Nachbarn gesendet. Diese leiten die Nachrichten so lange im Kreis weiter, bis der Empfänger den Node, welcher das Netzwerk verlässt nicht im Leaf-Set gespeichert hat. So erhalten alle, die den Node kennen, und daher informiert werden müssen, diese Logout-Nachricht.

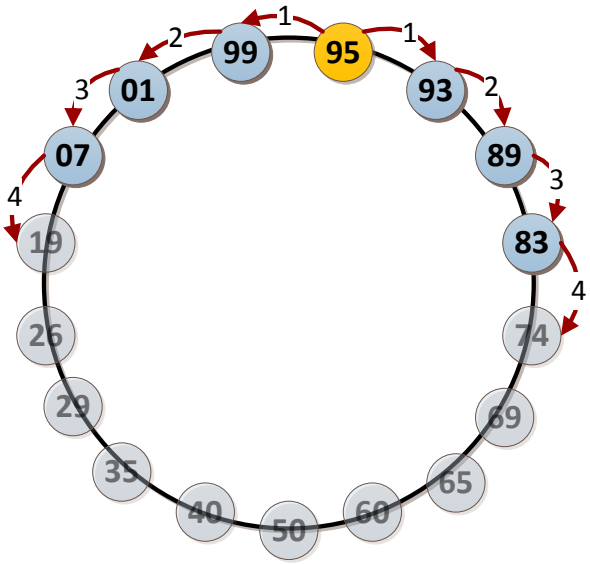


Abbildung 12 Logout-Nachricht-Weiterleitung beim Verlassen der MobileCloud

Diese Änderung ermöglicht es mehrere Nodes mit einer einzigen Nachricht aus dem Netzwerk zu entfernen. Hierfür muss lediglich eine Liste von Nodes anstatt eines einzelnen Nodes mitgesendet werden. Gleichzeitig muss der Node, welcher diese Nachricht absendet nicht wissen, wer alles über sein Verlassen informiert werden muss.

PingPongHandler

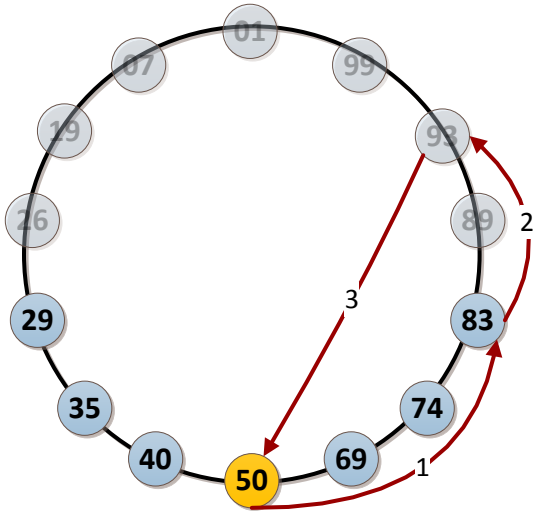


Abbildung 13 PingPongHandler

Dieser Handler wird verwendet, um zu überprüfen, ob ein Node noch aktiv ist. Dafür wird eine Nachricht an den zu überprüfenden Node gesendet (1+2). Kommt innerhalb einer vorgegebenen Zeit keine Antwort (3) zurück, gilt der Node als inaktiv.

Das Weiterleiten (Routing) von Nachrichten (2) ist in der Studienarbeit *MobileCloud* [2] beschrieben.

KeepAliveHandler

Die Hauptaufgabe des *KeepAliveHandler* ist es, Inkonsistenzen im Leaf-Set zu erkennen und anschliessend die nötigen Massnahmen zur Überführung in einen konsistenten Zustand einzuleiten. In der unten aufgeführten Tabelle sind inkonsistente Situationen aufgezeigt, welche erkannt und behoben werden.

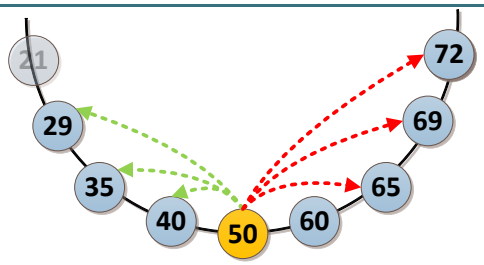
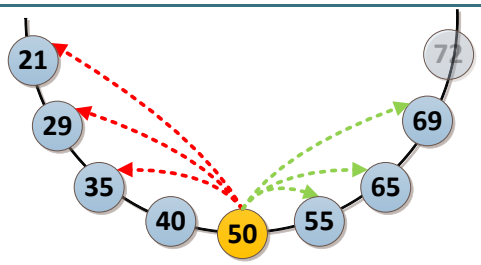
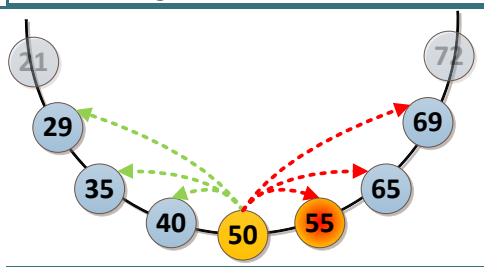
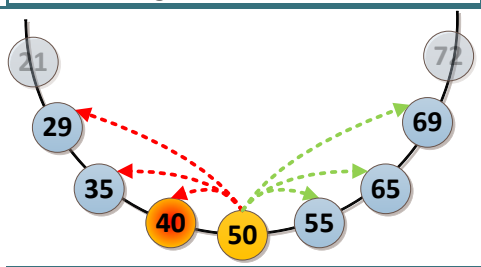
	Vorwärts Referenz	Rückwärts Referenz
Fehlender Node		
	Detektion: Node 65 **Behandlung:** Node 50	**Detektion:** Node 50 **Behandlung:** Node 50
Ungültiger Node		
	Detektion: Node 50 **Behandlung:** Node 50	**Detektion:** Node 35 **Behandlung:** Node 35

TABELLE 25 INKONSISTENTE LEAF-SET SITUATIONEN

Die oben aufgezeigten Fehlsituationen werden wie folgt behandelt:

Fehlender Node bei vorwärts Referenz	
Detektion Node 65	1. Der Node bekommt eine Keep-Alive Nachricht, jedoch nicht von dem von ihm erwarteten Node (60). 2. Der Absender (50) befindet sich nicht zwischen dem eigenen und dem erwarteten Node. Somit liegt das Problem beim Absender. 3. Der detektierte Fehler wird zurückgesendet.
Behandlung Node 50	1. Es wird der nächste im Leaf-Set fehlende aktive Node gesucht. 2. Das Leaf-Set wird mittels des aktiven Nodes aktualisiert.

TABELLE 26 FEHLENDER NODE BEI VORWÄRTS REFERENZ

Fehlender Node bei rückwärts Referenz	
Detektion Node 50	1. Der Node bekommt eine Keep-Alive Nachricht, jedoch nicht von dem von ihm erwarteten Node (35). 2. Der Absender (40) befindet sich zwischen dem eigenen und dem erwarteten Node. Somit liegt das Problem beim eigenen Leaf-Set.
Behandlung Node 50	1. Der Absender wird dem Leaf-Set hinzugefügt.

TABELLE 27 FEHLENDER NODE BEI RÜCKWÄRTS REFERENZ

Ungültiger Node bei vorwärts Referenz	
Detektion Node 50	1. Auch nach dem dritten Versuch wird keine Antwort empfange. Der Node 55 wird als inaktiv detektiert.
Behandlung Node 50	1. Es wird der nächste aktive Node gesucht. 2. Das Leaf-Set wird mittels des aktiven Nodes aktualisiert. 3. Die Nachbarschaft wird über den inaktiven Node informiert.

TABELLE 28 UNGÜLTIGER NODE BEI VORWÄRTS REFERENZ

Ungültiger Node bei rückwärts Referenz	
Detektion Node 35	Wird gleich behandelt wie *Ungültiger Node bei vorwärts Referenz*.
Behandlung Node 35	Wird gleich behandelt wie *Ungültiger Node bei vorwärts Referenz.*

Tabelle 29 Ungültiger Node bei rückwärts Referenz

HolePunchingHandler

Die Aufgabe des *HolePunchingHandlers* ist es, den in *Kapitel 4.3.5.4 Hole Punching* beschriebenen Algorithmus zu realisieren. Die genaue Funktionsweise wird ausführlich im oben genannten Kapitel beschrieben.

4.3.8 Tests

Das Testen in einem verteilten System ist bekannter Weise schwierig und aufwändig. Da es jedoch von Vorteil ist, dass Software getestet wird, wurden zwei Umsetzungsansätze gewählt. Der erste mit Unit Test und der zweite mittels Visualisierung der Informationen der Nodes in der MobileCloud.

Die Benutzertests wurden stets mit mobilen Geräten durchgeführt, somit wurde sichergestellt, dass die Software auch auf verschiedener Hardware läuft. Mit folgenden Modellen wurde die Software getestet:

- HTC Desire Z
- Google Samsung Nexus S
- HTC Desire
- HTC Desire HD

4.3.8.1 Unit Test

Für das Testen der einzelnen Klassen wurden Unit Test geschrieben. Diese sind in einem gesonderten Projekt *MobileCloudTest* abgelegt.

Das Ziel war es, vorwiegend die wichtigsten und komplexesten Funktionalitäten zu testen und eine Abdeckung von rund 70% zu erreichen.

Die Auswertung der Tests am Ende der Studienarbeit ergab:
Anzahl Tests: 234

Was	Abdeckung in Prozent	Abdeckung absolut	Abdeckung maximal
Klassen	67.1%	55	82
Methoden	67%	310	463
Blocks	63.4%	6197	9772
Linien	66.8%	1383	2071

Tabelle 30 Testabdeckung

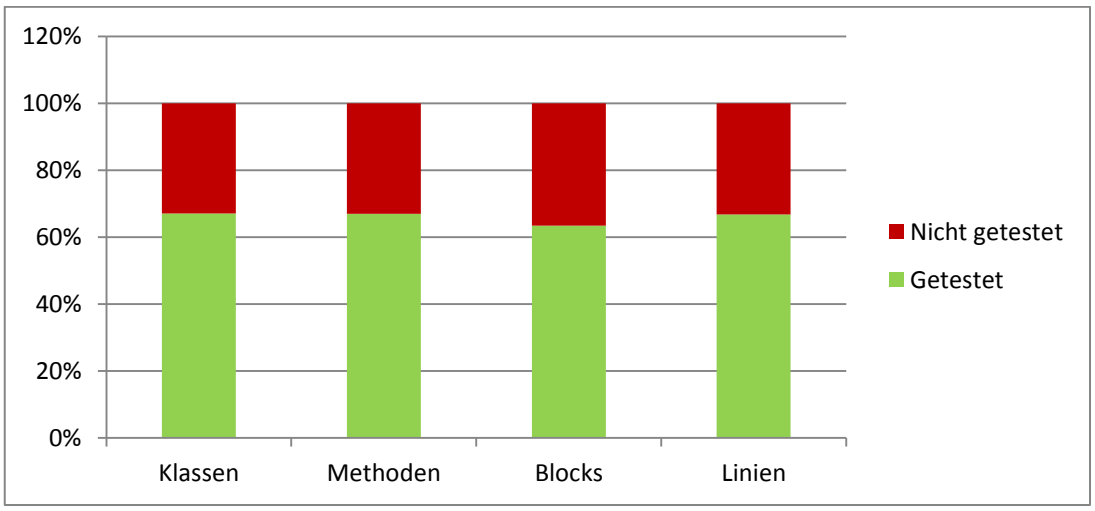


Abbildung 14 Testabdeckung

4.4 NGSON

4.4.1 Einführung

Am 10 September 2011 veröffentlichte das IEEE den Standard 1903: *Next Generation Service Overlay Network*. Wie der Name sagt, definiert dieser Standard die funktionale Architektur eines überliegenden Servicenetzwerkes.

Hier wird nur eine kurze Übersicht über den Standard gegeben. Weitere Informationen sind direkt aus dem Standard zu entnehmen [14].

4.4.1.1 Business Motivation

Die Provider der Netzwerke haben ein klares Interesse mehr zu sein als nur die Vermieter von Leitungen. Dies war bereits beim ISDN[12] Standard der Fall. Dort konnten neben der Sprache noch zusätzliche Daten von verschiedenen Services übertragen werden.
Durch die starke Verbreitung der IP-basierten Netzwerke stieg auch das Interesse, in den IP-Netzen solche Services anzubieten. Mit den schnellen IP-Netzwerken entstand auch die Möglichkeit, viel umfangreichere und komplexere Services anzubieten.

Ein Schritt weiter geht die SOA[13]. Bei dieser Architektur werden ganze Geschäftsprozesse mittels einzelnen Services abgebildet. Durch die Zusammenführung/Verkettung von eigenständigen Services entstehen so ganze Programm-Logiken. Ein Service Broker übernimmt dabei die Verwaltung der Services, was für den Benutzer die Nutzung völlig transparent macht.

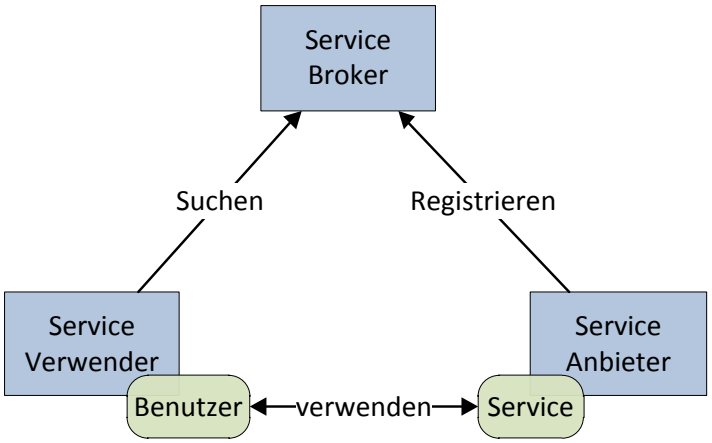


Abbildung 15 Serviceorientierte Architektur

4.4.1.2 Der Standard

Der IEEE 1903 Standard beschreibt ein auf IP-basiertes Framework für Service-Netzwerke. Der Fokus bezieht sich auf das Kontextbewusstsein (*context awareness*), die dynamische Anpassbarkeit (*dynamically adaptive*) und die Selbstorganisation (*self-organisation*).

Für die externe Kommunikation gibt es vier Schnittstellen (siehe *Abbildung 16*).

[12] Integrated Services Digital Network (ISDN) ist ein Standard für ein digitales Telekommunikationsnetzwerk über welches verschiedene Dienste wie z.B. Teletext angeboten werden können.
[13] Serviceorientierte Architektur (SOA) ist ein Architekturmuster welches verwendet wird um Geschäftsprozesse mit Services abzubilden.

- **S1**: Wird von Programmen und Services verwendet, um mit NGSON zu kommunizieren. Auch Drittanbieter können über diese Schnittstelle ihre eigenen Services publizieren.
- **S2**: Wird für die Kommunikation mit anderen NGSON's von anderen Anbietern verwendet.
- **S3**: Wird für die Kommunikation mit dem darunterliegenden Netzwerk verwendet. Dabei kann es sich um ein P2P-Netzwerk, Sockets oder sonstige IP basierten Technologien handeln.
- **S4**: Wird vom Endbenutzer verwendet, um die vom NGSON zu Verfügung gestellten Funktionen zu benutzen.

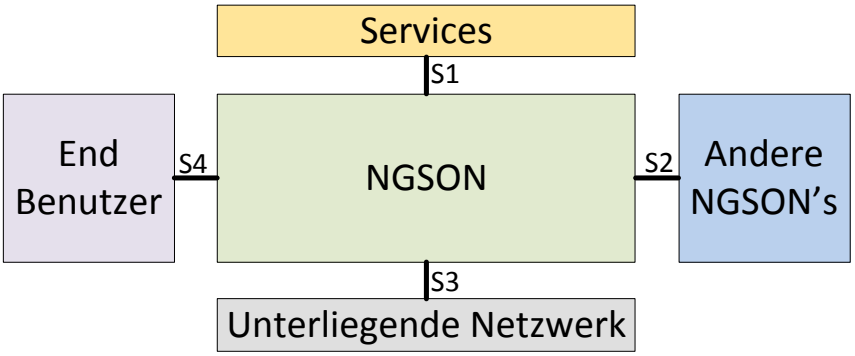


Abbildung 16 externe Schnittstellen

Das NGSON wurden in verschiedene Funktionsendpunkte (FE) unterteilt. Jeder dieser Endpunkte ist für einen Bereich zuständig und bietet anderen Endpunkten gewisse Funktionen an.
In der *Abbildung 17* sind alle Funktionsendpunkte und deren Beziehung zueinander abgebildet.

