MISS
Mac-based Identification and Signaling Service

Fabian Schwab (fabian.schwab@uni-ulm.de)

13. Oktober 2014

Kapitel 1

Einleitung

Um sich den Alltag zu erleichtern, werden immer mehr kleine Aufgaben auf das Smartphone bzw. Tablet ausgelagert, wodurch diese Geräte zu unseren ständigen Begleitern werden. Der auf einem Smartphone genutzte Kalender enthält beispielsweise alle Termine und wichtige Aufgaben, welche an bestimmten Tagen zu gewissen Zeiten zu erledigen sind. Einfache Funktionen innerhalb dieser Kalenderanwendungen erlauben es, individuelle Benachrichtigungen für jeden Termin einzustellen, welche den Nutzer durch Audio-, Visuelle- oder Audiovisuelle-Ereignisse an das bestimmtes Ereignis erinnern sollen.
Doch sind zeitabhängige Erinnerungen nicht immer praktisch für den Nutzer. In manchen Situationen lässt sich nicht genau festlegen, wann eine Aufgabe zu erledigen ist.
Soll die Aufgabe zum Beispiel nach der Arbeit erledigt werden, kann man dies natürlich als zeitabhängiges Event einstellen. Verlässt man aber die Arbeit nicht zur gewohnten oder gedachten Zeit, kann diese Erinnerung zu spät erscheinen. Wird die Erinnerung zu früh ausgelöst kann es sein, dass der Nutzer sie vergisst. Hierdurch entstanden viele Anwendungen, welche auch ortsabhängige Erinnerungen durchführen können. So lassen sich beim Verlassen oder Ankommen an voreingestellten Orten Erinnerungen auslösen. Zum Beispiel wäre es so möglich einzustellen, dass beim Verlassen der Arbeit, eine Erinnerung ausgelöst wird.
Damit lassen sich schon eine Vielzahl an möglichen Szenarien abdecken. Doch gibt es noch einen großen Teil, an Aufgaben, die weder Zeit noch Ortsgebunden bestimmt werden können. Dies trifft vor allem auf Szenarien im Zusammenhang mit Personen zu.
Es lässt sich schwer vorausahnen wann und wo man eine Person antrifft und damit entfallen orts- und zeitgesteuerte Erinnerungen. Einfache Notizen erscheinen hier noch das beste Mittel der Wahl zu sein. Doch an diese muss sich der Nutzer selbst erinnern. Dabei spielt der zeitliche Abstand der Notiz zum erneuten Treffen einer Person eine große Rolle. Je länger es her ist, desto größer ist die Wahrscheinlichkeit, dass der Nutzer nicht mehr daran denkt.
Bei all diesen Szenarien wäre eine personenbezogene Erinnerung von Vorteil. Ein möglicher Ansatz, wie dies zu realisieren ist, wird in dieser Arbeit erleutert.

Um eine Person maschinell zu erkennen und identifizieren, benötigt man einen einfachen Identifikator. Idealerweise ist dieser bereits digital und muss nicht aus analogen Werten, wie einer Geschichtserkennung oder ähnlichem gewonnen werden. Wie zu beginn erwähnt, sind Smartphones und Tablets ständig griffbereit und ein allgegenwärtiger Begleiter. Da diese Geräte meist ohne Unterbrechung mit dem Internet verbunden sind, liegt die Lösung nahe, den Aufenthaltsort aller Nutzer auf einem zentralen Server zu speichern. Doch diese Lösung würde einige Probleme mit sich bringen.
Unter anderem wäre das ständige Ermitteln und Übermitteln des eigenen Standorts an diese zentrale Komponente unerlässlich. Das wiederum bedeutet einen hohen Energiebedarf und eine hohe Erneuerungsrate bei schnellen Ortswechseln. Auch müssten alle Nutzer die selbe Anwendung installiert haben um einen funktionsfähigen Dienst zu ermöglichen.
Selbst wenn diese Hürden überwindet werden würden, besteht noch immer die Gefahr, dass diese Positionsdaten missbraucht werden könnten. Die Sicherheitsmaßnahmen würden enorm sein um ein solches System gegenüber Missbrauch zu sichern. Ebenfalls müssten alle Nutzer ihr Vertrauen in dieses System setzten. Würden Nutzer ihre Internetverbindung für solch eine Anwendung absichtlich sperren oder hätten keinen Empfang, so hätte auch dieser Ansatz keinen Erfolg. Ein dezentraler Ansatz, welcher keine ständige Internetverbindung benötigt und von jedem Nutzer kontrolliert werden könnte wäre somit die bessere Lösung. Um zu wissen ob sich eine Person in der Nähe befindet ist nicht die genaue Position interessant, sonder nur die Distanz zu dieser Person beziehungsweise zu ihrem Gerät. Genau auf dieser Grundlage entstand das Projekt *mac-based identification and signaling service* (kurz MISS).
Jedes WLAN-fähige Gerät muss sich beim Senden oder Empfangen von Daten, aber auch in regelmäßigen Abständen mit anderen WLAN-Geräten in der Nähe abstimmen, ob bzw. wann die Luftschnittstelle frei zur Übertragung ist. Genau diese Kommunikation kann genutzt werden um Geräte in der Nähe zu Erkennen und Identifizieren.
Die einzige Information dazu die benötigt wird, ist die MAC-Adresse des Zielgerätes. Dabei hat der Besitzer des Gerätes die volle Kontrolle, wem er diese Adresse mitteilt.
Der *mac-based identification and signaling service* ist ein für Android geschriebener Hintergrunddienst, welcher es ermöglicht mobile oder stationäre WLAN Geräte zu erkennen. Der Service unterscheidet dabei zwischen mobilen Geräten, die hier als *Clients* bezeichnet werden und Stationen den sogenannten *Stations*. Hierbei sind die *Clients* Geräte die sich mit einem WLAN verbinden und *Stations* die solch ein Netzwerk aufspannen. Je nach Typ werden verschiedene Informationen, wie beispielsweise Signalstärke oder *ESSID* gesammelt und zurückgegeben. Diese Informationen werden in Kapitel 2.1.1 genauer erläutert und beschrieben.
Sobald sich eines der gesuchten Geräte in der Nähe befindet und erfolgreich erkannt wird, kann eine Benachrichtigung über die Anwesenheit einer damit verknüpften Person, ausgelöst werden.