Die blau hinterlegten Funktionsendpunkte waren Bestandteil dieser Arbeit, die rot hinterlegten nicht. Weitere Informationen zur Umsetzung von NGSON sind im *Kapitel 4.4.2 Umsetzung* beschrieben.

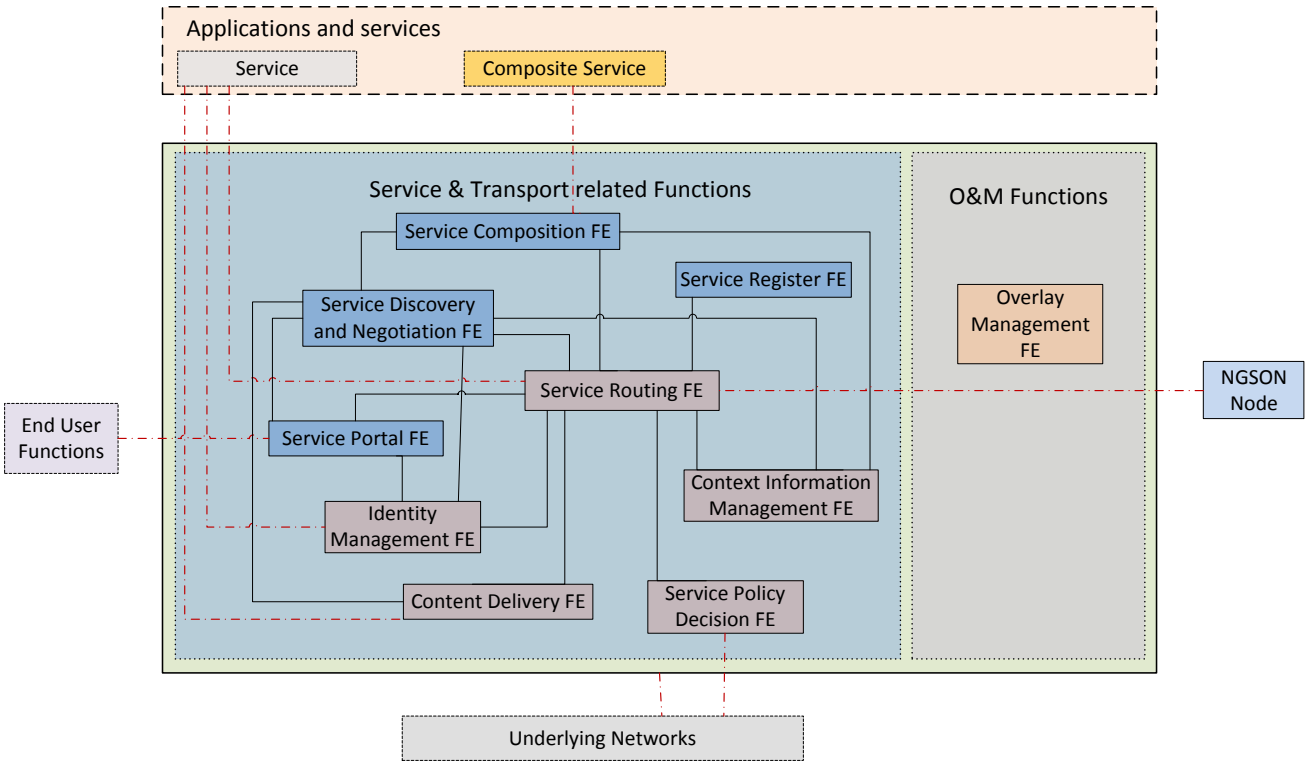


ABBILDUNG 17 NGSON ARCHITEKTUR DIAGRAMM

4.4.1.3 NGSON UND P2P

Ein NGSON Netzwerk besteht meistens aus mehreren Netzwerkkomponenten. Diese werden NGSON-Nodes genannt. Ein grosser Vorteil von NGSON ist die Selbstorganisation innerhalb des Netzwerkes. So können dynamisch neue Nodes hinzugefügt oder entfernt werden. Diese Funktion übernimmt der *Overlay Management* Funktionsendpunkt.
Es gibt drei unterschiedliche Arten wie das NGSON aufgebaut werden kann:

- **Zentral verwaltet**: Nur ein NGSON-Node im NGSON implementiert den *Overlay Management* Funktionsendpunkt. Dieser ist für die Verwaltung aller anderen NGSON-Nodes verantwortlich.
- **Verteilt verwaltet**: Jeder NGSON-Node implementiert den *Overlay Management* Funktionsendpunkt. Dies ist der typische Peer-to-Peer Ansatz, bei dem alle Teilnehmer gleichberechtigt sind.
- **Gemischt verwaltet**: Nur ein Teil der NGSON-Node implementiert den *Overlay Management* Funktionsendpunkt. Diese verwalten zusammen das NGSON. Dies ist ein typischer Peer-to-Peer Ansatz, in welchem mit Supernodes[14] gearbeitet wird.

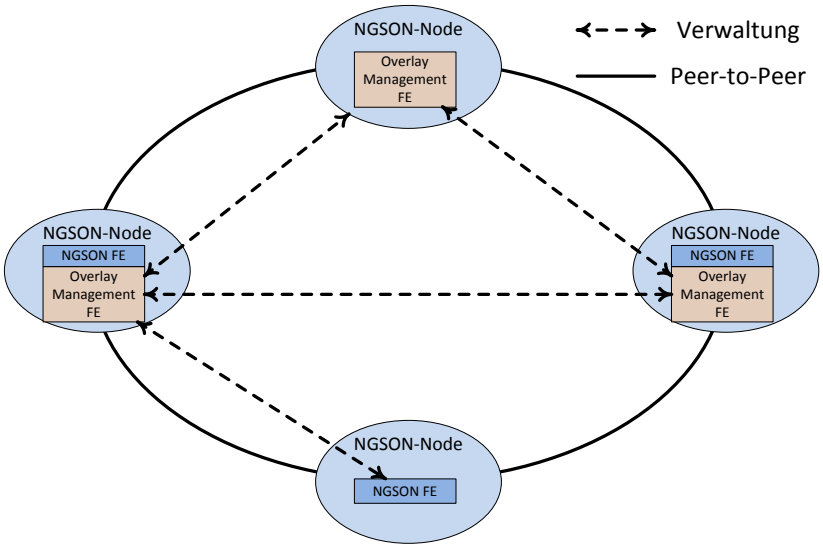


ABBILDUNG 18 GEMISCHT VERWALTETES NGSON

Als überliegendes Netzwerk zwischen NGSON und dem IP-basierten Netzwerk eignet sich ein Peer-to-Peer-Netzwerk für alle drei Arten der Verwaltung. Denn das Peer-to-Peer-Netzwerk übernimmt viele Aufgaben, wie das einbindend eines neuen oder das Aufsuchen eines bestimmten NGSON-Nodes im Netzwerk.

[14] Supernodes sind Computer, welche einen Knotenpunkt in einem Peer-to-Peer-Netzwerk einnehmen.

4.4.1.4 NGSON IN DER ZUKUNFT

Das IEEE hat am 7 Dezember 2011 bestätigt, dass drei zusätzliche Standards, welche den bisherigen NGSON-Standard erweitern in Arbeit sind [15].

- **P1903.1, P1903.2**: Diese Standards spezifizieren die Protokolle für die Kommunikation zwischen den Funktionsendpunkten.
- **P1903.3**: Dieser Standard spezifiziert das Protokoll für die Kommunikation zwischen den einzelnen NGSON-Nodes.

4.4.2 Umsetzung

Umgesetzt wurde ein NGSON-Node, mit welchem über REST kommuniziert werden kann. Für die Nutzung wurde eine Android Applikation entwickelt, mit welcher registrierte Dienste verwendet werden können.

4.4.2.1 Aufbau des Projekts

NGSON with Peer-to-Peer besteht aus vier einzelnen Projekten:

NgsonWithP2P
Ist das Projekt, welches den NGSON-Node beinhaltet. Darin sind alle Funktionalitäten enthalten, welche direkt mit der NGSON-Standard Implementation zu tun haben.

NgsonWithP2PClient
Dieses Projekt beinhaltet den Android Client für den Zugriff auf das NGSON.
Es ist von keinem anderen Projekt abhängig.

CompressRestService
Ist das Projekt, welches den Rest-Service für das herunterladen und Zippen von Fotos beinhaltet.
Es benötigt die Projekte NgsonWithP2P und FotoRestService.

PhotoRestService
Das Projekt PhotoRestService beinhaltet die zwei Rest-Service für das Suchen und Skalieren von Bildern.
Dieses Projekt benötigt das Projekt NgsonWithP2P.

4.4.2.2 Ziel und Fokus

Der NGSON-Standard umfasst eine Vielzahl von Funktionen. Die Umsetzung aller Funktionen würde den Rahmen dieser Bachelorarbeit sprengen. Deshalb wurden für die Umsetzung drei grundlegende Funktionen von NGSON ausgewählt.

- Registrieren, deregistrieren und aktualisieren von Services
- Kontextbasiertes abfragen und ausführen von Services
- Erstellen und ausführen von verketteten Services.

Das Ziel war es aufzuzeigen, wie ein NGSON grundsätzlich funktioniert, verwendet werden kann und welche Vorteile es bietet. Für die Verwaltung von NGSON sollte das im ersten Teil der Bachelorarbeit erweiterte *MobileCloud*-Framework verwendet werden.

Anhand eines Beispiels sollte sichtbar gemacht werden, was die oben erwähnten Funktionen für Vorteile bringen und wie diese zusammenarbeiten. Dafür durften bereits existierende oder zusätzlich erstellte Webservices verwendet werden.

Die wichtigsten Fragen, welche mit diesem Prototyp beantwortet werden sollten sind:

- Wie kann eine Umsetzung von NGSON aussehen?
- Wie kann NGSON verwendet werden?
- Was sind die Mehrnutzen für den Endanwender?
- Wie kann NGSON von einem Client benutzt werden?

- Wie können kontextbasiertes Ausführen und Zusammensetzen von Services realisiert werden?

4.4.2.3 Übersicht

Das NGSON stellt nur die Infrastruktur für das Arbeiten mit den Services zur Verfügung. Es fungiert als Vermittler zwischen den effektiven Serviceanbietern und den Endbenutzern. Es unterstützt das Verwenden und Verwalten der Services mit zusätzlicher Funktionalität. Bevor Services genutzt werden können, müssen diese jedoch registriert werden. Dies muss nicht vom selben Anbieter gemacht werden wie derjenige, welcher das NGSON unterhält.

Bei der Umsetzung des NGSON-Nodes wurden deshalb drei Rollen definiert, welche mit dem NGSON interagieren:

- **Endbenutzer**: Verwendet die eigenständigen oder verketteten Services über den NGSON-Node. Kann eine Liste mit allen für ihn verfügbaren Services abfragen.
- **Serviceanbieter**: Registriert, aktualisiert und entfernt eigenständige oder verkettete Services.
- **Service**: Bieten Dienstleistungen an und können von den Endbenutzern oder dem NGSON-Node für verkettete Aufrufe verwendet werden.

Die Aufteilung in die verschiedenen Komponenten (Tiers) ist ähnlich der Rollenverteilung.

- **Endbenutzer**: Kommuniziert mit dem NGSON-Node für die Abfrage und Benutzung von Services, sowie mit dem Service für dessen Benutzung.
- **Service**: Wird vom NGSON-Node und dem Endbenutzer verwendet. Verwaltet werden die Services von den Serviceanbietern.
- **Serviceanbieter**: Verwaltet die eigenständigen und die verketteten Services auf dem NGSON-Node.
- **NGSON-Node**: Verwaltet alle Arten von Services und führt verkettete Services aus. Für das Speichern der Services greift er auf den Persistenz-Tier zu.
- **Persistenz**: Speichert, aktualisiert und löscht die Services, welche über den NGSON-Node verwaltet werden.

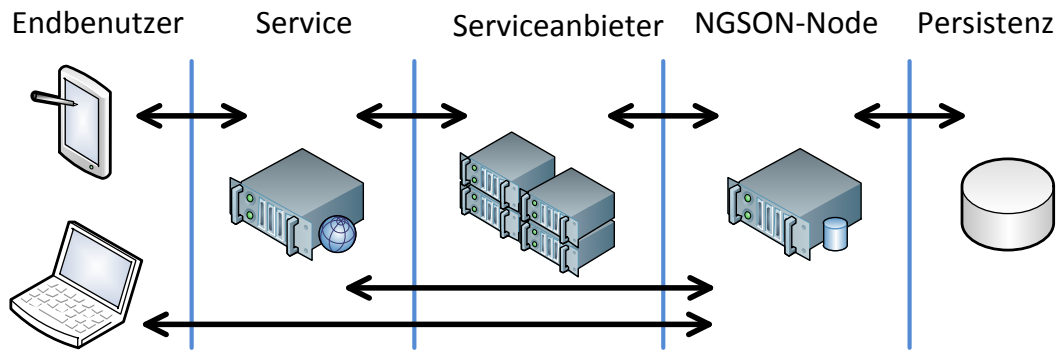


Abbildung 19 Tier Übersicht

4.4.2.4 FUNKTIONEN

Aus den Anforderungen ergeben sich die folgenden drei Schnittstellen mit den dazugehörigen Funktionen:

Schnittstelle 1: Services verwalten	
Rolle: Serviceanbieter	
Funktion 1	Einen neuen Service hinzufügen.
Funktion 2	Einen bestehenden Service abrufen.
Funktion 3	Einen bestehenden Service aktualisieren.
Funktion 4	Einen bestehenden Service entfernen.
Funktion 5	Einen neuen verketteten Service hinzufügen.
Funktion 6	Einen bestehenden verketteten Service abrufen.
Funktion 7	Einen bestehenden verketteten Service aktualisieren.
Funktion 8	Einen bestehenden verketteten Service entfernen.

TABELLE 31 SCHNITTSTELLE SERVICES VERWALTEN

Schnittstelle 2: Services abrufen	
Rolle: Endbenutzer	
Funktion 1	Liste mit verwendbaren Services abfragen.

TABELLE 32 SCHNITTSTELLE SERVICES ABRUFEN

Schnittstelle 3: Verkettete Services verwenden	
Rolle: Endbenutzer	
Funktion 1	Verkettete Services ausführen.

TABELLE 33 SCHNITTSTELLE VERKETTETE SERVICES VERWENDEN

Von allen im Standard definierten Funktionsendpunkten, welche in der *Abbildung 17* aufgezeigt sind, werden nur einige verwendet, um den Rahmen der Arbeit nicht zu sprengen. Die folgenden Beschreibungen wurden vom Standard [14] abgeleitet.

- **Service Composition FE**: Dieser Funktionsendpunkt ist zuständig für die Ausführung der verketteten Services. Er enthält auch die Logik für das Zusammenstellen von dynamisch verketteten Services.
- **Service Discovery and Negotiation FE**: Verwendet die Informationen über die in NGSON registrierten Services, um die optimalen zu gegebenen Kriterien zu suchen.
- **Service Portal Fe**: Ermöglicht den Endbenutzern NGSON zu verwenden. Bietet die Möglichkeit für Benutzer sich zu registrieren und sich zu authentisieren.
- **Services Register FE**: Unterstützt das Registrieren und Verwalten von dynamischen Serviceinformationen.

4.4.2.5 NGSON-Node

Der NGSON-Node ist der zentrale Punkt des NGSON's. Er implementiert alle Funktionen, welche vom NGSON angeboten werden. Das Verteilen der Funktionen auf mehrere NGSON-Nodes und die Kommunikation zwischen diesen war nicht Teil dieser Arbeit. Deshalb besteht die Arbeit aus nur einem NGSON-Node.

Architektur

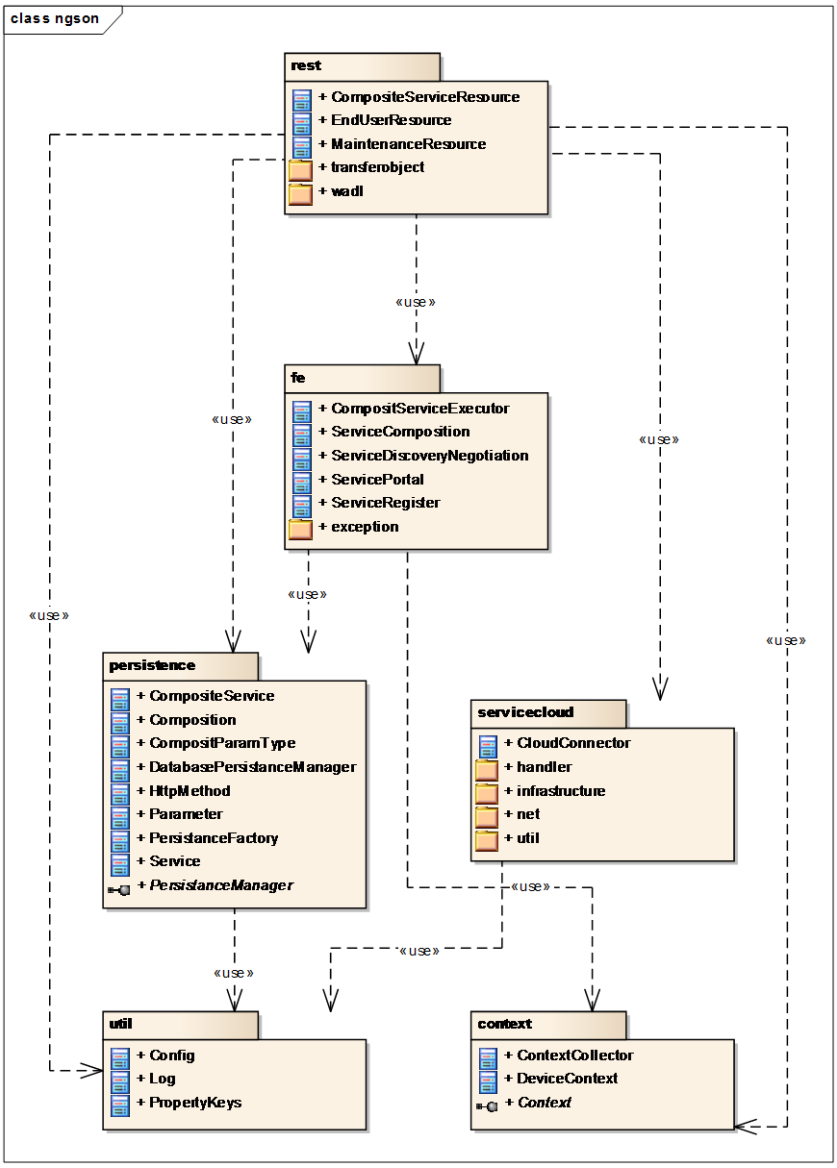


Abbildung 20 NGSON-Node Architektur

Der NGSON-Node ist in mehrere Pakete unterteilt. Die verschiedenen Pakete und deren Abhängigkeiten sind in der *Abbildung 17* zu sehen.
Die Pakete haben folgenden Fokus:

rest

Dieses Paket beinhaltet alle REST-Ressourcen, welche von aussen angesprochen werden können. Zusätzlich enthalten sind noch die Transferobjekte, welche für die Übertragung der Daten verwendet werden.

fe

Dieses Paket enthält die Funktionsendpunkte. Diese wurden aus dem Standard übernommen. Daneben sind noch Klassen enthalten, welche einzelne Aufgaben, wie zum Beispiel das parsen von WADL, übernehmen.

persistence

Dieses Paket ist für die Speicherung und das Abrufen der Services verantwortlich. Momentan ist das Speichern in einer PostgreSQL-Datenbank möglich. Durch das Abstrahieren mittels einer Factory kann das Paket mit weiteren Speichervarianten erweitert werden.

servicecloud

Dieses Paket beinhaltet das P2P-Framework. Es ist für Verwaltung des NGSONs zuständig. Weitere Informationen zur Verwaltung mittels P2P sind im *Kapitel 4.4.1.3 NGSON und P2P* zu finden.
Die aktuelle Version der *ServiceCloud* entspricht der Version 0.5.0 von *MobileCloud.*

context

Dieses Paket ist für die verschiedenen Kontexte verantwortlich. Zum gegebenen Zeitpunkt ist nur der Gerätekontext implementiert.

util

Dieses Paket beinhaltet die Konfiguration und die Logging Mechanismen.

Schnittstellen

Die Schnittstellen wurden mit REST umgesetzt. Jede Schnittstelle stellt eine oder mehrere Rest-Ressourcen dar. Diese können über die Konfiguration aktiviert oder deaktiviert werden. Als Webserver wurde das Jersey Framework verwendet.

Die Schnittstellen zu NGSON können dank REST über eine URL angesprochen werden. Wie die verschiedenen REST-Ressourcen auf die Funktionen abgebildet und wieder angesprochen werden, ist aus den nächsten Tabellen ersichtlich.

Die genaue Beschreibung aller Ressourcen als WADL kann über die URL http://{NGSON-Node Adresse}/ngson/application.wadl abgerufen werden. Die der Datentypen als XML-Schema kann über die URL http://{NGSON-Node Adresse}/ngson/application.wadl/xsd0.xsd abgerufen werden.

REST Ressource: MaintenanceResource	
Schnittstelle 1: Services verwalten	
Funktion:	Einen neuen Service hinzufügen.
URL:	http://{NGSON-Node Adresse}/ngson/maintenance
Befehl:	PUT
Parametertype:	serviceTo
Funktion:	Einen bestehenden Service abrufen.
URL:	http://{NGSON-Node Adresse}/ngson/maintenance/{id}
Befehl:	GET
Rückgabetype:	serviceTo
Funktion:	Einen bestehenden Service aktualisieren.
URL:	http://{NGSON-Node Adresse}/ngson/maintenance
Befehl:	PUT
Parametertype:	serviceTo
Funktion:	Einen bestehenden Service entfernen.
URL:	http://{NGSON-Node Adresse}/ngson/maintenance/{id}
Befehl:	DELETE
Funktion:	Einen neuen verketteten Service hinzufügen.
URL:	http://{NGSON-Node Adresse}/ngson/maintenance/comp
Befehl:	PUT
Parametertype:	serviceTo
Funktion:	Einen bestehenden verketteten Service abrufen.
URL:	http://{NGSON-Node Adresse}/ngson/maintenance/comp/{id}
Befehl:	GET
Rückgabetype:	compositServiceTo
Funktion:	Einen bestehenden verketteten Service aktualisieren.
URL:	http://{NGSON-Node Adresse}/ngson/maintenance/comp
Befehl:	PUT
Parametertype:	compositServiceTo
Funktion:	Einen bestehenden verketteten Service entfernen.
URL:	http://{NGSON-Node Adresse}/ngson/maintenance/comp/{id}
Befehl:	DELETE

Tabelle 34 Ressource Services verwalten

REST Ressource: EndUserResource	
Schnittstelle 2: Services abrufen	
Funktion:	Liste mit verwendbaren Services abfragen.
URL:	http://{NGSON-Node Adresse}/ngson/enduser/{type}
Befehl:	POST
Parametertype:	deviceContextTo
Rückgabetype:	serviceDescriptionList
Funktion:	Beschreibung eines Services als WADL abfragen.
URL:	http://{NGSON-Node Adresse}/ngson/enduser/{id}
Befehl:	GET
Rückgabetype:	XML

TABELLE 35 RESSOURCE SERVICES ABRUFEN

REST Ressource: CompositeServiceResource	
Schnittstelle 3: Verkettete Services verwenden	
Funktion:	Verkettete Services ausführen.
URL:	http://{NGSON-Node Adresse}/ngson/composite/execute/{id}
Befehl:	POST
Parametertype:	String
Rückgabetype:	*

TABELLE 36 RESSOURCE VERKETTETE SERVICES VERWENDEN

PERSISTIERUNG

Für die Speicherung der registrierten, eigenständigen und verketteten Services wurde eine relationale Datenbank eingesetzt, welche auf einem PostgreSQL-Server läuft. Dafür werden folgende Tabellen verwendet:

- **services**: Speichert die allgemeinen Informationen von Services, unabhängig davon, ob eigenständig oder verkettet.
- **compositions**: Speichert die Kompositionen, welche einzelne Aufrufe von Services repräsentieren. Nur verkettete Services enthalten Kompositionen.
- **parameter**: Speichert die einzelnen Parameter, welche in den Kompositionen verwendet werden.
- **settings**: Speichert Schlüssel-Werte Paare bezogen auf die Datenbank, wie zum Beispiel deren Version.

Die Tabellen in der Übersicht:

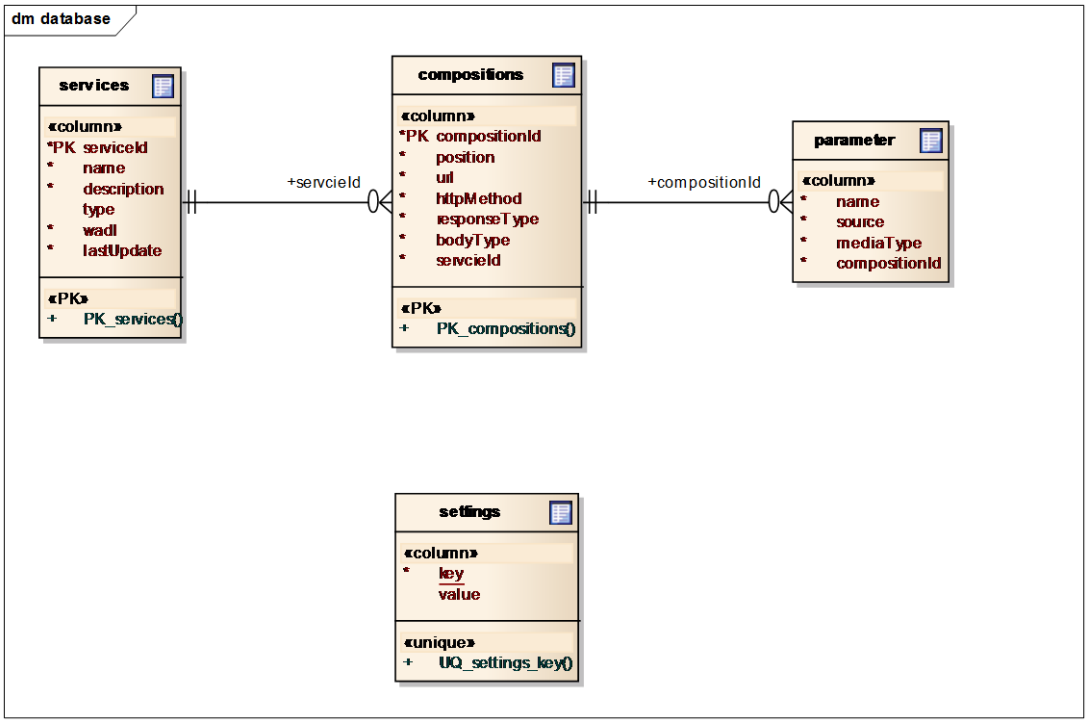


Abbildung 21 Entity-Relationship-Modell

Die Persistierung wurde über eine Factory realisiert. Das ermöglicht jederzeit die Art der Persistierung zu ändern. Beispielsweise könnten die Services nur im Arbeitsspeicher gehalten oder als XML abgelegt werden.

Kontextbasiertes Benutzen

Ein grosses Plus von NGSON ist das kontextbasierte Ausführen von Services. Dies ermöglicht es ohne das aktive Zutun des Endbenutzers, die auf ihn abgestimmten Services auszuführen. Zum Beispiel kann mittels des Gerätekontextes herausgefunden werden, welche Media-Typen[15] das Gerät des Endbenutzers unterstützt. Beim Abfragen von Services kann anhand dieser Informationen aussortiert und nur diejenigen Services zurückgeliefert werden, welche der Endbenutzer auch wirklich verwenden kann.

Für die kontextbasierte Ausführung wurde der Gerätekontext implementiert. Dieser ist auf ein Minimum beschränkt und beinhaltet nur die Bildschirmauflösung und die unterstützen Dateiformate. Diese Eigenschaften wurden aufgrund des Anwendungsbeispiels ausgewählt, welches im *Kapitel 4.4.2.8 Beispiel* genauer erläutert wird.

Bedienung

Der NGSON-Node wurde als Java Konsolenapplikation implementiert. Nach dem Starten werden alle Lognachrichten auf der Konsole ausgegeben. So können allfällige Probleme erkannt und behandelt werden. Für die aktive Steuerung gibt es folgende Kommandos, welche über die Kommandozeile eingegeben werden können:

[15] Der *Internet Media Type* bestimmt, wie die Daten im Rumpf einer Nachricht interpretiert werden sollen.

Kommando	Auswirkung
stop	Stoppt den NGSON-Node und beendet diesen.
restart	Startet den NGSON-Node neu. Dabei wird auch die Konfiguration neu geladen.

Abbildung 22 Kommandos

Logging

Für das Nachverfolgen der Aktivitäten und aufgetretenen Fehler werden die Lognachrichten in Logdateien gespeichert. Erreicht eine Logdatei ihre maximale Grösse, wird eine neue Datei erstellt. Ist die maximale Anzahl von Dateien erreicht, werden die ältesten überschrieben. Die Logdateien werden mit *ngson_{Nummer}.log* benannt. Die Nummer mit dem tiefsten Wert ist dabei immer die Aktuellste.

Die maximale Grösse und Anzahl solcher Logdateien kann über die Konfiguration eingestellt werden.

KONFIGURATION

Die Konfiguration des NGSON-Nodes wird über eine Konfigurationsdatei eingestellt. Diese wird im Hauptverzeichnis vom NGSON-Node abgelegt und heisst *config*. Beim Starten des NGSON-Nodes wird diese Datei, falls sie noch nicht existiert, erstellt. Standardmässig sind alle Schnittstellen deaktiviert. Nach dem automatisierten Erstellen der Datei muss die Konfiguration zuerst angepasst und der NGSON-Node neu gestartet werden. Ansonsten sind die Rest-Ressourcen nicht aktiviert.

Die Einstellungen werden als Schlüssel-Wert Paare gespeichert. Dies sehen folgendermassen aus: `Schlüssel=Wert`. Folgende Einstellungen können gemacht werden:

Schlüsse	Standardwert
activate_interface_composit	false
Legt fest ob die CompositeServiceResource aktiviert werden soll oder nicht.	
activate_interface_end_user	false
Legt fest ob die EndUserResource aktiviert werden soll oder nicht.	
activate_interface_maintenance	false
Legt fest ob die MaintenanceResource aktiviert werden soll oder nicht.	
database_host	localhost
Setzt die Adresse für den Datenbankserver.	
database_name	ngsonServer
Setzt den Namen der Datenbank.	
database_password	ngsonuser
Setzt das Passwort für den Datenbankserver.	
database_port	5432
Setzt den Port für den Datenbankserver.	
database_user	ngsonuser
Setzt den Benutzer für den Datenbankserver.	
max_logfilecount	3
Maximale Anzahl der Logfiles die erstellt werden dürfen. Ist das Maximum erreicht, werden die Ältesten überschrieben.	
max_logfilesize	1000000
Maximale Grösse eines einzelnen Logfiles bevor ein neues erstellt wird.	
rest_service_port	9998
Setzt den Port für die REST Services.	
servername	ngsonNode_{Hostname}
Setzt den Name des NGSON-Nodes.	

TABELLE 37 SETTINGS

VERKETTETE SERVICES

Die verketteten Services werden mittels XML beschrieben. Das XML-Schema kann von der *CompositeServiceResource* über die URL http://{NGSON-Node Adresse}/ngson/application.wadl/xsd0.xsd abgerufen werden.

Ein verketteter Service hat die gleichen Attribute wie ein eigenständiger Service. Zusätzlich dazu besitzt er verschiedene Kompositionen. Diese beschreiben die einzelnen Serviceaufrufe mit den dazugehörigen Parametern und der Position, an der sie in der Verkettung aufgerufen werden. Alle Kompositionen werden anhand ihrer Position sortiert und in aufsteigender Reihenfolge aufgerufen.