Kapitel 2

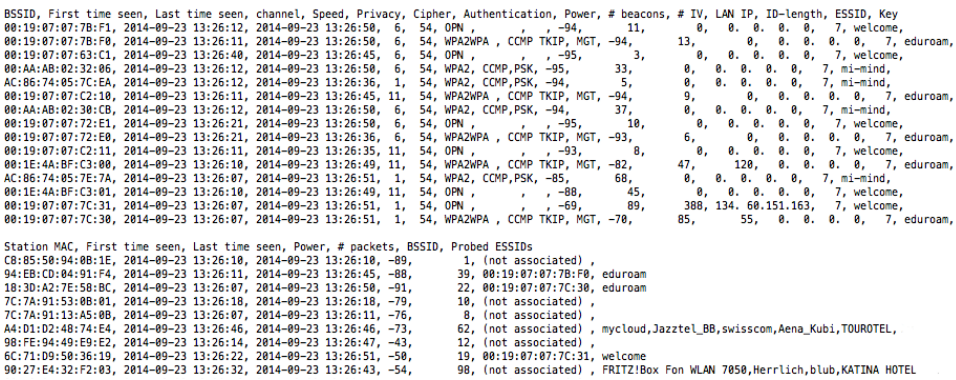
Grundlagen

Um die Funktionsweise des entwickelten Services genauer verstehen zu können, werden zunächst einige Grundlagen erklärt. In diesem Abschnitt werden die vom Service genutzten Technologien kurz beschrieben. Da dies keine vollständige Beschreibung darstellt, sollten die Technologien in ihren Grundzügen bereits bekannt sein.

2.1 Kabellose Kommunikation

Die Anzahl der Geräte die mithilfe von WLAN kommunizieren nehmen ständig zu. Bei einem Großteil dieser Geräte wird das *WNIC*[1], manchmal trotz limitierter Ressourcen, nicht abgeschaltet. Trotz verschlüsselter Verbindungen, werden durch den *IEEE 802.11* Standard [13] definiert, einige Daten in Klartext übertragen. Diese Daten werden von MISS genutzt um Geräte in der Nähe zu erkennen und Bekannte Geräte zu identifizieren.
Durch den *IEEE 802.11* Standard werden auf der Sicherungsschicht des *OSI-Modells* [14] Datagramme, sogenannte *Frames* definiert welche in spezielle Teile unterteilt sind. Jeder *Frame* enthält ein *MAC-Header* Feld, in dem die Absender MAC-Adresse im Klartext dargestellt ist. Durch belauschen der umliegenden Kommunikation kann so nach einer bestimmten MAC-Adresse gesucht werden. Aber nicht nur Geräte die gerade aktuell Daten austauschen können so erkannt werden, sonder auch Geräte die zur Zeit inaktiv sind. Jedes WLAN-fähige Gerät sendet abhängig von der Implementierung des *WNIC* Treibers periodisch Pakete aus. Diese Pakete gehören zu der Gruppe der *Managementframes*.
Bei den *probe request frames* handelt es sich um Pakete, die von *Clients* gesendet werden, um Informationen über naheliegende Netzwerke zu sammeln. Durch die Angabe einer *SSID*[2] kann im Kommunikationsbereich des Gerätes nach bestimmten Netzwerken gesucht werden. Durch die Angabe der *broadcast SSID* kann nach allen Netzwerken gesucht werden, die dieses Paket empfangen

[1] Wireless Network Interface Controller
[2] Service Set Identifier



```
BSSID, First time seen, Last time seen, channel, Speed, Privacy, Cipher, Authentication, Power, # beacons, # IV, LAN IP, ID-length, ESSID, Key
00:19:07:07:7B:F1, 2014-09-23 13:26:12, 2014-09-23 13:26:50,  6,  54, OPN ,       ,   , -94,       11,        0,   0.  0.  0.  0,   7, welcome,
00:19:07:07:7B:F0, 2014-09-23 13:26:11, 2014-09-23 13:26:50,  6,  54, WPA2WPA , CCMP TKIP, MGT, -94,      13,        0,   0.  0.  0.  0,   7, eduroam,
00:19:07:07:63:C1, 2014-09-23 13:26:40, 2014-09-23 13:26:45,  6,  54, OPN ,       ,   , -95,        3,        0,   0.  0.  0.  0,   7, welcome,
00:AA:AB:02:32:06, 2014-09-23 13:26:12, 2014-09-23 13:26:50,  6,  54, WPA2, CCMP,PSK, -95,       33,        0,   0.  0.  0.  0,   7, mi-mind,
AC:86:74:05:7C:EA, 2014-09-23 13:26:12, 2014-09-23 13:26:36,  1,  54, WPA2, CCMP,PSK, -94,        5,        0,   0.  0.  0.  0,   7, mi-mind,
00:19:07:07:C2:10, 2014-09-23 13:26:11, 2014-09-23 13:26:45, 11,  54, WPA2WPA , CCMP TKIP, MGT, -94,       9,        0,   0.  0.  0.  0,   7, eduroam,
00:AA:AB:02:30:CB, 2014-09-23 13:26:12, 2014-09-23 13:26:50,  6,  54, WPA2, CCMP,PSK, -94,       37,        0,   0.  0.  0.  0,   7, mi-mind,
00:19:07:07:72:E1, 2014-09-23 13:26:21, 2014-09-23 13:26:50,  6,  54, OPN ,       ,   , -95,       10,        0,   0.  0.  0.  0,   7, welcome,
00:19:07:07:72:E0, 2014-09-23 13:26:21, 2014-09-23 13:26:36,  6,  54, WPA2WPA , CCMP TKIP, MGT, -93,       6,        0,   0.  0.  0.  0,   7, eduroam,
00:19:07:07:C2:11, 2014-09-23 13:26:11, 2014-09-23 13:26:35, 11,  54, OPN ,       ,   , -93,       8,        0,   0.  0.  0.  0,   7, welcome,
00:1E:4A:BF:C3:00, 2014-09-23 13:26:10, 2014-09-23 13:26:49, 11,  54, WPA2WPA , CCMP TKIP, MGT, -82,      47,      120,   0.  0.  0.  0,   7, eduroam,
AC:86:74:05:7E:7A, 2014-09-23 13:26:07, 2014-09-23 13:26:51,  1,  54, WPA2, CCMP,PSK, -85,       68,        0,   0.  0.  0.  0,   7, mi-mind,
00:1E:4A:BF:C3:01, 2014-09-23 13:26:10, 2014-09-23 13:26:49, 11,  54, OPN ,       ,   , -88,       45,        0,   0.  0.  0.  0,   7, welcome,
00:19:07:07:7C:31, 2014-09-23 13:26:07, 2014-09-23 13:26:51,  1,  54, OPN ,       ,   , -69,       89,      388, 134. 60.151.163,   7, welcome,
00:19:07:07:7C:30, 2014-09-23 13:26:07, 2014-09-23 13:26:51,  1,  54, WPA2WPA , CCMP TKIP, MGT, -70,      85,       55,   0.  0.  0.  0,   7, eduroam,

Station MAC, First time seen, Last time seen, Power, # packets, BSSID, Probed ESSIDs
C8:85:50:94:0B:1E, 2014-09-23 13:26:10, 2014-09-23 13:26:10, -89,        1, (not associated) ,
94:EB:CD:04:91:F4, 2014-09-23 13:26:11, 2014-09-23 13:26:45, -88,       39, 00:19:07:07:7B:F0, eduroam
18:3D:A2:7E:58:BC, 2014-09-23 13:26:07, 2014-09-23 13:26:50, -91,       22, 00:19:07:07:7C:30, eduroam
7C:7A:91:53:0B:01, 2014-09-23 13:26:18, 2014-09-23 13:26:18, -79,       10, (not associated) ,
7C:7A:91:13:A5:0B, 2014-09-23 13:26:07, 2014-09-23 13:26:11, -76,        8, (not associated) ,
A4:D1:D2:48:74:E4, 2014-09-23 13:26:46, 2014-09-23 13:26:46, -73,       62, (not associated) , mycloud,Jazztel_BB,swisscom,Aena_Kubi,TOUROTEL,
98:FE:94:49:E9:E2, 2014-09-23 13:26:14, 2014-09-23 13:26:47, -43,       12, (not associated) ,
6C:71:D9:50:36:19, 2014-09-23 13:26:22, 2014-09-23 13:26:51, -50,       19, 00:19:07:07:7C:31, welcome
90:27:E4:32:F2:03, 2014-09-23 13:26:32, 2014-09-23 13:26:43, -54,       98, (not associated) , FRITZ!Box Fon WLAN 7050,Herrlich,blub,KATINA HOTEL
```