Die in den Kompositionen für den Serviceaufruf benötigten Parameter, sind in den Kompositionseigenschaften beschrieben. Für den Aufruf können auch Rückgabewerte von vorherigen Servicesaufrufen oder Kontextparameter verwendet werden. Dies wird in den Parameterbeschreibungen angegeben.
Die Struktur der verketteten Services sieht wie folgt aus:

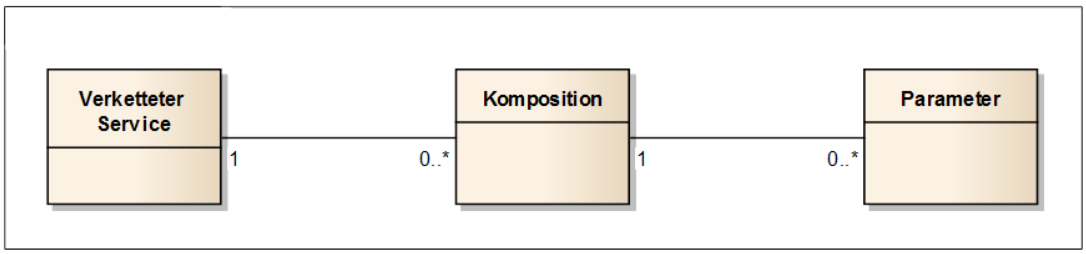


ABBILDUNG 23 STRUKTUR VERKETTETER SERVICE

Die Kompositionen haben folgende Eigenschaften:

compositionTo	
bodyType	Schlüssel des Parameters, welcher den HTTP-Body beschreibt.
httpMethod	HTTP-Methode wie: GET, POST, HEAD, PUT, DELETE, TRACE, OPTIONS oder CONNECT.
parameters	Liste der Parameterbeschreibungen.
position	Position, an welcher der Service in der Verkettung aufgerufen wird.
responseType	Media-Typ des Rückgabewertes des Serviceaufrufs.
url	URL für den Serviceaufruf. Diese kann weitere Parameter enthalten, welche in geschweiften Klammern angegeben werden. Zum Beispiel http://127.0.0.1/photo/convert/{with}/{high}

TABELLE 38 BESCHREIBUNG KOMPOSITION

Die Parameter haben folgende Eigenschaften:

parameterTo	
mediaType	Media-Typ des Parameters
name	Name des Parameters
source	Legt die Herkunft des Parameters fest. Dabei gibt es folgende Möglichkeiten: • **parameter/{Name}**: Name des Parameters, welcher beim Aufruf des Services in URL-enkodierter Form [16] mitgegeben wurde. • **deviceContext/{Property Name}({Default Wert}):** Name des Properties des Gerätekontextes. Für den Fall, dass der Gerätekontext nicht vorhanden ist, muss noch ein Standardwert mitgegeben werden. • **response/{Serviceposition}:** Position des Services, dessen Rückgabewert als Parameter verwendet wird.

TABELLE 39 BESCHREIBUNG PARAMETER

Am untenstehenden Beispiel ist zu sehen, wie eine solche Komposition aussehen könnte.

```
<compositions>
  <bodyType>picList</bodyType>
  <httpMethod>POST</httpMethod>
  <parameters>
    <mediaType>application/xml</mediaType>
    <name>picList</name>
    <source>response/2</source>
  </parameters>
  <position>3</position>
  <responseType>text/plain</responseType>
  <url>http://127.0.0.1:1234/compress</url>
</compositions>
```

Diese Komposition wird an Position drei aufgerufen und hat einen Parameter *picList*, welcher im HTML-Body mitgegeben wird und das Resultat des zweiten Aufrufs ist. Der Aufruf wird mittels der POST Methode gemacht und liefert Text zurück.

4.4.2.6 Client

Funktionsumfang

Der Android Client umfasst folgende zwei Hauptfunktionen:

- Serviceliste herunterladen und anzeigen
- Service ausführen, Resultat entgegennehmen und anzeigen

RESTful Android Applikation

Der NGSON Client ist eine RESTful Android Applikation. Beim Entwickeln solcher Applikationen muss auf das Memory Management und das Mutlithreading von Android geachtet werden. Je nach Architektur können Performance- oder Bedienbarkeitsprobleme auftreten. Um den Aufbau des Clients nachvollziehen zu können, werden anschliessend einige Probleme und eine mögliche Lösung aufgezeigt.

Probleme

Android benutzt ein *Single Thread*[16] Modell [1]. Pro Applikation wird ein Prozess gestartet, in dem alle Komponenten der Applikation laufen. Wie bereits in der Studienarbeit [2] beschrieben, wird der Main-Thread als einziger Thread mit dem Prozess gestartet. Dieser nimmt Benutzereingaben entgegen und führt Operationen auf dem GUI[17] aus. Führt er eine blockierende oder rechenintensive Operation durch, reagiert das GUI während dieser Zeit nicht mehr. Dauert diese Blockade zu lange, erscheint eine ANR-Meldung[18] und die Applikation wird beendet [2]. Mit Hilfe

[16] Single Threaded beschreibt ein Programm, welches immer nur ein Kommando nach dem anderen abarbeiten kann.
[17] Graphical User Interface, kurz GUI, bezeichnet die Benutzeroberfläche einer Applikation.
[18] Application Not Responding (Applikation reagiert nicht mehr)

von Worker-Threads[19], welche die blockierenden bzw. rechenintensiven Operationen ausführen, kann dieses Problem gelöst werden. Doch dies genügt für eine RESTful Android Applikation nicht.

Damit stets genügend Ressourcen vorhanden sind, muss das Android Betriebssystem bei Knappheit geeignete Prozesse beenden und deren belegten Ressourcen freigeben. Die Wichtigkeit für den Benutzer bzw. der Benutzerinteraktion entscheidet, ob ein Prozess beendet wird. Das heisst, Prozesse von Applikationen, die für den Benutzer nicht sichtbar sind und keinen direkten Einfluss auf die momentane Interaktion mit dem Benutzer haben, werden stets beendet. Diese werden bei Android als Hintergrundprozesse (*background process*) bezeichnet [1]. Die folgende Situation soll veranschaulichen, wo dabei das Problem bei einer RESTful Applikation liegt.

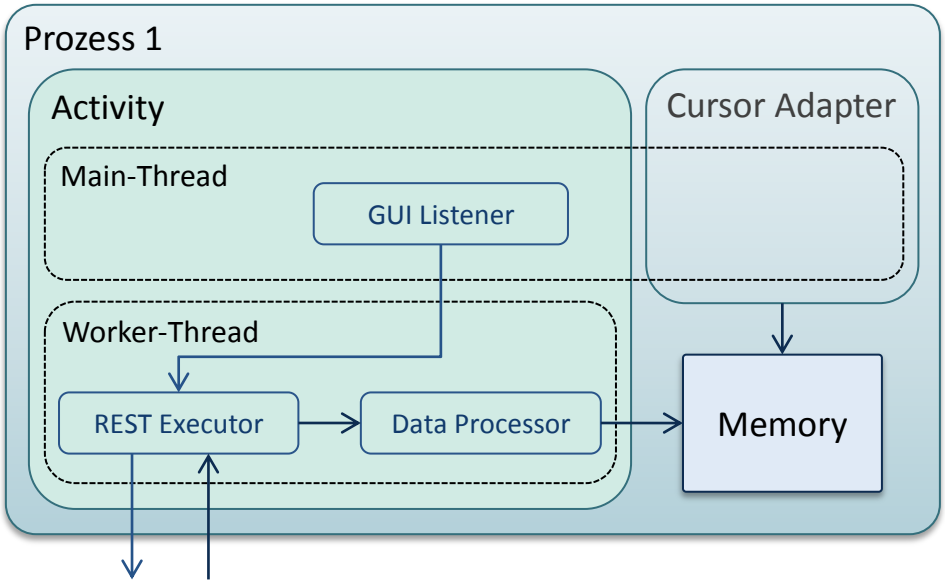


Abbildung 24 Prozess 1 nur mit Activity

In Prozess 1 läuft die *Activity*[20] einer Applikation. Damit das GUI nicht blockiert, übernimmt ein Worker-Thread die zeitintensive und blockierende Netzwerkkommunikation. Ein *GUI Listener* nimmt die Benutzereingabe entgegen und startet den Worker-Thread. Dieser führt den REST-Aufruf aus und wartet auf die Antwort des Servers. Sobald diese eintrifft, verarbeitet ein *Data Processor* die Daten bzw. nimmt diese entgegen und speichert sie in eine Datenstruktur im *Memory* (flüchtiger Speicher) ab. Über den *Cursor Adapter*, der auf diese Datenstruktur im *Memory* zugreift, wird das GUI aktualisiert, sobald die Daten vorhanden sind.

Nun kann es vorkommen, dass während dem REST-Aufruf der Benutzer die Applikation wechselt, zum Beispiel wegen eines eingehenden Anrufs. Dabei wird die *Activity* gestoppt und Prozess 1 zum Hintergrundprozess. Räumt nun Android wegen Ressourcenknappheit diesen Prozess weg, hat dies folgende Auswirkungen:

- Die heruntergeladenen Daten gehen verloren, weil das *Memory* gelöscht wird.
- Der Benutzer muss nach dem Neustart der *Activity* den Rest-Aufruf erneut ausführen.

[19] Ein Worker-Thread ist ein Thread, welche für das Ausführen einer rechenintensiven Aufgabe erstellt wird.
[20] Eine Activity ist ein Tätigkeit, welche der Benutzer mit einer Applikation tu kann. Sie bestimmt also, für was und wie der Benutzer die Applikation nutzen kann.

Ein weiteres Problem bei diesem Ansatz ist, dass es zu Speicherproblemen führen kann, wenn sehr viele Daten heruntergeladen werden und nicht genug flüchtiger Speicher zur Verfügung steht.

Lösung

Eine von Google vorgeschlagenes Lösung [17] besteht aus den Komponenten *Activity*, *Android Service* und *Content Provider* und sieht wie folgt aus:

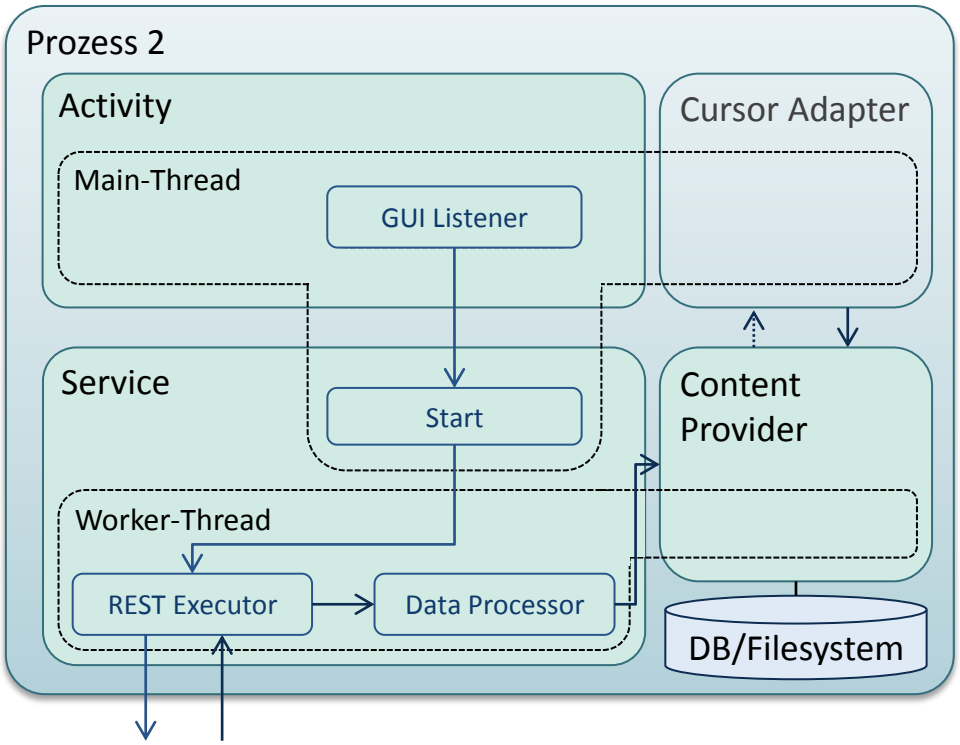


Abbildung 25 Prozess 2 mit Activity und Service

Bei dieser Variante wird ein *Android Service* für die Netzwerk-Operationen verwendet. Da der Main-Thread den Service startet, muss zusätzlich einen Worker-Thread für die Ausführung dieser Netzwerk-Operationen erstellt werden. Die heruntergeladenen Daten werden einem *Content Provider* übergeben, welcher sie in eine Datenbank speichert. Mittels Callbacks wird das GUI über die Änderungen benachrichtigt.

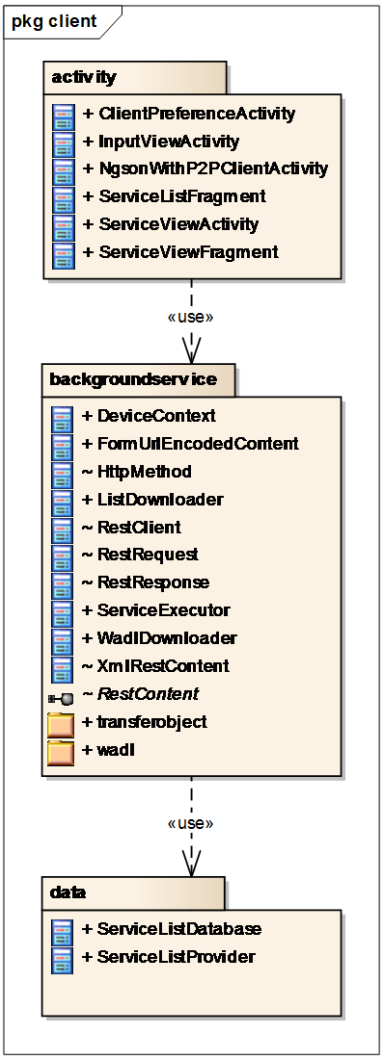
Folgende Vorteile bietet dieser Ansatz:

- Das GUI bleibt dank Worker-Threads reaktionsfähig.
- Beim Stoppen der *Activity* wird der Prozess dank dem Service nicht zum Hintergrund- sondern zum Serviceprozess (*service process*), vorausgesetzt der Service wurde gestartet. Service-Prozesse haben eine höhere Priorität als Hintergrundprozesse und werden wenn möglich von Android nicht beendet [1]. Das heisst, die Daten werden im Hintergrund weiter heruntergeladen und abgespeichert.
- Wird die *Activity* wieder gestartet, können nun die in der Zwischenzeit heruntergeladenen Daten angezeigt werden.
- Das Persistieren in eine Datenbank dient auch als Caching. Ist keine Netzwerkverbindung vorhanden, können einfach früher geladene Daten angezeigt werden.

- Android bietet viel Unterstützung bei der Umsetzung dieser Lösung. Komponenten wie *Cursor Adapter* und *Content Provider* übernehmen die Aktualisierung des GUI. Es müssen nur die Daten über den *Content Provider* gespeichert bzw. geändert werden und das GUI wird automatisch über den Main-Thread aktualisiert [1]. Dank Komponenten wie *Intent Service* oder *AyncTask* muss man sich nicht mehr um das Erstellen und Starten von Worker-Threads kümmern [1].

ARCHITEKTUR

LOGISCHE SICHT



activity

Dieses Paket beinhaltet alle *Activities* sowie *Acitivity*-Fragmente.

backgroundservice

Dieses Paket beinhaltet alle Android Service-Klassen, die mittels Worker-Threads im Hintergrund laufen, und deren Hilfsklassen. Dies sind unteranderem die Transferobjekte für die XML-Serialisierung und Deserialisierung sowie der REST-Client.

data

Dieses Packet beinhaltet die Datenbankdefinition und der Content Provider für den Zugriff auf die Datenbank.

ABBILDUNG 26 LOGISCHE SICHT NGSON CLIENT

Activity, Android Service, Content provider

Die von Google vorgeschlagene Lösung (siehe *Kapitel RESTful Android Applikation*) wird für das Herunterladen der Servicelisten und der WADLs verwendet. Die Liste enthält pro NGSON Service den Name, die Beschreibung und ein Veröffentlichungsdatum. Da die WADLs nur von den vom Benutzer verwendeten Services benötigt werden und einige Kilobytes gross sein können, werden diese erst bei Bedarf nachgeladen.

Für das Herunterladen der Serviceliste kommen die folgenden drei Komponenten zum Einsatz:

- NgsonWithP2PClientActivity (*Activity*)
- ListDownloader (*Android Service*)
- ServiceListProvider (*Content Provider*)

Für das Herunterladen des WADLs kommen die folgenden drei Komponenten zum Einsatz:

- Je nach Bildschirmauflösung ServiceViewActivity bzw. NgsonWithP2PClientActivity (*Activity*)
- WadlDownloader (*Android Service*)
- ServiceListProvider (*Content Provider*)

Für das Ausführen eines NGSON Service und Herunterladen dessen Antwort kommt kein Content Provider zum Einsatz. Die Daten werden direkt auf dem Dateisystem gespeichert. Anschliessend werden diese von einer *Activity* geöffnet. Dabei kommen folgende Komponenten zum Einsatz:

- InputViewActivity (*Activity*)
- ServiceExecutor (*Android Service*)

Bildschirmoptimierung

Fragments sind bei Android GUI-Elemente genannt. Diese können in einer einzigen *Activity* miteinander kombiniert werden. Durch die Verwendung von *Fragments* kann der verfügbare Platz bei verschiedenen Bildschirmauflösungen besser ausgenutzt werden. Bei kleinen Auflösungen werden jeweils nur einzelne *Fragments* auf einmal angezeigt. Bei grösseren Auflösungen können mehrere gleichzeitig nebeneinander angezeigt werden.

Umsetzung REST Client und Serialisierung

Leider bietet Android kein REST Client an, deshalb musste eine Alternative gefunden werden.

Test mit Jersey

Da auf dem Server Jersey zum Einsatz kommt, wurde zuerst überprüft bzw. getestet, ob der Jersey Client auf Android verwendet werden kann. Es stellte sich heraus, dass dies nicht so einfach geht. Das Problem liegt unteranderem daran, das Android nicht den gesamten Umfang von Java SE[21] anbietet, sondern nur die wichtigsten Teile daraus. Somit fehlen Jersey einige Komponenten auf Android.

[21] Java Standard Edition enthält grundlegende Java APIs.

Damit die fehlenden Komponenten benutzt werden können, müssen die entsprechenden JAR-Dateien[22] heruntergeladen und ins Android Projekt eingebunden werden. Das Problem dabei ist, das Android das Hinzufügen eines bereits existierenden Package verbittet. Darum muss ein *Repackaging* mit den heruntergeladenen JARs durchgeführt werden [18]. Dabei hilft das Programm JarJar weiter [19]. Es ermöglicht zum Beispiel das Package javax. auf ngson.javax zu ändern. Anschliessend müssen noch die Jersey Klassen für die Nutzung der neuen Packages angepasst werden. Ist das alles erledigt, kann Jersey verwendet werden. Diese Lösung läuft jedoch nicht stabil und es müssten Änderungen am Source Code von Jersey gemacht werden.

Das Fazit: Jersey ist nicht für Android geeignet, weil der Aufwand für die Verwendung zu gross und das Framework für mobile Plattformen ungeeignet ist. Für eine einfache mobile Client Anwendung sind einige Megabytes an JARs zu viel.

Alternative

Weiter wurde die *Google HTTP Client Library for Java* [20] betrachtet, von welcher eine Android-Version zur Verfügung gestellt wird. Damit lassen sich HTTP-Aufrufe ausführen und Java-Objekte mit XML oder JSON serialisieren. Leider ist das Projekt noch in Entwicklung (Beta-Version erhältlich) und der Code ist noch nicht sehr stabil.

Schlussendlich wurde der *Apache HTTP Client*, welcher von Android zur Verfügung gestellt wird verwendet, obwohl dieser keine XML-Serialisierung und Deserialisierung von Java Objekten zu Verfügung stellt. Deshalb wurde *Simple* für die Objekt-Serialisierung benutzt. Diese Bibliothek läuft ohne zusätzlichen JARs auf Android und passt von der Datengrösse her gut zu mobilen Anwendungen [21].

Service-Liste aktualisieren

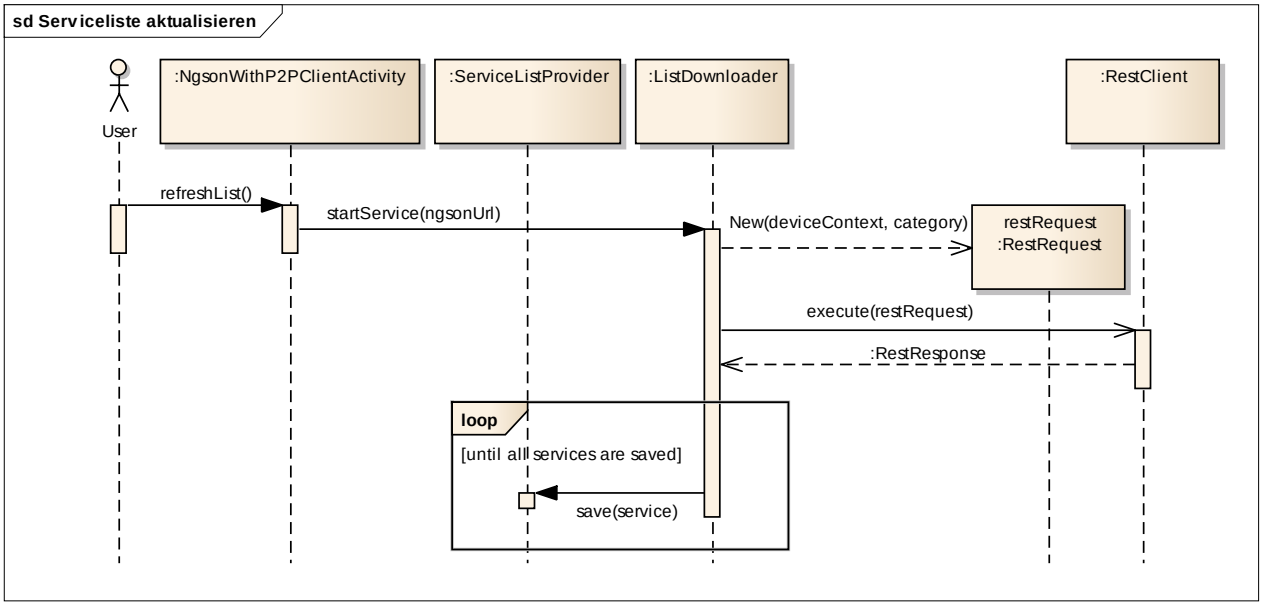


Abbildung 27 Serviceliste herunterladen

[22] Ein Java Archive enthält unteranderem den Java-ByteCode von Klassenbibliotheken und Metadaten.

Um die Liste der Services zu aktualisieren, wird die REST-Ressource *EndUserResource* benutzt. Genaue Details zu dieser Ressource sind im *Kapitel Schnittstellen* zu finden.

Abbildung 27 zeigt den stark vereinfachten Ablauf beim Aktualisieren der Liste. Beim Starten des Services wird die URL, unter welcher der NGSON Node erreichbar ist, mitgegeben. Es handelt sich dabei um einen asynchronen Aufruf. Für die REST-Anfrage der Liste werden der Gerätekontext mit den Geräteinformationen und die Service-Kategorie mitgeschickt. Der NGSON-Node liefert eine Liste mit Servicebeschreibungen in der Rest-Antwort zurück. Die einzelnen Services werden über den *Content Provider* in die Datenbank-Tabelle gespeichert. Das GUI wird dabei automatisch aktualisiert. *Abbildung 28* zeigt wie dieses GUI bei einem Tablet aussieht nachdem die Services geladen wurden.

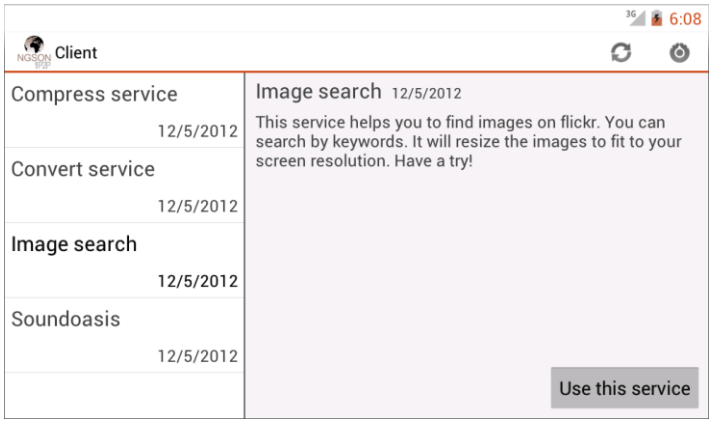
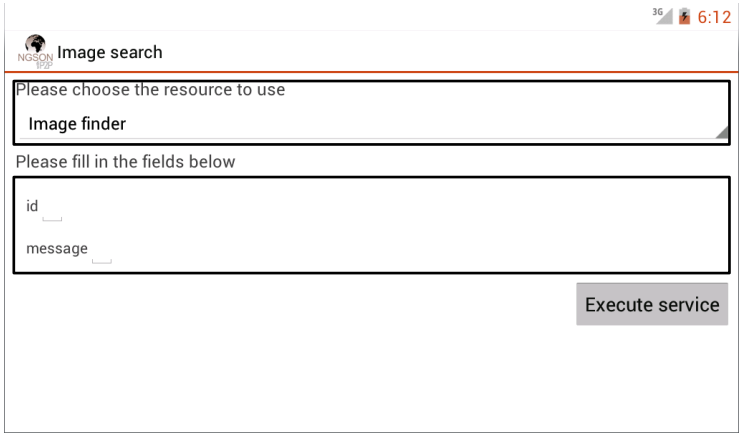


Abbildung 28 Service-Liste und Detailanzeige

Service-Ausführung

Bevor der ausgewählte Service ausgeführt werden kann, muss das dazugehörige WADL heruntergeladen werden. Mit einem *Visitor*[23] werden die im WADL definierten Ressourcen in eine Liste extrahiert. Die einzelnen Ressourcen beschreiben unter welcher URL und mit welchen Parametern sie nutzbar sind. Wie *Abbildung 29* zeigt, werden die einzelnen Ressourcen dem Benutzer zur Auswahl angezeigt (Punkt 1). Sobald dieser eine Ressource ausgewählt hat, werden die Eingabefelder für die vom Service definierten Parameter generiert (Punkt 2). Nachdem der Benutzer die Felder ausgefüllt hat, kann er die Services ausführen lassen.

[23] Visitor Pattern



1. Ressource auswählbar

2. Generierte Eingabefeldern

ABBILDUNG 29 GUI GENERIERUNG

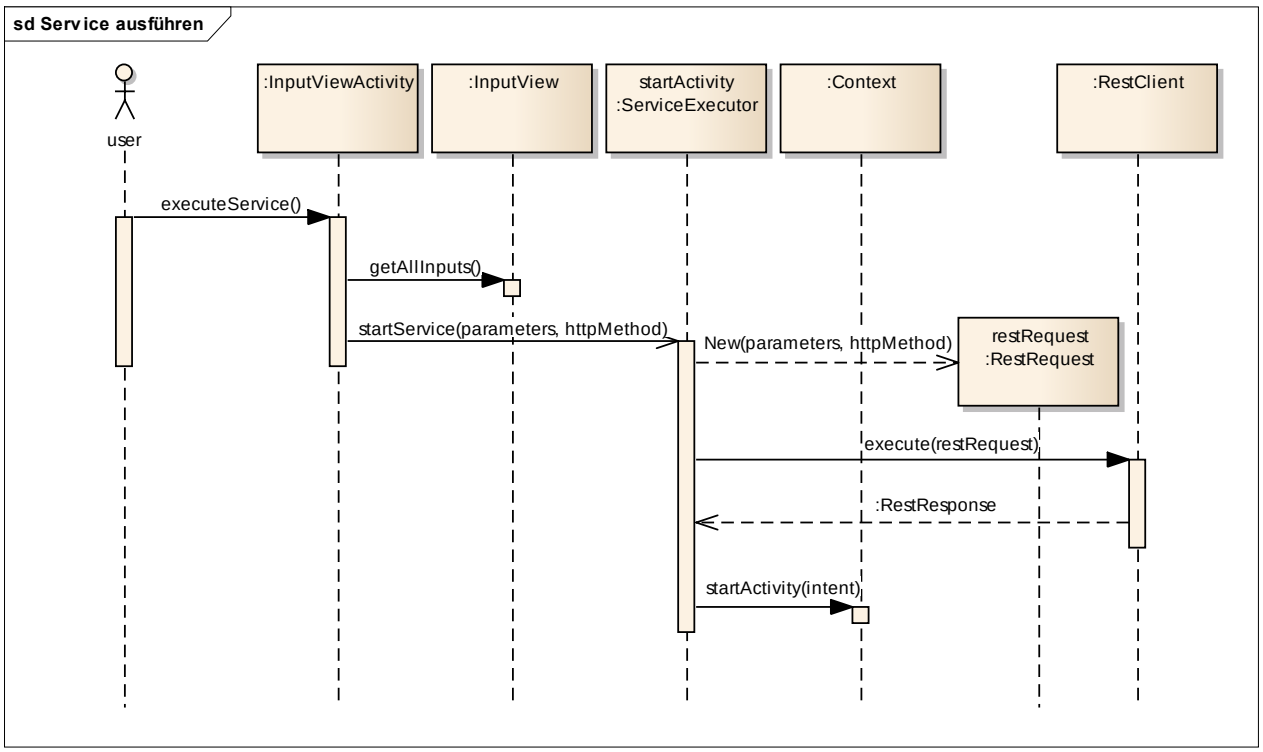


ABBILDUNG 30 SERVICE AUSFÜHREN

Abbildung 30 ist ein stark vereinfachtes Sequenzdiagramm und zeigt, wie der Service ausgeführt wird. Der Unterscheid zum Aktualisieren der Liste ist die Verarbeitung der Rest-Antwort. Dies wird in *Kapitel Service-Ausführung* genauer beschrieben.

WADL PARSEN

Damit *Simple* WADL parsen kann, mussten aus dem XML-Shema des WADL die nötigen Klassen generiert und annotiert werden. Diese befinden sich im Package com.sun.research.ws.wadl.

REST-AUFRUF MIT PARAMETER

Es gibt zwei Möglichkeiten, wie die Parameter den REST-Webservices übergeben werden können. Bei GET geschieht dies über die URL. Bei POST können zusätzlich Eingabedaten im HTTP-Body mitgegeben werden.

Ein REST-Aufruf kann mit folgenden Informationen ausgeführt werden:

- URL mit Parameter
- HTTP-Methode
- Eingabedaten im HTTP-Body

Folgendes Beispiel zeigt, wie diese Daten aus dem WADL entnommen werden.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://wadl.dev.java.net/2009/02">
  …
  <grammars>
    <include href="application.wadl/xsd0.xsd">
      <doc title="Generated" xml:lang="en"/>
    </include>
  </grammars>
  <resources base="http://152.96.235.167:8888/">
    <resource path="/photo">
      …
      <resource path="/convert/{with}/{height}">
        <param name="with" type="xs:int"/>
        <param name="height" type="xs:int"/>
         <method id="convertImage" name="POST">
           <request>
             <representation element="urlList" mediaType="application/xml"/>
           </request>
           <response>
             <representation element="urlList" mediaType="application/xml" />
           </response>
         </method>
      </resource>
      …
    </resource>
  </resources>
</application>
```

1. URL
    a. *resources*-Tag gibt die Basis-URL mit dem *base*-Attribut an (im Beispiel: http://152.96.235.167:8888/).
    b. jeder *resource*-Tag entspricht einer Ressource. Das *path*-Attribut gibt an, unter welchem Pfad die Ressource benutzbar ist. Bei verschachtelten *resource*-Tags müssen diese Pfadangaben zusammengesetzt werden (im Beispiel: /photo/convert/{with}/{height}).
    c. Die Ausdrücke in geschweiften Klammern dienen als Platzhalter für die Parameter. Die *param*-Tags innerhalb des *resource*-Tags definieren diese Parameter. (im Beispiel: *with* und *height* vom Typ Integer).
2. HTTP-Methode
    a. Der *method*-Tag gibt mit dem *name*-Attribut die zu verwendende HTTP-Methode an (im Beispiel: POST)
3. Eingabedaten im HTTP-Body
    a. Innerhalb des *request*-Tag der Methode werden die Eingabedaten definiert.
    b. *Das mediaType*-Attribut des *representation*-Tags gibt den Media-Typ[24] der Eingabedaten an.

Bei XML (*application/xml*) als Media-Typ für die Eingabedaten, wird zusätzlich mit dem *element*-Tag der Name des in einem XML-Shema definierten Elements angegeben (im Beispiel: urlList). Somit

[24] Der *Internet Media Type* bestimmt, wie die Daten im Rumpf einer Nachricht interpretiert werden sollen.

lassen sich komplexere Typen für die Eingabedaten definieren. Im WADL gibt das *href*-Attribut des *include*-Tags den Pfad zum XML-Schema an. In diesem Beispiel sieht dieses wie folgt aus:

```
<?xml version="1.0" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="urlList" type="urlList"/>
  <xs:complexType name="urlList">
    <xs:sequence>
      <xs:element name="url" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Mit diesen Angaben lässt sich eine Klasse generieren, welche diesen komplexeren Typ umsetzt. Somit kann man beim REST-Aufruf ein Objekt dieser Klasse als XML serialisieren und im HTTP-Body mitsenden.