Abbildung 2.1: Mit airodump-ng erstellte Log-Datei

und darauf antworten. Diese Pakete werden versendet, egal ob das Gerät bereits mit einem Netzwerk verbunden ist oder nicht.
Stations senden sogenannte *beacon frames* aus. Damit geben sie periodisch ihre Präsenz, *SSID* und andere Parameter bekannt.
Im Standard Betriebsmodus eines *WNIC* ist es nicht vorgesehen, dass Pakete, die an andere MAC-Adressen gerichtet sind an höhere Schichten, bezogen auf das *OSI-Modell*, weitergeleitet werden. Diese Pakete werden bereits auf der zweiten Schicht dem *Data Link Layer* im *WNIC* verworfen. Damit aber auch diese Pakete weitergeleitet werden, ist es notwendig den Betriebsmodus des *WNIC* in den sogenannten *Monitor Mode* zu versetzten. Dieser Modus wird über den Treiber des *WNIC* eingestellt und muss dort vom Hersteller implementiert sein.

2.1.1 Airodump-ng

Airodump-ng [2] ist ein Kommandozeilenprogramm welches für das Erfassen von *IEEE 802.11 frames* genutzt wird und ist Bestandteil der *Aircrack-ng suite* [1]. Mit diesem Programm lassen sich Log-Dateien erstellen, welche Informationen über alle erfassten *Stations* und *Clients* enthalten. Eine Auszug einer solchen Log-Datei ist in Abbildung 2.1 zu sehen. Die Tabellen 2.1 und 2.2 beschreiben kurz die erfassbaren Daten für *Stations* und *Clients* [2].

2.2 Android

Bevor der eigentliche Hintergrunddienst installiert werden kann, müssen einige Vorbereitungen getroffen werden. Da Google in seinem offenen Betriebssystem

Tabelle 2.1: Stationdaten

Erfassbare *Station* Daten	Beschreibung
BSSID	MAC-Adresse der *Station*
First time seen	Datum und Uhrzeit der ersten Kontakts
Last time seen	Datum und Uhrzeit des letzten Kontakts
channel	Kanal auf der die *Station* sendet
Speed	Übertragungsgeschwindigkeit in MBit/s
Privacy	Privatsphäreneinstellungen
Cipher	Art der Verschlüsselung
Authentication	Genutztes Authentifizierungsprotokoll
Power	Signalstärke
# beacons	Anzahl der Empfangenen *beacon frames*
# IV	Anzahl der Erkannten Initialisierungsvektoren
LAN IP	IP Adresse
ID-length	Zeichenanzahl ESSID
ESSID	Netzwerkname
Key	Netzwerkschlüssel falls bekannt (genutzt mit aircrack-ng)

Tabelle 2.2: Clientdaten

Erfassbare *Client* Daten	Beschreibung
Station MAC	MAC-Adresse *Clients*
First time seen	Datum und Uhrzeit der ersten Kontakts
Last time seen	Datum und Uhrzeit der letzten Kontakts
Power	Signalstärke
# packets	Anzahl der Empfangenen *frames*
BSSID	BSSID mit dem der *Client* verbunden ist
Probed ESSIDs	ESSID die der *Client* sucht

Android keine Möglichkeit bietet den *Monitor Mode* explizit zu aktivieren, implementieren die Hersteller in ihren Gerätetreibern diesen auch nicht.
Zum Zeitpunkt dieses Projekts gibt es kein Android Gerät, welches einen *WNIC* besitzt, für den es einem vom Hersteller implementierten *Monitor Mode* gibt. Für bestimmte Chipsätze des Herstellers Broadcom [7] gibt es eine kleine Gruppe von Programmierer, die für die Chipsätze *BCM4330* [6] und *BCM4329* [5] eine neue Firmware geschrieben haben um eben diesen Modus zu aktivieren. Momentan werden folgende Geräte mit diesem Chipsatz unterstützt [3]:

- Samsung Galaxy GS1 mit Cyanogen 7
- Samsung Galaxy GS2 mit Cyanogen 9 oder Cyanogen 10
- HTC Nexus One mit Cyanogen 7
- Asus Nexus 7 mit Cyanogen 7

2.2.1 Cyanogen

Alle zuvor aufgelisteten Geräte nutzen Cyanogen [8]. Cyanogen ist eine erweiterte *open source* Firmware Distribution für Smartphones und Tablets, welche auf den Android Betriebssystem basiert. Cyanogen bietet Eigenschaften und Erweiterungen die es in der offiziellen Android Firmware oder in der von Herstellern ausgelieferten nicht gibt.
Um Cyanogen auf einem unterstützten Gerät zu installieren, muss dies zuvor *gerootet* werden. Diese bedeutet, das man alle Rechte erlangt was normalerweise auch aus Sicherheitsgründen von Google nicht vorgesehen ist.