In dieser Arbeit wurde das Parsen des XSD und Generieren der Klassen von komplexeren Typen aus Zeitgründen nicht umgesetzt. Es können alle Service genutzt werden, welche Parameter in der URL oder zusätzlich Eingabedaten vom Media-Typ *application/x-www-form-urlencoded* entgegennehmen. Ein Beispiel eines solchen Services sieht wie folgt aus:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://wadl.dev.java.net/2009/02">
  …
  <resources base="http://www.soundoasis.net/ngson/">
    <resource path="/loremipsum">
      <method id="generateText" name="POST">
        <request>
          <representation mediaType="application/x-www-form-urlencoded">
            <param name="words" type="xs:int"/>
            <param name="paragraphs" type="xs:int"/>
            <param name="title" type="xs:string"/>
          </representation>
        </request>
        <response>
          <representation mediaType="text/plain"/>
        </response>
      </method>
    </resource>
  </resources>
</application>
```

Dieser Dienst nimmt drei Parameter (*words*, *paragraphs* und *title*) in URL-enkodierter Form [16] als Eingabedaten im HTTP-Body an.

Verarbeitung der Service-Antwort

Der Media-Typ der Antwort bestimmt, welche *Activity* gestartet wird. Dieser Typ kann aus dem WADL im *response*-Tag oder aus dem HTTP-Header der Antwort entnommen werden. Da Android das Starten einer *Activity* anhand eines Media-Typs unterstützt, kann dies einfach realisiert werden.

Dazu wird ein *Intent*[25] mit dem entsprechenden Media-Typ dem Android Kontext übergeben. Dieser überprüft dann, ob eine *Activity* den gewünschten Media-Typ unterstützt und startet die *Activity* [1].

Es können eigene wie auch von anderen Applikationen zur Verfügung gestellte *Activities* benutzt werden. Nur wenige Media-Typen müssen vom NGSON Client verarbeitet werden können. Um dem unterstützten Media-Typ einer *Activity* anzugeben, muss im Manifest der Applikation der *Intent-Filter* um diesen Typ ergänzt werden [1]. Liefert ein Service beispielsweise ein Bild mit Media-Typ *application/jpeg*, wird beim *Intent*-Aufruf der Android Picture Viewer gestartet.

Falls Android keine passende *Activity* findet, wird eine Meldung angezeigt. Der Gerätekontext sollte diesen Fall verhindern, weil dieser dem NGSON-Node angibt, welche Media-Typen das Gerät unterstützt. Zum jetzigen Zeitpunkt wird aber beim Erstellen des Gerätekontexts nicht überprüft, ob die angegebenen Media-Typen unterstützt werden. Somit kann dies gegenwärtig nicht verhindert werden.

4.4.2.7 REST SERVICES

Für die Verwendung von NGSON wurden drei REST Services erstellt, welche im NGSON registriert, verkettet und vom Endbenutzer verwendet werden können. Dabei handelt es sich um folgende Services:

Foto Service	
Funktion	Sucht anhand der Suchkriterien die angegebene Anzahl von verfügbaren Fotos auf Flickr und liefert die URLs zu den Fotos in einer Liste zurück.
URL	http://{Server Adresse}:8888/photo/{Such Kriterien}/{Anzahl Fotos}
Befehl:	GET
Rückgabetype:	urlList
WADL:	http://{Server Adresse}:8888/application.wadl
Typenbeschreibung:	http://{Server Adresse}:8888/application.wadl/xsd0.xsd

TABELLE 40 FOTO SERVICE

Foto skalier Service	
Funktion	Skaliert die Fotos welche anhand einer URL-Liste mitgegeben werden, in die angegebene Grösse unter Berücksichtigung der Proportionen. Der Rückgabewert ist wiederum eine Liste der URLs der konvertierten Fotos.
URL	http://{Server Adresse}:8888/photo/convert/{breite}/{höhe}
Befehl:	POST
Parametertype:	urlList
Rückgabetype:	urlList
WADL:	http://{Server Adresse}:8888/application.wadl
Typenbeschreibung:	http://{Server Adresse}:8888/application.wadl/xsd0.xsd

TABELLE 41 FOTO SKALIER SERVICE

[25] Ein Intent ist bei Android eine abstrakte Beschreibung einer Operation, die ausgeführt werden soll.

Komprimier Service	
Funktion	Fügt die angegebenen Dateien zu einem Zip zusammen und liefert als Rückgabewert die URL, mit welcher die Zip-Datei heruntergeladen werden kann.
URL	http://{Server Adresse}:7777/compress
Befehl:	POST
Parametertype:	urlList
Rückgabetype:	String
WADL:	http://{Server Adresse}:7777/application.wadl
Typenbeschreibung.	http://{Server Adresse}:7777/application.wadl/xsd0.xsd

TABELLE 42 KOMPRIMIER SERVICE

Diese Services können auch unabhängig von NGSON verwendet werden.

4.4.2.8 BEISPIEL

Damit die Funktionsweise von NGSON aufgezeigt werden kann, wurde ein konkretes Beispiel erstellt, welches die folgenden Funktionen von NGSON aufgezeigt:

- Zentrales Verwalten von REST Web Services.
- Verketten einzelner Services zu einem neuen Service.
- Kontextbasiertes Ausführen von REST Webservices.

Für den verketteten Service wurden die drei im *Kapitel 4.4.2.7 Rest Services* beschriebenen Services zusammengefügt. So entstand ein Service, welcher Bilder sucht, diese dank des Gerätekontextes auf die Geräteauflösung skaliert und als Zip-Datei zu Verfügung stellt.

Der Ablauf des gesamten Beispiels ist nachfolgend als Systemsequenzdiagramm aufgezeigt.

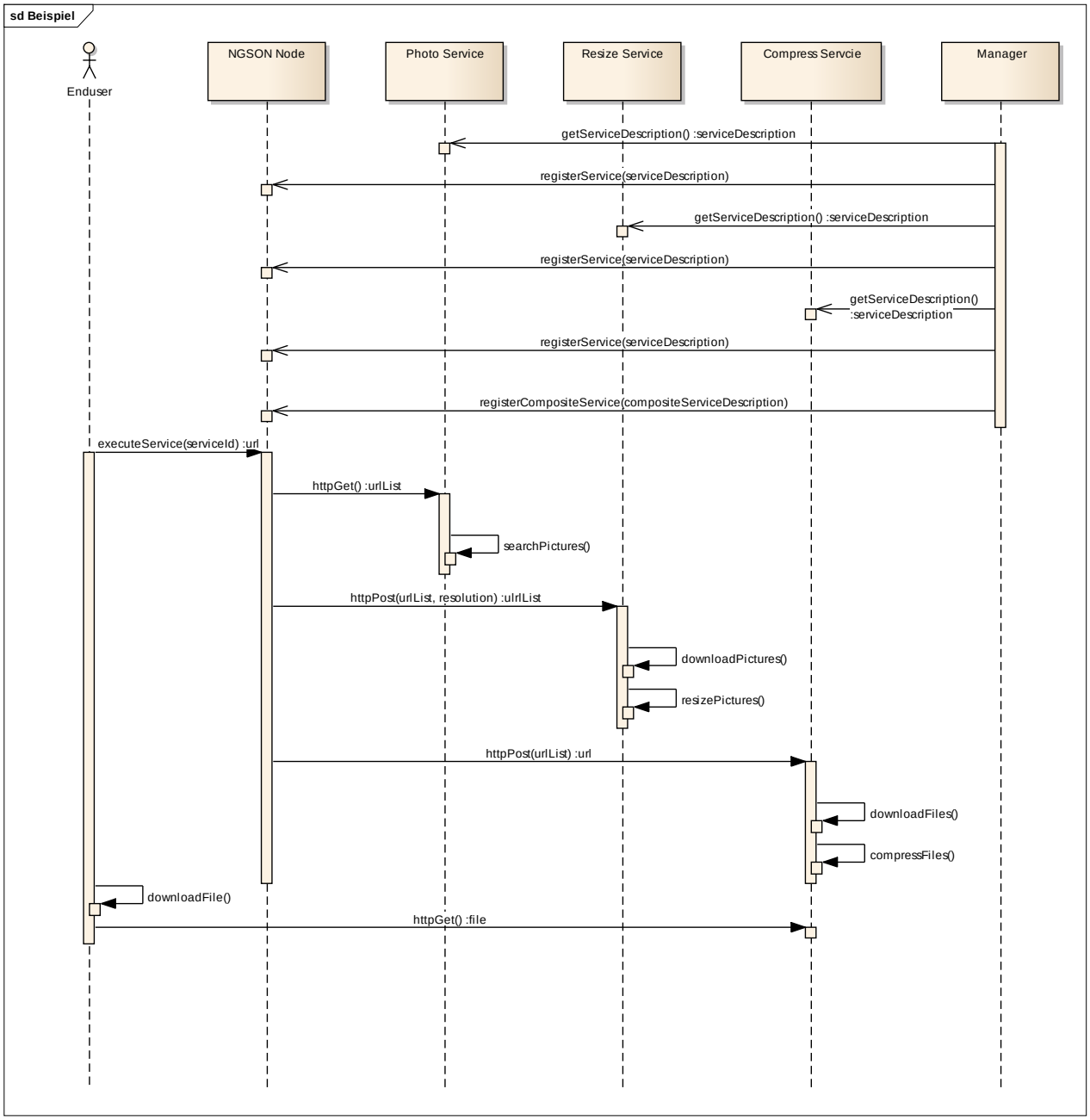


ABBILDUNG 31 SSD BEISPIEL

Als erstes müssen alle REST-Services beim NGSON-Node registriert werden. Das Wichtigste dabei ist die Beschreibung der Services als WADL. Dies kann von den einzelnen Services abgefragt werden.

Anschliessend muss der verkettete Service registriert werden. Dies geschieht über eine XML-Datei. Wie diese aussehen muss, ist im *Kapitel verkettete Services* beschrieben.
Da der verkettete Service nicht direkt auf die registrierten Services verweist, kann dieser auch registriert und verwendet werden, ohne dass vorher die Services registriert wurden.

Als nächstes kann ein Client den verketteten Service auf dem NGSON-Node aufrufen. Der NGSON-Node führt alle Serviceaufrufe nacheinander aus und liefert am Schluss dem Aufrufer die URL für die Zip-Datei. Theoretisch könnte auch direkt die Datei zurückgesendet werden, allerdings müsste sie dann zusätzlich vom NGSON-Node heruntergeladen werden, was so vermieden werden kann.

4.5 Infrastruktur

Ein grundlegender und wichtiger Punkt bei der Bachelorarbeit war es, eine unterstützende Infrastruktur zu verwenden. Dazu gehört ein Versionsverwaltungssystem, ein Continuous Integration Server [26] und ein Projektmanagement-Tool. Diese Produkte wurden bereits in der Studienarbeit eingesetzt und konnten zu Projektbeginn auf einem von der HSR zur Verfügung gestellten Server mit kleinen Anpassungen weiter verwendet werden. Der Server ist über http://www.mobilecloud.ch.vu erreichbar.

4.5.1.1 Versionsverwaltungssystem

Als Versionsverwaltungssystem wurde Git, mit Gitosis zur Benutzerverwaltung verwendet. Git ist ein verteiltes Versionsverwaltungssystem und wird oft als Nachfolger von Subversion[27] bezeichnet. Es ermöglicht ohne direkte Serververbindung *Commits* zu tätigen, oder frühere Versionen zu laden. Die Benutzerverwaltung wurde über Gitosis mit SSH-Keys realisiert, was auf Windows-Clients während der Studienarbeit zu gewissen Problemen führte. Deshalb wurde zusätzlich noch die Möglichkeit geschaffen, mittels eines Passwortes über HTTPS mit Git zu verbinden.

Für die Entwicklung wurden Zweige namens *dev* erstellt. Auf diesen findet die aktuelle Entwicklung statt. Die aktuelle Version kann mit dem Befehl `git checkout dev` verfügbar gemacht werden.

Für Versionen werden Tags erstellt, damit zu jeder Zeit auf diese zurückgegriffen werden kann. Diese können mit dem Befehl `git checkout [Versionsnummer]` verfügbar gemacht werden.

Weil der entwickelte Code nicht für alle frei zugänglich gemacht werden soll, wurde vorerst auf eine Lösung mit Gitorious[28] oder GitHub verzichtet.

Im Verlaufe des Projektes sind folgende Repositorien entstanden:

- **MobileCloud**: Dieses Repository kann mit dem Befehl `git clone git@152.96.56.19:mobilecloud.git` verfügbar gemacht werden.
- **NGSON with Perr-to-Peer**: Dieses Repository kann mit dem Befehl `git clone git@152.96.56.19:ngsonwithp2p.git` verfügbar gemacht werden.
- **NAT Testapplikation**: Dieses Repository kann mit dem Befehl `git clone git@152.96.56.19:mobilecloudnat.git` verfügbar gemacht werden.

4.5.1.2 Continuous Integration Server

Als CI-Server wurde Jenkins verwendet. Jenkins stellt nur die Grundfunktionalitäten zur Verfügung. Durch das Installieren von Add-ons können auch Android Applikationen verwaltet werden. Für die Verwaltung und Überwachung wird ein Webinterface zur Verfügung gestellt.

Die Konfiguration der Builds wurde in einem ANT-File vorgenommen. Dies ermöglicht die Build- und Testvorgänge über Git zu verwalten. Android stellte ein *Build Script* zur Verfügung. Bei diesem

[26] Ein Continuous Integration Server wird in der Informatik für die Automatisierung von Builden, Testen, Erstellen, usw. der Software.
[27] Apache Subversion (SVN) ist eine Server-Client Software zur Versionsverwaltung.
[28] Gitorious oder GitHub sind Webbasierte Plattformen, welche für Open-Source Programme kostenloses Git Hosting anbietet

mussten, für einen optimalen Gebrauch, noch einige Anpassungen und Erweiterungen gemacht werden.
Das *Build Script build.xml* befindet sich im Projekt MobileCloudTest und NgsonWithP2P.

Der CI-Server übernimmt folgende Funktionen:

- Vorbereiten und Starten eines Android Emulators (nur bei MobileCloud)
- Erstellen der Applikation
- Testen der Applikation
- Testabdeckung berechnen und auswerten (nur bei MobileCloud)
- JavaDoc[29] erstellen und publizieren

4.5.1.3 Projektmanagement-Tool

Für das Projektmanagement wurde das freie Tool Redmine verwendet. Redmine ist webbasiert und ermöglicht so wie Jenkins einen Zugriff über den Browser.

Durch den Einsatz von Redmine konnte jederzeit den Projektstand überwacht werden. So wurde auch schnell sichtbar, falls der Zeitplan nicht mehr stimmte und es konnten Massnahmen eingeleitet werden.

Redmine wurde für folgende Aktivitäten verwendet:

- Zeitplanung
- Zeiterfassung
- Projektübersicht
- Reporting

[29] JavaDoc ist ein Dokumentationswerkzeug, das aus Java-Quelltexten HTML-Dokumentationsdateien erstellt

5 SCHLUSSFOLGERUNGEN

5.1 ZUSAMMENFASSUNG

Nachdem in einem ersten Schritt das Interfaces von *MobileCloud* für die Verwendung in Services sowie die Kommunikation über TCP relativ schnell umgesetzt werden konnte, stellte die verteilte Fehlerbehandlung und das Erstellen eines globalen Clusters eine grössere Herausforderung dar. Die verteilte Fehlerbehandlung konnte dank des CAP-Theorems [11] eingegrenzt und mit dem Cooperative Keep-Alive Algorithmus [12] realisiert werden.

Beim Erstellen von globalen Clustern mussten einige Einschränkungen gemacht werden. Durch die Umsetzung von UDP Hole Punching (*Kapitel 4.3.5.4 Hole Punching*) wurde es möglich, Geräte über NAT-Grenzen hinweg zu verbinden. Allerdings funktioniert diese Methode nur mit einfachen NAT-Geräten, wie sie von Heimanwendern genutzt werden, und nicht bei den Mobilfunkanbietern.

Im zweiten Teil dieser Arbeit wurde nach einer kurzen Einarbeitungszeit in den *IEEE 1903 NGSON* Standard [15] das zuvor erweiterte *MobileCloud*-Framework auf Java portiert. Mittels des Jersey-Frameworks wurde anschliessend das Verwalten, kontextbasierte Ausführen und Verketten von Services implementiert. Diese sind durch das Jersey-Framework mittels REST benutzbar. Damit die umgesetzte Funktionalität von NGSON aufgezeigt werden kann, wurden noch drei Services erstellt, welche im NGSON registriert und verkettet werden können. Für das kontextbasierte Verwenden dieser Services wurde zusätzlich noch ein Client auf Android erstellt, mit welchem die Services im NGSON abgefragt und ausgeführt werden können.

Mit dem NGSON-Node, den Beispielservices und dem Android Client ist es gelungen eine Umgebung zu schaffen, welche auf eine einfache und verständliche Weise aufzeigt, was die Vorteile von NGSON sind und wie dies in der Praxis aussehen könnte.

5.2 Projektauswertung

5.2.1 Codeauswertung

Folgende Code Metriken wurden mit dem Tool Structure 101 ermittelt.

Beschreibung	Kompress Service	Foto Service	NGSON Server	NGSON Client	Gesammt