2.2.2 bcmon.apk

Nachdem das Gerät *gerootet* wurde und Cyanogen installiert ist, wird ein zusätzliches Programm installiert, mit dessen Hilfe der *Monitor Mode* aktiviert werden kann. Das Programm *bcmon* [4] enthält dabei eine neuen Treiber für den *WNIC* in dem der *Monitor Mode* und weitere Modi implementiert sind. Anschließend ist das Gerät für die Verwendung des MISS Hintergrunddienstes vorbereitet.

Kapitel 3

Architektur und Implementierung

In diesem Kapitel wird zunächst die grundlegende Architektur des verwendeten Dienstes erläutert und anschließend auf die Implementierung eingegangen. Für diese Kapitel werden Grundlagen der Programmierung in Java und Android vorausgesetzt. Sind diese nicht vorhanden, wird an dieser Stelle auf die Android-Entwicklerseite [10] verwiesen.

3.1 Architektur

Dieser und darauffolgende Abschnitte erklären die Funktionsweise des Service, so wie er auch implementiert wurde. Daher ist auch die Dokumentation des Quelltextes des Services hilfreich. Es wird empfohlen auch dies ergänzend zu diesem Kapitel zu lesen.

3.1.1 Android Bound Service

Ein Service ist eine im Hintergrund laufende Komponente welche keine direkte Interaktion mit einem Nutzer besitzt. Da ein Service keine Benutzeroberfläche benötigt, ist dieser auch nicht an den normalen Lebenszyklus einer *Activity* gebunden. Im Allgemeinen werden Services genutzt um wiederkehrende und potentiell lange andauernde Aufgaben zu erledigen, wie beispielsweise das Herunterladen von Inhalten aus dem Internet oder das Aktualisieren von Daten. Ebenfalls werden Services mit einer höheren Priorität ausgeführt, als Anwendungen die sich im Hintergrund befinden. Somit ist es unwahrscheinlicher, dass sie vom Betriebssystem abgeschaltet werden.
Zusätzlich können Services unter Android so konfiguriert werden, dass sie neu gestartet werden sollte das Betriebssystem den Service beenden.
Da der in diesem Projekt genutzte Service nicht immer aktiv sein soll, wird hier auf eine besondere Art eines Services zurückgegriffen. Bei dieser Art von

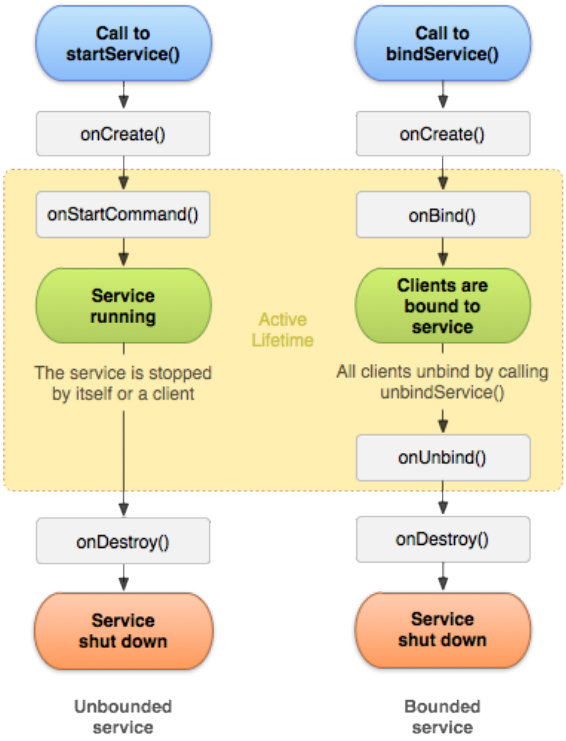


Abbildung 3.1: Lebenszyklus eines *Service* und *Bound Service* [12]

Hintergrunddienst handelt es sich um einen in *Bound Service* [11], welcher von der selben Klasse wie ein normaler *Service* erbt.
Man kann einen *Bound Service* als Server einer Client-Server-Kommunikation betrachten. Dies ermöglicht es anderen Komponenten sich an einen Service zu binden und so Anfragen zu senden und Antworten zu erhalten. Ebenso kann durch dies eine *Interprocess Communication (IPC)* realisiert werden, was bei normalen Services nur mit erhöhtem Aufwand oder der *Android Interface Definition Language (AIDL)* möglich wäre.
Um zwischen einem herkömmlichen und einem *Bound Service* zu unterscheiden, müssen lediglich verschiedene Methoden implementiert werden. Die Lebenszyklen eines *Service* und eines *Bound Service* sind in Abbildung 3.1 abgebildet.
Hierbei wird ersichtlich, dass diese Art von Service nicht immer im Hintergrund aktiv ist. Nur wenn ein Client an den Service gebunden ist wird, ist dieser aktiv. Im Gegensatz zu einem herkömmlichen Service, bei dem ein Aufruf von `stopService()` genügt um diesen zu beenden, wird ein *Bound Service* erst beendet, wenn sich alle Clients abgemeldet haben.