JARs	1	1	1	1	4
Packages	3	4	20	7	34
Klassen	6	7	132	43	188
Zeilen mit Bytecode Instruktionen (NI)	796	684	27000	4820	33300
Zeilen mit Code (LOC)	342	294	12000	2073	14709

Tabelle 43 Code metriken

5.2.2 Zeitauswertung

Die folgende Zeitauswertung bezieht sich auf die aufgewendete Zeit des Teams. Als Vergleich wurde der Durchschnittswert nach ECTS-Punkten angenommen. Eine detailliertere Auswertung ist im *SprintBacklog* im Anhang zu finden.

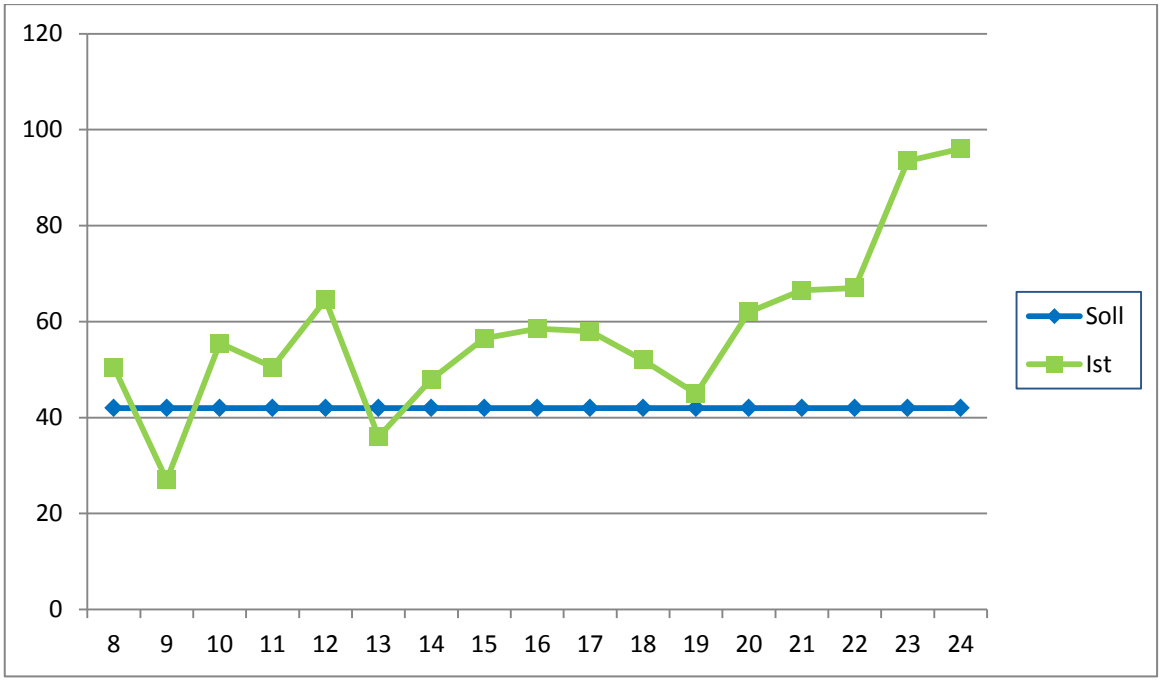
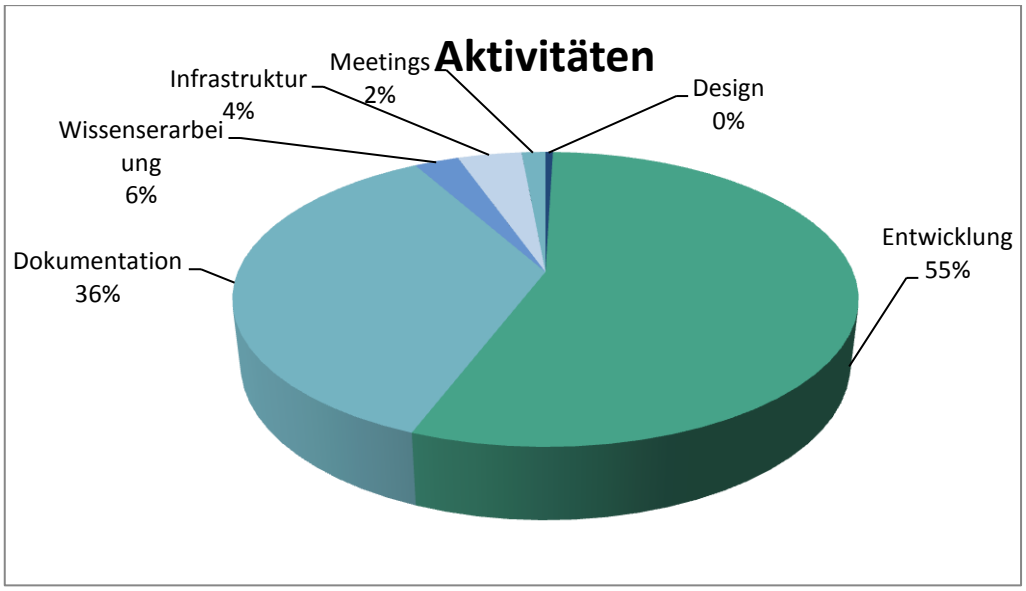


Abbildung 32 Aufgewendete Zeit pro Woche

Die folgende Grafik gibt Auskunft über die Aufteilung der aufgewendeten Zeit. Die grössten Teile nehmen die Entwicklung und die Dokumentation in Anspruch. Für das Design wurde eher weniger Zeit aufgewendet, da viel Design während der Entwicklung entstand und daher in diesen Bereich fällt.



5.3 Ausblick

Der NGSON-Standard ist ein grosser Schritt in die Richtung, dem Benutzer überall und möglichst einfach benutzerspezifische Services anzubieten. Da im verwendeten IEEE 1903 Standard nur der funktionale Umfang spezifiziert ist und die Schnittstellen ziemlich offen sind, ist es noch nicht möglich Anwendungen zu entwickeln, welche mit NGSON vollumfänglich kompatibel sind. Dies sollte aber mit den IEEE-Standards 1903.1-1903.3 möglich werden.

Ein weiterer Punkt ist die Verwendung von BPEL für die Verkettung von Services. Mit PBEL 2.0 wurde ein Standard für Webservices geschaffen, welcher im Bereich der Service orientierten Architektur schon ziemlich verbreitet ist. Somit könnte eine bereits existierende Technologie und BPEL-Engines für die Ausführung von verketteten Services verwendet werden.

Im Bereich der Clients ist abzuklären, ob es sinnvoll ist, einen Client zu entwickeln, welcher alle Services ausführen kann. Es wäre sicherlich benutzerfreundlicher, wenn zum Beispiel ein Mediaplayer Unterstützung für NGSON bieten würde. Mit diesem könnten dann Dienste verwendet werden, welche für den spezifischen Anwendungsbereich ausgelegt sind, in diesem Fall das Betrachten von Videos.

Leider konnten nur einige Funktionen von NGSON genauer betrachtet werden. Mit dem Hinzufügen von Authentisierung, dynamisch verketteten Services oder einer verteilten Verwaltung, könnte das Potential und der Anwendungsbereich von NGSON noch besser aufgezeigt werden. Mögliche Erweiterungen währen:

- Erweitern des NGSON mit mehreren NGSON-Nodes
- Dynamisches Verketten von Services
- Erweitern der Kontexte
- Einführen von Quality of Service
- Suchen von Services in einem NGSON mit mehreren NGSON-Nodes
- Verbindung zu anderen NGSONs
- Verwendung von BEPL

Abschliessend kann zusammengefasst werden, dass während der Bachelorarbeit nur an der Oberfläche des NGSONs gekratzt wurde. Dennoch hat sich dabei schon das grosse Potential dieses neuen Standards gezeigt und eine Weiterführung der Studien ist sicher erfolgsversprechend.

6 Abbildungsverzeichnis

7 TABELLENVERZEICHNIS

8 LITERATURVERZEICHNIS

[1] Android, «developer.android,» Android, 20 Dezember 2011. [Online]. Available: http://developer.android.com. [Zugriff am 20 Dezember 2011].

[2] H. Muralt und M. Steiner, «Studienarbeit MobileCloud,» HSR Hochschule für Technik Rapperswil, Rapperswil, 2011.

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

[4] M. Hadley und Sun Microsystems, Inc, «Web Application Description Language,» 31 August 2009. [Online]. Available: http://www.w3.org/Submission/wadl/. [Zugriff am Mai 2012].

[5] Oracle America, Inc, «Jersey,» 2012. [Online]. Available: http://jersey.java.net/. [Zugriff am 2012].

[6] J. Rosenberg, J. Weinberger, C. Huitema und R. Mahy, «http://www.ietf.org/rfc/rfc3489.txt,» 2003. [Online]. [Zugriff am 20 Dezember 2011].

[7] R. Mahy, p. Matthews, Alcatel-Lucent, J. Rosenberg und jdrosen.net, «Traversal Using Relays around NAT (TURN),» April 2010. [Online]. Available: http://tools.ietf.org/html/rfc5766. [Zugriff am Februar 2012].

[8] P. Srisuresh, Kazeon Systems, B. Ford, M.I.T., D. Kegel und kegel.com, «State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs),» März 2008. [Online]. Available: http://tools.ietf.org/html/rfc5128. [Zugriff am Februar 2012].

[9] J. Rosenberg und jdrosen.net, «Interactive Connectivity Establishment (ICE),» April 2012. [Online]. Available: http://tools.ietf.org/html/rfc5245. [Zugriff am Februar 2012].

[10] B. Ford, P. Srisuresh und D. Kegel, «Peer-to-Peer Communication Across Network Address Translators,» [Online]. Available: http://midcom-p2p.sourceforge.net/.

[11] S. Gilbert and N. A. Lynch, "Perspectives on the CAP Theorem," *IEEE Computer Society,* pp. 30-36, 2012.

[12] I. Dedinski, A. Hofmann und B. Sick, «Cooperative Keep-Alives: An Efficient Outage Detection Algorithm for P2P Overlay Networks,» in *Seventh IEEE International Conference on Peer-to-Peer Computing*, Galway, 2007.

[13] P. Mahlmann und C. Schindelhauer, Peer-to-Peer-Netzwerke Algorithmen und Methoden, Berlin: Springer-Verlag Berlin Heidelberg, 2007.

[14] IEEE Communications Society, «IEEE Standard for the Functional Architecture of Next Generation Service Overlay Networks,» IEEE Communications Society, New York, 2011.

[15] I. o. E. a. E. Engineers, «Next Generation Service Overlay Network (NGSON) Working Group,» 08 Juni 2012. [Online]. Available: http://grouper.ieee.org/groups/ngson/. [Zugriff am 08 Juni 2012].

[16] T. Berners-Lee, W3C/MIT, R. Fielding, Day Software, L. Masinter und Adobe Systems, «Uniform Resource Identifier (URI): Generic Syntax,» Januar 2005. [Online]. Available: http://tools.ietf.org/html/rfc3986. [Zugriff am Mai 2012].

[17] D. Virgil, «Developing Android REST client applications,» Google, 2010. [Online]. Available: http://www.google.com/events/io/2010/sessions/developing-RESTful-android-apps.html. [Zugriff am April 2012].

[18] Team Android VM, «Including additional javax.* packages in your Android,» Januar 2010. [Online]. Available: http://code.google.com/p/dalvik/wiki/JavaxPackages. [Zugriff am April 2012].

[19] C. Nokleberg, «JarJar,» Mai 2012. [Online]. Available: http://code.google.com/p/jarjar/wiki/GettingStarted. [Zugriff am Mai 2012].

[20] Google, «Google HTTP Client Library for Java,» Mai 2012. [Online]. Available: http://code.google.com/p/google-http-java-client/. [Zugriff am Mai 2012].

[21] Simple, «Simple, XML Serialization,» Mai 2012. [Online]. Available: http://simple.sourceforge.net/. [Zugriff am Mai 2012].

9 ANHANG

Dokument 1 Projektplan

NGSON with Peer-to-Peer

Projektplan

Änderungen

Autor	Beschreibung	Datum	Version
hmu	Dokument erstellt	20.02.2012	0.1.0
hmu	Erste Kapitel geschrieben	23.02.2012	0.1.1
hmu	Kap. 5-8 geschrieben	24.02.2012	0.1.2
hmu	Meilensteine, Korrekturen.	25.02.2012	0.1.3
mst	Risiken eingeführt	29.02.2012	0.1.4
hmu, mas	Review	14.06.2012	1.0.0

Mitgeltende/Übergeordnete Dokumente

Dokumentname	Beschreibung
ProductBacklog.xlsx	Scrum Product Backlog für alle Features/Bugs
SprintBacklog.docx	Details zu den einzelnen Sprints mit Auswertung
CodeReview.docx	Auswertung der Code Reviews
CodeGuidelines.docx	Code Richtlinien

INHALT

TABELLEN

1 EINFÜHRUNG

1.1 ZWECK

Dieses Dokument gibt eine Projektübersicht der Bachelorarbeit NGSON with Peer-to-Peer an der Hochschule für Technik Rapperswil. Es beschreibt die Ziele der Arbeit sowie der Ablauf, die Organisation und weitere Projektmanagement spezifische Informationen.

1.2 GÜLTIGKEIT

Die Gültigkeit dieses Dokumentes bezieht sich auf die Dauer der Bachelorarbeit NGSON with Peer-to-Peer während dem Frühlingssemester 2012. In dieser Zeit können jederzeit Änderung vorgenommen werden.
Das Dokument kann basierend auf dem aktuellen Projektstand als gültig betrachtet werden.

2 PROJEKTÜBERSICHT

2.1 VISION

Mobile Geräte (iPhones, iPads, Android basierte Geräte) sind sehr stark verbreitet und verfügen häufig über ungenutzte oder ungenügend genutzte Fähigkeiten.

Falls die mobilen Geräte miteinander/gemeinsam Aufgaben lösen würden/könnten, hätte man in fast jeder Gruppe von mobile Usern ein enormes Rechenpotential zur Verfügung. Heute stecken viele dieser Geräte arbeitslos in irgendeiner Tasche.

Ansätze welche in dieselbe Richtung gehen, jedoch nur einzelne Basisdienste zu Verfügung stellen sind:

- Viber: Telefonie (auf iPhone und Android)
- Skype: Telefonie (iPhone, Android, Windows Phone)
- WhatsApp Messenger: Nachrichtendienst (iPhone, Android, Windows Phone)

Das Ziel dieser Bachelorarbeit ist es in einem ersten Schritt mit den Erkenntnissen aus der Semesterarbeit MobileCloud das Framework bezüglich Fehlertoleranz, Sicherheit und Performance auszubauen.

In einem zweiten Schritt soll mit dem Framework eine konkrete Anwendung entstehen. Der genaue Umfang dieser Anwendung wird in der Woche acht festgelegt.

2.2 AUSGANGSLAGE

Diese Bachelorarbeit setzt auf der Semesterarbeit MobileCloud auf, welche im Herbstsemester 2011/2012 von Heinrich Muralt und Marcel Steiner bearbeitet wurde. Das Ziel der Semesterarbeit war es zu untersuchen, wie mobile Geräte mit den neusten Technologien zu einem Netz verbunden werden können.

2.2.1 SEMESTERARBEIT MOBILECLOUD

In der Semesterarbeit MobileCloud wurde untersucht wie mobile Geräte vernetzt werden können und anschliessend in einen Prototyp umgesetzt. Dabei wurde schlussendlich ein P2P[1]-Ansatz gewählt, welcher auf dem Betriebssystem Android implementiert wurde.

Die Semesterarbeit wurde in zwei Phasen unterteilt:

Phase 1: Es wurde untersucht, ob es grundsätzlich möglich ist, mobile Geräte in einem P2P - Netzwerk zu verbinden. Dabei wurde noch ein zentralisierter Ansatz mit einem Server verwendet. Der entstandene Prototype ist im Source Verwaltungstool Git [2]mit dem Tag 0.1.0 gekennzeichnet und kann mittels des Befehles `git checkout 0.1.0` verfügbar gemacht werden.

[1] Peer-to-Peer (P2P) beschreibt ein Computernetzwerk, in welchem alle Teilnehmer gleichberechtigt sind und Dienste anbieten, sowie auch in Anspruch nehmen können.
[2] Git ist eine freie Software zur verteilten Versionsverwaltung von Daten.

Phase 1: Der gewählte P2P-Ansatz wurde in einem konkreten Beispiel umgesetzt. Dabei entstand ein Framework, welches die Grundfunktionalitäten eines P2P-Netzwerkes implementiert. Auf diesem Framework wurde noch ein Nachrichtendienst implementiert, welcher es ermöglicht Textnachrichten unter den einzelnen Geräten auszutauschen. Der dabei entstandene Prototype ist im Source Verwaltungstool Git mit dem Tag 0.2.0 gekennzeichnet und kann mittels des Befehles `git checkout 0.2.0` verfügbar gemacht werden.

Bei der Semesterarbeit wurde weder auf Performance noch auf Fehlertoleranz oder Sicherheit geschaut, sondern lediglich auf die Machbarkeit.

2.2.2 Grundlage

Als Grundlage für die Bachelorarbeit wird der Source Code der Semesterarbeit mit der Version 0.2.1 verwendet. Der bestehende Code kann uneingeschränkt abgeändert und erweitert werden. Das Ziel ist jedoch der Ausbau und nicht der Umbau des Frameworks.

Welche Änderungen und Erweiterungen genau gemacht werden, ist im Dokument *SprintBacklog.docx* für jeden Sprint genau aufgeführt.

3 PROJEKTORGANISATION

3.1 DURCHFÜHRUNG

Dieses Projekt wird von 2 Mitgliedern durchgeführt und von Herrn Prof. Dr. Josef M. Joller betreut. Alle Entscheidungen beruhen auf gemeinsame Beschlüsse und wenn für nötig empfunden, nach Absprache mit dem Betreuer.

Die bestehende Software soll schrittweise in festen Intervallen mit neuen Features erweitert werden. Nach jedem Intervall wird jeweils der Stand der Arbeit geprüft und die weiteren Ziele und das weitere Vorgehen festgelegt. In Besprechungen mit dem Betreuer sollen die in den Entwicklungsintervallen implementierten Erweiterungen angeschaut und abgenommen werden. Dazu eignet sich Scrum[3] als Vorgehensmodell. Als Leitfaden dienen die oben erwähnten Vision und Ziele. Genauere Details zu den Zielen beziehungsweise Features sind dem Product Backlog zu entnehmen.

Mit dem Betreuer finden in der Regel wöchentliche Besprechungen statt. Zusätzliche Besprechungen sind nach Bedarf durch die Studierenden (Teammitglieder) zu veranlassen. Das Sitzungsintervall ist flexibel: falls wenig oder keine Probleme auftreten, können die Sitzungsintervalle vergrössert werden. Falls Probleme auftauchen, werden die Intervalle verkleinert.

Folgende feste Arbeitszeiten pro Woche sind für die Arbeit an diesem Projekt geplant:

Montag: 13^{00}-17^{00}

Donnerstag: 08^{00}-17^{00}

Freitag: 08^{00}-17^{00}

Generell wird in dieser Zeit an den von der Schule zur Verfügung gestellten Arbeitsplätze gearbeitet. Bis auf oben erwähnte Zeiten, kann jedes Mitglied selbst seine Arbeitszeiten einteilen, um durchschnittlich 24 Arbeitsstunden pro Woche zu erreichen.

3.2 ORGANIGRAM

3.2.1 ROLLEN

Das Entwicklungsteam besteht aus Heinrich Muralt und Marcel Steiner Steiner. Stakeholder ist Prof. Dr. Josef M. Joller. Die Rollen Product Owner und ScrumMaster sind nicht zugeordnet, da es sich um ein kleines Team handelt und die Zuordnung nicht klar definiert werden kann.

Users sind Android Apps Entwickler.

[3] Scrum ist ein Vorgehensmodell mit einem empirischen, inkrementellen und iterativen Ansatz

3.3 Externe Schnittstellen

Name	E-Mail Adresse	Funktion
Prof. Dr. Josef M. Joller	jjoller@hsr.ch	Betreuer
Prof. Hansjörg Huser	hhuser@hsr.ch	Gegenleser
Matthias Lips	matthias.lips@medidata.ch	Experte

Tabelle 1 Externe Schnittstelle

4 Management Abläufe

4.1 Projektplan

4.1.1 Planung

Wie bereits erwähnt wird Scrum als Vorgehensmodell verwendet. Es wird daher wie in diesem Vorgehensmodell üblich eine Planung pro Sprint erstellt. Diese Planung wird vor dem Start des Sprints im Projektmanagement- und Planungstool Redmine festgehalten.

Der erste Sprint (Intervall) dauert eine Woche und dient für das Einrichten der Arbeitsplätze und des Servers, die Einarbeitung sowie der Projektplanung. Alle weiteren Sprints dauern jeweils 2 Wochen. Die letzte Woche dient als Reserve. Dies ergibt folgende Planung für die Sprints:

Sprints	Start	Ende
Sprint 0	20.02.2012	26.02.2012
Sprint 1	27.02.2012	11.03.2012
Sprint 2	12.03.2012	25.03.2012
Sprint 3	26.03.2012	08.04.2012
Sprint 4	09.04.2012	22.04.2012
Sprint 5	23.04.2012	06.05.2012
Sprint 6	07.05.2012	20.05.2012
Sprint 7	21.05.2012	03.06.2012
Sprint 8	04.06.2012	15.06.2012

Tabelle 2 Sprints

Die Versionsnummern werden wie folgt hochgezählt:

Versionsnummer	Bedeutung
0.0.*	*= optional, z.B. für Bugfix innerhalb von Sprintversionen.
0.*.0	*= Sprintversionen: wird nach einem Sprint um eins erhöht. Steigt die Release Version, wird diese wieder auf 0 gesetzt.
*.0.0	*= Release Version: Wird bei jedem Release um eins erhöht.

Tabelle 3 Versionsnummer

4.1.2 Product Backlog

Die Planung der einzelnen Sprints beruht auf dem Product Backlog. Im Excel-Sheet *ProductBacklog.xlsx* werden alle Features und Bugs als User Storys mit jeweils einer Priorität und Schätzung des Zeitaufwands erfasst. Es werden alle möglichen Features und Wünsche eingetragen.

4.1.3 Sprint Backlog

Pro Sprints werden User Storys aus dem Product Backlog entnommen. Dabei wird die Priorität und Aufwandschätzung berücksichtigt. Die Beschreibung der einzelnen Sprints und deren Auswertung sind im Dokument *SprintBacklog.docx* zu finden.

Die Planungserstellung und Sitzungen werden nicht im Backlog eingetragen. Diese regelmässigen Arbeiten werden direkt im Redmine für jeden Sprint eingetragen. Aus der Planungserstellung resultiert der Sprint Backlog. Das Sitzungsprotokoll enthält die Ergebnisse der Sitzungen.

4.1.4 Arbeitspakete

Die einzelnen Sprints werden als Versionen und die pro Sprint gewählten User Storys als Ticket erfasst. Ein Ticket entspricht einem Arbeitspaket und wird von den Teammitgliedern abgearbeitet. Am Ende jedes Sprints werden die neuen Tickets für den nächsten Sprint eingetragen.

Aus dem Product Backlog können Storycards ausgedruckt werden. Diese enthalten die User Story mit der Aufwandschätzung und Priorität. Jedes Teammitglied kann Karten auswählen und abarbeiten.

4.1.5 Meilensteine

Diese Meilensteine sind bis auf MS3 vorgegeben. MS3 ist der Zeitpunkt, bei welchem auf die Implementation der Beispielanwendung gewechselt wird und MobileCloud möglichst stabil, schnell und sicher laufen muss.

Meilenstein	Inhalt/Lieferobjekte	Zeitpunkt	Sprint
MS1	Projektplan und erste Backlog-Einträge erstellt	26.02.2012	0
MS3	Konzept für Server-Anbietung definiert	22.04.2012	4
MS4	Alle Features implementiert	27.05.2012	7
MS5	Abstract und A0-Poster zur Prüfung an Betreuer	08.06.2012	8
MS6	Bericht an Betreuer, finales A0-Posters	15.06.2012	8

5 Risiko Management

Es können unterschiedliche Risiken im Verlauf des Projekts auftreten. Diese wurden in zwei Gruppen unterteilt:

1. Generelle Risiken, die sich nicht spezifisch aufs Projekt beziehen jedoch vorkommen können.
2. Projektspezifische Risiken mit deren Auswirkung und Vermeidung beziehungsweise Lösung.

5.1 Generelle Risiken

- Ausfall einer Arbeitsstation
  - Es sind 2 Arbeitsstationen und 2 Laptops mit der nötigen Entwicklungsumgebung vorhanden. Beim Ausfall einer Arbeitsstation kann auf einen Laptop ausgewichen werden.
- Ausfall des Git-Servers
  - Die Repositories können lokal ausgetauscht werden.
- Ausfall der Netzwerkinfrastruktur
  - Alternative Austauschmöglichkeiten bereithalten (z.B. externe Festplatte).
- Ausfall Handys
  - Mit Emulatoren und PC-Version (Java Client) weiterarbeiten. Ersatz auftreiben.

5.2 Projektspezifische Risiken

Risk ID	Risiko	Auswirkung	Massnahme	Kosten der Massnahmen in Stunden	Max. Schaden in Stunden	Wahrschein-lichkeit des Eintreffens	Gewichteter Schaden in Stunden	Priorität
R01	Ziele mit gewählter Richtung/Technologie nicht erreichbar	Projekt kann nicht wie geplant fortgesetzt/beendet werden. Es entsteht ein Mehraufwand durch das Wechseln/ändern der Richtung/Technologie.	Wöchentliche Besprechungen mit dem Betreuer (Expert). Durch Scrum ist ein schnelles Feedback garantiert.	**15**	90	10%	**9**	Hoch
R02	Zu kleiner Zeitrahmen für die Implementation aller Ziele (Haupt- und Unterziele)	Nicht alle Ziele können implementiert werden. Es entsteht ein Mehraufwand durch das Nachtragen der Ziele.	Ziele werden bei jedem Sprint neu bewertet und festgelegt. Dass ermöglicht eine wiederkehrende Gewichtung der einzelnen Funktionalitäten	**5**	90	15%	**14**	Hoch
R03	Probleme mit CIS (Continuous Integration Server)	Mehraufwand für das Testen, Bilden, etc.	Backup der wichtigen Daten auf dem Server erstellen.	**10**	30	5%	**2**	Mittel
R04	Datenverlust	Mühsame Datenwiederherstellung.	Regelmässiges Backup der Daten, alle Daten unter Versionskontrolle stellen. Backup Recovery testen.	**6**	480	1%	**5**	Mittel
R05	Ausfall des HSR WLAN	Bestimmte Funktionalitäten können nicht mehr weiterentwickelt werden.	Eigener Access Point bereithalten.	**2**	30	1%	**0**	Niedrig
R06	NAT Traversierung bei den Telefonanbietern funktioniert nicht.	Es muss eine andere Lösung für die Kommunikation gefunden werden, z.B. C2Dm, JGroup.	Frühes abklären der genauen NAT Typen bei den Mobilfunkbetreibern.	**20**	150	5%	**8**	Mittel

Total Kosten in Arbeitspaketen enthalten	**58**	870			
Total Rückstellungen				**38**	

6 INFRASTRUKTUR

6.1 ENTWICKLUNGSTOOLS

6.1.1 HARDWARE

Hardware	Beschreibung
Arbeitsstationen	Für die Entwicklung werden die Arbeitsstationen im Raum 1.262 Nr. 24 und 25 verwendet.
Laptop	Unterwegs oder privat werden die eigenen Laptops verwendet.
Server	Für das Redmine, Jenkins und Git wird von der HSR ein virtueller Linux Server zur Verfügung gestellt.
Handys	Für das Testen auf mobile Geräte werden 2 von der HSR zur Verfügung gestellte Android Handys benutzt.

TABELLE 4 HARDWARE- ENTWICKLUNGSTOOLS

6.1.2 SOFTWARE

Software	Einsatzbereich	Link
Apache Server	Wird für den http Zugriff für Jenkins und Redmine verwendet.	http://httpd.apache.org/
Eclipse mit Android SDK	Wird als Entwicklungsumgebung verwendet.	http://www.eclipse.org http://developer.android.com/sdk/index.html
Enterprise Architekt	Enterprise Architekt wird für das Erstellen von UML-Diagrammen verwendet.	http://www.sparxsystems.com/
FindBugs	Wird für die Fehlersuche im Code verwendet.	http://findbugs.sourceforge.net
Git	Wird für die Versionskontrolle aller digitalen Daten verwendet. Gleichzeitig wird die Versionierung der Software mittels Tags in Git festgehalten.	http://git-scm.com
Jenkins	Wird als Integration Server verwendet.	http://jenkins-ci.org/
Microsoft Skydrive	Wird für die Versionierung und parallele Bearbeitung von Microsoft Office Dokumenten verwendet. Ende jeder Iteration werden diese im Git aktualisiert.	https://skydrive.live.com
Redmine	Wird für die Zeiterfassung, die Iterationsplanung sowie für das Bugtracking verwendet.	http://www.redmine.org/
Ubuntu 12.04	Wird als Betriebsystem für die Entwicklung und den Server verwendet.	http://www.ubuntu.com/

TABELLE 5 SOFTWARE- ENTWICKLUNGSTOOLS

7 Qualitätsmassnahmen

7.1 Allgemein

7.1.1 Versionsverwaltungssystem

Für die Versionierung und Verwaltung aller Projekt-relevanten Dateien wird Git eingesetzt. Die Entwicklung findet auf dem dev[4] Zweig statt. Nach jedem Synchronisieren mit dem Server testet der Jenkins-Server den neuen Code, überprüft die Testabdeckung und erstellt das Javadoc neu.

Jede erstellte Version, wird mit einem Tag gekennzeichnet. Tritt bei dieser Version ein Fehler auf, wird vom Tag aus ein neuer Zweig erzeug um diesen zu beheben. Nach dem Beheben wird der HEAD mit der neuen Version getagt und der Zweig wieder gelöscht.

Für die Verwaltung von Office Dokumenten wird in erster Linie MS Windows Live Skydrive eingesetzt. Damit ist gleichzeitiges Bearbeiten desselben Dokuments möglich. Für die Verwaltung der Dokumente, wurde ein separates Git-Repository angelegt.

7.1.2 Projektmanagement-Tool und Bugtracker

Die gesamte Verwaltung von Arbeitspaketen und das Bugtracking sowie die Zeiterfassung laufen über das webbasierten Projektmanagement-Tool Redmine (www.mobilecloud.ch.vu). Die Arbeitspakete können leicht verteilt und für die Erfassung der Arbeitszeiten benutzt werden. Entdeckte Bugs können ebenfalls eingetragen und dem Verantwortlichen zugewiesen werden. Die Übersichten über die voranstehende und erledigte Arbeit und das Bugtracking ergeben stets einen guten Einblick auf den Stand des Projektes.

7.1.3 Projektautomation

Die Projektautomation beinhaltet automatische Builds, Unit-Tests und Code testabdeckungs-Analysen, welche ausgeführt werden, sobald eine neue Version auf dem Git-Server verfügbar ist. Dafür wird ein Jenkins Server eingesetzt (www.mobilecloud.ch.vu/jenkins).

7.1.4 Dokumentation

Für die Dokumentation werden eigene Formatvorlagen verwendet. Die Dokumentation wird fortlaufend weitergeführt, damit sie möglichst aktuell ist. Bei jeder Änderung führt der Bearbeitende im geänderten Dokument die Versionsnummer nach und schreibt eine kurze Änderungsbeschreibung mit seinem Kürzel. Alle Dokumente werden von beiden Projektteam-Mitgliedern durchgelesen und auf Fehler überprüft.

7.1.5 Teamsitzungen

Unter den Projektteam-Mitgliedern finden fortlaufend Besprechungen über den Stand der Arbeit, Reviews und die weitere Planung statt. Diese werden nicht protokolliert.

[4] dev steht für „development“

8 Software-Qualität

8.1 Code-Qualität

8.1.1 Allgemein

Der Code wird von den 2 Teammitgliedern nach jedem Sprint überprüft. Die Resultate der Reviews werden im Dokument *CodeReview.docx* festgehalten.

Die Code-Guidelines sind im separaten Dokument *CodeGuidelines.docx* zu finden.

Für die Automation der Test und Überprüfung der Testabdeckung kommt, wie bereits erwähnt, ein CI-Server zum Einsatz.

8.2 Testing

8.2.1 Unit-Tests

Die Module, Klassen und Methoden werden mit Hilfe der Unit-Tests getestet. Diese Tests werden vor dem Ausprogrammieren der Klassen definiert. Sie werden von den Projektteilnehmern während der Entwicklung manuell und nach einem Commit auf dem Server automatisch ausgeführt. Diese Tests zeigen an, ob der Code lauffähig ist und ins Haupt-Repository abgelegt werden kann. Es wird eine Testabdeckung von 60 - 70% angestrebt.