3.1.2 Service Logik

Die Kommunikation der Komponenten, wie sie im Service Implementiert wurden ist in Abbildung 3.2 ersichtlich. Bindet sich eine Anwendung den Service wird dieser gestartet, sofern dieser noch nicht aktiv war. Der Service läuft als eigenständiger Prozess und verfügt über einen zusätzlichen Thread, welcher die eigentliche Arbeit übernimmt. Dies ermöglicht ein sofortiges Abarbeiten ankommender Anfragen im Hauptprozess von bereits gebundenen Anwendungen oder solchen, die sich gerade an den Service binden möchten.
Aus Effizienzgründen wird der Arbeiter-Thread nur dann gestartet wenn Geräte in der Umgebung gesucht werden. Zu suchenden Geräte werden dann über Nachrichten den Service übermittelt, der daraufhin den Thread nach Bedarf startet oder stoppt.
Die Aufgabe des Threads ist es alle Geräte in der Nähe zu erfassen. Dabei werden alle zu erfassenden Daten, wie in Abschnitt 2.1.1 gezeigt, ausgelesen und protokolliert. Anschließend werden die gefunden Geräte mit denen verglichen, für die es eine Erinnerung gibt. Befindet sich ein solch ein Gerät unter den Gefunden so benachrichtigt der Arbeiter-Thread den eigentlichen Service. Der Service sendet eine Nachricht an die entsprechende Anwendung und informiert sie über den Fund des angefragten Gerätes.
Je nach Implementierung der Zielanwendung entscheidet diese über das weitere Vorgehen. In der Regel wird davon ausgegangen, dass die Anwendung das gesuchte Gerät nicht mehr benötigt, da eine Aktion ausgelöst wurde. Damit der Service das Gerät nicht weiterhin sucht, muss dies an den Service übermittelt werden. Durch eine Nachricht welche die Anwendung an den Service sendet, wird das zuvor gesuchte Gerät entfernt.
Der Service überprüft bei jedem Empfang einer Nachricht ob der Arbeiter-Thread gestartet werden muss oder nicht. Sind keine Geräte mehr vorhanden wird der Thread beendet. Antwortet eine Anwendung auf den Fund eines Gerätes nicht, wird davon ausgegangen, dass die Anwendung unerwartet beendet wurde. Der Service entfernt die Anwendung und alle mit dieser ihr in Verbindung stehenden Geräte.
Befinden sich danach keine gesuchten Geräte mehr im Dienst, beendet dieser den Arbeiter-Thread um Ressourcen an das Betriebssystem freizugeben. Haben sich alle Anwendungen ordnungsgemäß vom Service abgemeldet oder wurden durch eine ausbleibende Antwort entfernt, beendet sich der Service selbst da er nicht mehr benötigt wird.

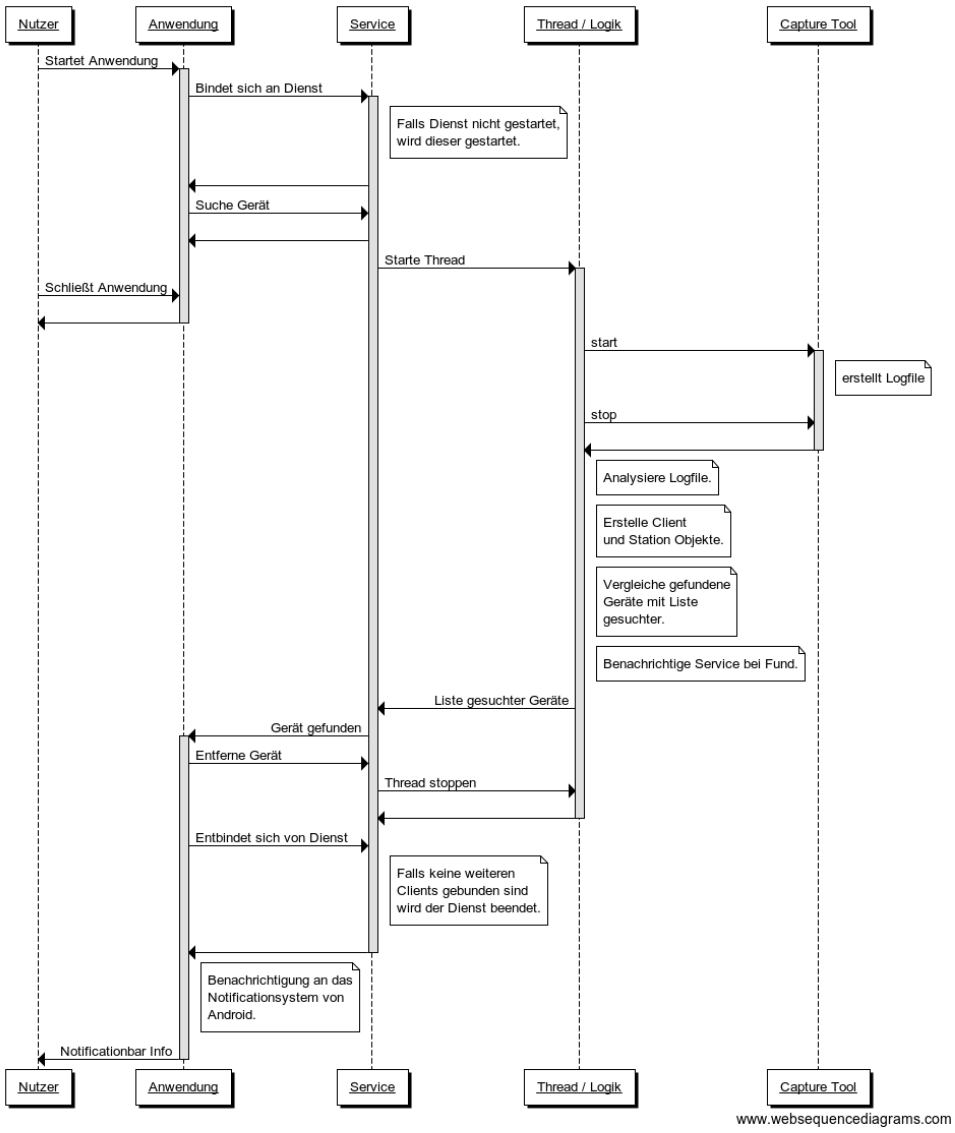


Abbildung 3.2: Anwendungsfall und Lebenszyklus des MISS als UML-Sequenzdiagramm.

3.2 Implementierung

Für die Implementierung des erläuterten Aufbaus wurde das Eclipse ADT [9] verwendet. Als Zielgerät wurde das Samsung Galaxy GS1 [15] mit Cyanogen 7 gewählt. Hierbei war zu beachten, das bei der Programmierung nur Methoden

bis API Level 10 genutzt werden konnten. Dies liegt an der geringen Android Version 2.3.3 welche dem Cyanogen 7 Mod zu Grunde liegt.
Da im späteren Betrieb des Service Skripte benötigt werden, wurden diese zur besseren Kapselung und Entwicklung als `.sh` Datei im `assets` Ordner gespeichert. Dies hat den entscheidenden Vorteil, dass diese programmatisch generiert werden können und nicht nach der Installation von Nutzer im entsprechenden Programmverzeichnis abgelegt werden müssen.
Ist die Luftschnittstelle auf einem anderen Gerät nicht unter der Standard Bezeichnung `wlan0` zu finden, so kann dies bequem in den Skripts des Projekts geändert werden.
Folgende Skripts werden bei der Installation erstellt:

- `removeCaptureFiles.sh` - entfernt Log-Dateien.
- `startCapture.sh` - startet das Aufzeichnen der umliegenden Kommunikation.
- `stopCapture.sh` - stoppt das Aufzeichnen.

Alle Skripte werden ausschließlich vom Arbeiter-Thread genutzt. Um unnötige Speicherbelegung zu vermeiden dient das `removeCaptureFiles.sh` Skript, welches die anfallenden Log-Datei entfernt.
Das Starten und Stoppen von *airodump-ng* wird durch die beiden anderen Skripte veranlasst, wobei `startCapture.sh` wie in Listing 3.1 einige Parameter enthält. So wird hier das Ausgabeformat festgelegt und der *WNIC* Name angegeben. Des weiteren werden die Umgebungsvariablen ergänzt, um gewisse Bibliotheken für *airodump-ng* bereitzuhalten damit dies fehlerfrei Ausführt werden kann.

Listing 3.1: Airodump-ng Parameter

```
1 export PATH = $PATH:/data/data/com.bcmon.bcmon/files/tools
2 export LD_LIBRARY_PATH =
3   $LD_LIBRARY_PATH:/data/data/com.bcmon.bcmon/files/libs
4 export LD_PRELOAD = /data/data/com.bcmon.bcmon/files/libs/libfake_driver.so
5 airodump-ng -w /datadata/de.uulm.miss/files/capture
6   --output-format csv -w capture wlan0 2>&1
```

In der nachfolgenden Auflistung werden alle Klassen und eine dazugehörige Beschreibung des Service MISS aufgeführt.

Auf eine ausführliche Beschreibung der Funktionen und Parameter wurde an dieser Stelle verzichtet, da diese aus dem Quellcode entnommen werden kann. Um diesen Service zu nutzen zu können kann die beiliegende Anwendung, *PAR* welche in Abschnitt 3.3 erläutert wird, genutzt werden.
Die Basisfunktionen die der Service als Schnittstelle anbietet werden im nächsten Kapitel 3.2.1 erklärt.

Tabelle 3.1: Übersicht von essentiellen Klassen

Klassenname	Beschreibung
MainActivity.java	Wird nur bei der Installation geöffnet und erzeugt alle benötigen Skripte und legt diese im entsprechenden Ordner ab.
MISService.java	Nimmt Anwendungsanfragen entgegen, erzeugt und kontrolliert den Arbeiter-Thread.
ScanLogic.java	Arbeiter-Thread der mithilfe der Skripte alle Geräte in der Umgebung findet und mit zu suchenden Geräten vergleicht.
FileParser.java	Erstellt Client und Station Objekte anhand des Logfiles.
Client.java	Objekt, welches alle für Clients erfassbaren Daten enthält.
Station.java	Objekt, welches alle für Stations erfassbaren Daten enthält.
ScanOrder.java	Enthält die Liste von gesuchten Stations und Clients.

3.2.1 Nutzung des Service

Für die Nutzung des Service müssen folgende Punkte erfüllt sein:

- Ein unterstützter Chipsatz (`BCM4330` bzw. `BCM4329`) wurde verbaut.
- Das Gerät ermöglicht *root* Zugriff.
- Eine kompatible Cyanogen Firmware ist aufgesetzt.
- Die *bcmon* Anwendung ist installiert.
- Der Service MISS ist installiert und *root* Rechte wurden bewilligt.
- Der *Monitor Mode* wurde über *bcmon* aktiviert.

Für eine Nutzung des Service MISS innerhalb einer Anwendung müssen nicht viele Änderungen an einer bestehenden Anwendung vorgenommen werden. Die Klasse welche mit dem Service letztendlich Kommuniziert soll, muss das Interface `ServiceConnection` implementieren.
Hierbei müssen die Methoden `onServiceConnected(ComponentName name, IBinder service)` und `onServiceDisconnected(ComponentName name)` implementiert werden. Erstere wird aufgerufen, wenn die Verbindung zum Service erfolgreich aufgebaut wurde, letztere, wenn die Verbindung zum Service abbricht oder beendet wurde.

An Service binden

Listing 3.2 zeigt, wie eine Anwendung an den Service gebunden wird.
Das `ServiceConnection` Objekt ist in diesem Fall die `Aktivity` welche `ServiceConnetion` implementiert. Durch den Aufruf von `bindService()` wird die Anwendung an den Service gebunden und die Methode `onServiceConnected()` wird durchlaufen.

Listing 3.2: Anwendung an Service binden.

```
1  ServiceConnection mConnection = this;
2  Intent mIntent = new Intent("de.uulm.miss.MISService");
3  bindService(mIntent, mConnection, Context.BIND_AUTO_CREATE);
```

An dieser Stelle wird ein Objekt vom Typ `Messenger` benötigt. Diese Objekt ermöglicht eine Umsetzung für eine nachrichtenbasierte Kommunikation über Prozesse hinweg. Somit lassen sich Nachrichten an den Service übermitteln und von diesem empfangen. Das Objekt wird wie in Listing 3.3 initialisiert wobei sich die Anwendung gleich am Service registrieren. Dies wird mittels einer Nachricht an den Service und der entsprechenden Konstanten umgesetzt.

Listing 3.3: Registrieren der Anwendung am Service

```
1  ...
2  private final Messenger mMessenger =
3    new Messenger(new IncomingMessageHandler(this));
4  ...
5  @Override
6  public void onServiceConnected(ComponentName name, IBinder service) {
7    mServiceMessenger = new Messenger(service);
8    try {
9      Message regMessage = Message.obtain(null, MSG_REGISTER_APPLICATION);
10     regMessage.replyTo = mMessenger;
11     mServiceMessenger.send(regMessage);
12   } catch (RemoteException e) {
13         // In diesem Fall wurde der Service unerwartet beendet,
14         // bevor etwas mit ihm gemacht werden konnte.
15   }
16 }
```

Die Konstanten mit denen zwischen Anwendung und Service kommuniziert werden kann sind im nachfolgenden Aufzählung aufgeführt.

- `MSG_REGISTER_APPLICATION` - Anwendung an Service binden.
- `MSG_UNREGISTER_APPLICATION` - Anwendung am Service abmelden.
- `MSG_ADD_CLIENT` - Einen neuen zu suchenden *Client* am Service hinzufügen.
- `MSG_ADD_STATION` - Einen neue zu suchende *Station* am Service hinzufügen.
- `MSG_REMOVE_CLIENT` - Ein *Client* aus dem Service entfernen.
- `MSG_REMOVE_STATION` - Eine *Station* aus dem Service entfernen.
- `MSG_FOUND_DEVICE` - Service sendet dies bei einem Fund an die entsprechende Anwendung.

Um auch eine Antworten verarbeiten zu können, muss wie in Listing 3.3 in Zeile 2 & 3 ein `Messenger` Objekte erstellt werden. Die Klasse `IncomingHandler` erbt von `Handler`, wobei die Methode `handleMessage()` überschrieben werden muss, um die vom Entwickler gewünschten Aktionen durchführen zu können. Eine Implementierung von `Handler` könnte beispielsweise wie in Listing 3.4 aussehen. In diesem Fall wird überprüft ob ein Gerät gefunden wurde.

Listing 3.4: Behandelt Nachrichten die MISS an die Anwendung sendet.

```
1  private class IncomingMessageHandler extends Handler {
2    MainActivity main;
3    public IncomingMessageHandler(MainActivity parent) {
4       main = parent;
5    }
6    @Override
7    public void handleMessage(Message msg) {
8
9    switch (msg.what) {
10     case MSG_FOUND_DEVICE:
11           Bundle b = new Bundle();
12           b.putString("MAC", (String) msg.getData().get("MAC"));
13           b.putString("Name", (String) msg.getData().get("Name"));
14
15           sendMessageToService(b, MSG_REMOVE_CLIENT);
16       break;
17     default:
18       super.handleMessage(msg);
19     }
20   }
21 }
```

Ist dies der Fall wird eine neue Nachricht an der Service gesendet damit dieser das Gerät aus seiner Liste von zu suchenden Geräten entfernt.
Um Nachrichten zu senden wurde in Listing 3.4 die Methode `sendMessageToService()` genutzt die in Listing 3.5 vorgestellt wird.

Listing 3.5: Sendet Nachrichten an den Service

```
1  private void sendMessageToService(Bundle data, int action) {
2    if (mServiceMessenger != null) {
3      try {
4        Message msg = Message.obtain(null, action);
5        msg.setData(data);
6        msg.replyTo = mMessenger;
7        mServiceMessenger.send(msg);
8      }
9      catch (RemoteException e) {
10       ...
11     }
12   }
13 }
```

3.3 Anwendungsprogramm PAR

Um die Praktikabilität des MISS Services und die Kommunikation mit einer anderen Anwendung zu testen, wurde im Laufe des Projekts die Anwendung PAR entwickelt. PAR steht in diesem Fall für *Person Aware Reminder*. Die Anwendung soll ein Einsatzgebiet des Service MISS aufzeigen. Hierbei können

Erinnerungen an Personen gebunden werden.
Dabei werden die MAC-Adressen der Smartphone(s) der Besitzer im Adressbuch angegeben. Unter `Instantmessenger` wird die Adresse mit dem Label `MAC` abgelegt. Das Format muss aus Großbuchstaben bzw. Zahlen bestehen und die Blöcke müssen durch Doppelpunkte getrennt werden. Des weiteren erlaubt die Anwendung auch Funktionen, wie sie eine einfache Erinnerungsanwendung bietet. Unter anderem sind zeit- und ortsabhänige Erinnerungen möglich und es besteht die Möglichkeit einfache Notizen zu hinterlegen. In der Abbildung 3.3 sind Auszüge aus der Anwendung zu sehen.

Wird nun eine Erinnerung erstellt, welche eine Bezug auf eine Person nimmt, kann diese bequem über das im Smartphone vorhandene Adressbuch ausgewählt werden. Hierbei sind nur die Namen der Personen zu sehen und keine weiteren Details. Wird die Erinnerung gespeichert kommunizieren Anwendung und Service automatisch miteinander ohne weitere Schritte des Benutzers.

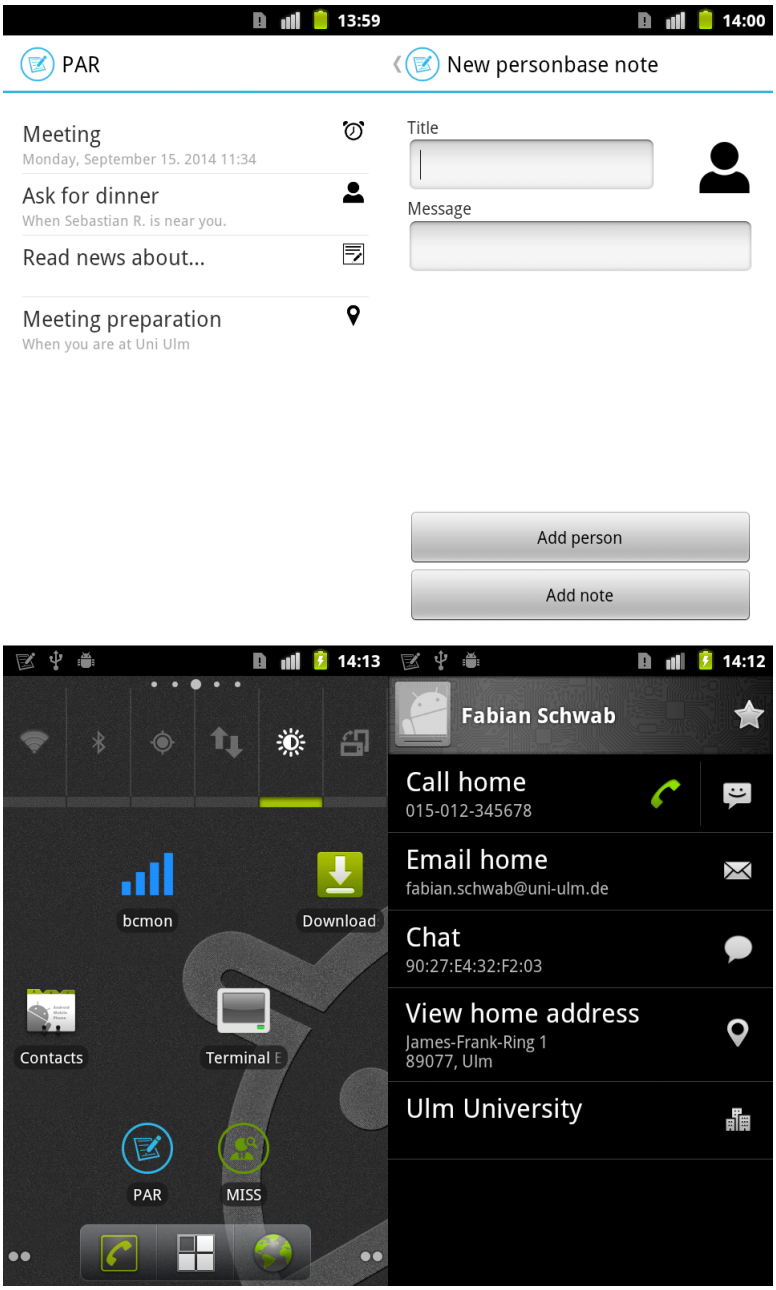


Abbildung 3.3: Screenshots der Anwendung PAR und Notifikation in der Statusleiste.

Kapitel 4

Zusammenfassung und Ausblick

In diesem Projekt wurde ein Service für das mobile Betriebssystem Android entwickelt. Dieser Service nutzt WLAN um Geräte in seiner Nähe zu erfassen und zu identifizieren. Das wird dadurch erreicht, indem die komplette Netzwerkkommunikation in der Umgebung mitgehört wird. Anhand von bestimmten Paketen werden MAC-Adressen anderer Geräte ermittelt und mit MAC-Adressen bekannter Geräte verglichen.
Der Service steht alle Hintergrunddienst bereit und kann von beliebig vielen Anwendungen genutzt werden. Wichtig hierbei ist, dass die Anwendungen korrekt mit dem Service kommunizieren. Die Kommunikation findet über ein Anfrage-Antwort Schema statt.
Mit diesem entwickelten Service und der ebenfalls implementierten Beispielanwendung wurde das zu Beginn beschriebene Problem gelöst. Es lassen sich MAC-Adressen von WLAN-fähigen Geräten übergeben, die anschließend vom Service gesucht werden. Befindet sich ein Gerät in der Nähe, wird die Anwendung vom Service benachrichtigt und der Nutzer wird wiederum von der Anwendung über den Fund informiert.

Ein Manko das hierbei aber noch bleibt, ist die Geräteabhängigkeit und die damit einhergehende Limitierung auf Android 2.3.3.
Würde Google für zukünftige Versionen des Betriebssystem eine Schnittstelle anbieten um den *Monitor Mode* zu aktivieren wäre dieses kein Problem mehr, da hierdurch die Implementierung des *Monitor Modes* im Gerätetreiber zwingend wäre.

4.1 Ausblick

Der Service ist in seinen Grundzügen gut nutzbar, aber dennoch hat er ein paar offensichtliche Schwachstellen. Diese Schwachstellen könnten aber durch eine

Weiterentwicklung entfernt werden.
So können zum Beispiel beim abtippen der MAC-Adresse Fehler gemacht werden. Ein falscher Buchstabe oder eine falsche Zahl kann dazu führen, dass das eigentliche Gerät nicht mehr gefunden werden kann.
Um dies auszuschließen, könnte das einlesen der MAC-Adresse automatisiert werden. Durch ein vorherigen Suchvorgang können alle in der Nähe befindlichen Geräte aufgelistet werden um das richtige anschließend auszuwählen. Eine weitere Möglichkeit ist durch eine alternative Verbindung, wie NFC oder Bluetooth, um die MAC-Adresse zwischen den betreffenden Geräten auszutauschen.
Durch einen aktivierten *Moditor Mode* kann kein Datenaustausch mehr über WLAN erfolgen. Dieser wird dann vom Betriebssystem automatisch auf das Mobilfunknetz ausgelagert. Um eine WLAN Nutzung zu ermöglichen, wäre es sinnvoll bekannte Netzwerke zu suchen und den *Monitor Mode* dann je nach bedarf zu deaktivieren. Beispielsweise könnte bei aktivem Display der *Monitor Mode* deaktiviert werden. Ebenfalls könnte der Service noch um eine Funktionalität erweitert werden, mit dem sich der *Monitor Mode* automatisch in begrenzten Zeitintervallen an- und abschalten lässt. Dies schont die Batterie und trägt zu einer längeren Laufzeit des Gerätes bei.

Literaturverzeichnis

[1] AirCrack-ng. AirCrack-ng Suite. http://www.aircrack-ng.org, April 2014. zuletzt besucht am 13. Oktober 2014.

[2] AirCrack-ng. Airodump-ng Documentation. http://www.aircrack-ng.org/doku.php?id=airodump-ng, April 2014. zuletzt besucht am 13. Oktober 2014.

[3] bcmon Authors. bcmon Blog. http://bcmon.blogspot.de. zuletzt besucht am 13. Oktober 2014.

[4] bcmon Authors. Monitor Mode for Broadcom WiFi Chipsets. https://code.google.com/p/bcmon. zuletzt besucht am 13. Oktober 2014.

[5] Broadcom Corporation. BCM4329 Technische Informationen. http://www.broadcom.com/products/Wireless-LAN/802.11-Wireless-LAN-Solutions/BCM4329. zuletzt besucht am 13. Oktober 2014.

[6] Broadcom Corporation. BCM4330 Technische Informationen. http://www.broadcom.com/products/Wireless-LAN/802.11-Wireless-LAN-Solutions/BCM4330. zuletzt besucht am 13. Oktober 2014.

[7] Broadcom Corporation. Internetpräsenz. http://www.broadcom.com/company. zuletzt besucht am 13. Oktober 2014.

[8] CyanogenMod. CyanogenMod Website. http://www.cyanogenmod.org. zuletzt besucht am 13. Oktober 2014.

[9] Eclipse Foundation and Google Inc. Eclipse and Android Developer Tools Bundle. http://developer.android.com/sdk/index.html. zuletzt besucht am 13. Oktober 2014.

[10] Google Inc. Android Developer Site. http://developer.android.com. zuletzt besucht am 13. Oktober 2014.

[11] Google Inc. Bound Services on Android Developer. http://developer.android.com/guide/components/bound-services.html. zuletzt besucht am 13. Oktober 2014.

[12] Google Inc. Services unter Android. http://developer.android.com/guide/components/services.html. zuletzt besucht am 13. Oktober 2014.

[13] IETF. RFC5416. http://tools.ietf.org/html/rfc5416, März 2009. zuletzt besucht am 13. Oktober 2014.

[14] ITU-T Study Group 17. ITU-T X.200. http://handle.itu.int/11.1002/1000/2820, September 1994. zuletzt besucht am 13. Oktober 2014.

[15] Samsung Group. Technical Specs of Samsung GALAXY S I9000. http://www.samsung.com/de/consumer/mobile-device/mobilephones/archive-mobile-phones/GT-I9000HKDDBT. zuletzt besucht am 13. Oktober 2014.