MISS
Mac-based Identification and Signaling Service

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

26. September 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 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 ein bestimmtes Ereignis erinnert.
Doch sind Zeitabhängige Erinnerungen nicht immer praktisch. In manchen Situationen lässt sich nicht genau festlegen wann eine Aufgabe zu erledigen ist.
Sollte zum Beispiel etwas bestimmtes nach der Arbeit erledigen 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, oder wurde vom Nutzer bereits wieder vergessen, da diese schon vor Stunden erschien.
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 möglich einzustellen, dass beim verlassen der Arbeit, etwas bestimmtes erledigen sollte. Damit lassen sich schon eine Vielzahl an möglichen Szenarien abdecken. Doch gibt es noch einen großen Teil, welcher weder Zeit noch Ortsgebunden bestimmt werden kann. Diese trifft vor allem auf Szenarien in Zusammenhang mit Personen zu.
Es lässt sich schwer vorausahnen wann und wo man eine Person antrifft womit Orts- und Zeitgesteuerte Erinnerungen entfallen. Einfache Notizen erscheinen hier noch das beste Mittel der Wahl zu sein. Doch an diese muss sich der Nutzer wieder 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. Doch wie ist dies am besten zu realisieren?

Die Lösung dieses Problems befindet sich bereits in der Hosentasche der meisten Menschen. Wie zu beginn erwähnt, sind Smartphones und Tablets ständig griffbereit und ein ständiger Begleiter. Da diese Geräte meist ohne Unterbrechung mit dem Internet verbunden sind, liegt die Lösung nahe, alle Aufenthaltsorte aller Nutzer auf einem zentralen Server zu speichern. Doch würde diese Lösung einige Probleme mit sich bringen.
Unter anderem wäre das ständige Ermitteln und Übermitteln des eigenen Standorts unerlässlich, was wiederum einen hohen Energiebedarf und einer hohen Erneuerungsrate bei schnellen Ortswechseln mit sich zieht. Auch müssten alle Nutzer die selbe Anwendung installiert haben um einen funktionsfähigen Dienst zu ermöglichen.
Selbst wenn diese Hürden überwindet werden würden, besteht immer noch die Gefahr, dass diese Daten missbraucht werden könnten. Die Sicherheitsmaßnahmen würden enorm sein um solch ein 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 beste Lösung. Da keine Notwendigkeit besteht ständig wissen zu müssen wo sich eine gesuchte Person aufhält, ist eine Erinnerung nur von Interesse wenn diese Person bzw. ihr begleitendes Smartphone sich in unmittelbarer Nähe befindet. 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 und auch sonst in Gewissen 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 seiner Nähe zu erkennen und identifizieren.
Die einzige Information 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 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 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 erkannt wird, kann eine Benachrichtigung über dessen Fund ausgelöst werden.

Kapitel 2

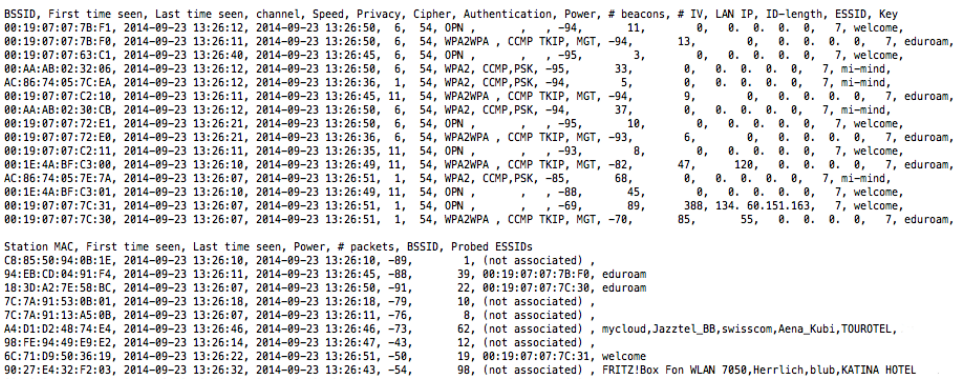
Grundlagen

Um die Funktionsweise des Service genauer verstehen zu können, sollten zunächst einige Grundlagen erklärt werden. In diesem Abschnitt werden die vom Service genutzten Technologien kurz beschrieben. Da dies keine vollständige Beschreibung darstellt, sollten die Technologien in ihren Grundzügen 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 definiert durch den *IEEE 802.11* Standard einige Daten in Klartext übertragen. Diese Daten werden von MISS genutzt um Geräte in der Nähe zu erkennen und bekannte zu identifizieren.
Durch den *IEEE 802.11* Standard werden auf der Sicherungsschicht des *OSI-Modell* 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 im Moment Daten austauschen können erkannt werden, sonder auch Geräte die inaktiv sind. Jedes WLAN fähige Gerät senden abhängig von der Implementierung des *WNIC* Treibers periodische 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. Aber auch durch Angabe der *broadcast SSID* kann nach allen Netzwerken gesucht werden, die dieses Paket empfangen und darauf Antworten. Diese Pakete werden versendet egal ob das Gerät bereits

[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

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* auf den sogenannten *Monitor Mode* zu setzten. Dieser Modus wird über den Treiber des *WNIC* eingestellt um muss dort aber auch implementiert sein.

2.1.1 Airodump-ng

Airodump-ng[3] ist ein Kommandozeilenprogramm welches für das erfassen von *IEEE 802.11 frames* genutzt wird und Bestandteil der *Aircrack-ng suit*[4]. Mit diesem Programm lassen sich Log-Dateien erstellen, welche Informationen über alle erfassten *Stations* und *Clients* enthalten. Eine Auszug von solch einer Log-Datei ist in Abbildung 2.1 zusehen. Die nachfolgenden Auflistungen beschreiben stichpunktartig die erfassbaren Daten für *Stations* und *Clients*.

[3] http://www.aircrack-ng.org/doku.php?id=airodump-ng
[4] http://www.aircrack-ng.org

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	Länge der ESSID
ESSID	Netzwerkname
Key	Netzwerkschlüssel falls bekannt (genutzt mit aircrack-ng)

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

2.2 Android

Bevor der eigentliche Hintergrunddienst installiert werden kann, müssen einige Vorbereitungen getroffen werden. Da Google in seinem offenen Betriebssystem Android keine Möglichkeit bietet einen *Monitor Mode* zu aktivieren, implementieren die Hersteller in ihren Gerätetreibern diesen auch nicht.
Aktuell 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 gibt es eine kleine Gruppe von Programmierer, die für die Chipsätze *BCM4330* und *BCM4329* eine neue Firmware geschrieben haben um eben diesen Modus zu aktivieren. Momentan werden folgende Geräte mit diesem Chipsatz unterstützt:

- Samsung Galaxy GS1 mit Cyanogen 7
- Samsung Galaxy GS2 mit Cyanogen 9 oder Cynaogen 10

- HTC Nexus One mit Cyanogen 7
- Asus Nexus 7 mit Cyanogen 7

2.2.1 Cyanogen

Alle zuvor aufgelisteten Geräte nutzen Cyanogen[5]. 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 oder in der von Herstellern mit ausgelieferten Firmware nicht gibt.
Um Cyanogen auf einem unterstützten Gerät zu installieren, muss es zuvor *gerootet* werden. Diese bedeutet, das man alle Rechte erlangt was normalerweise auch aus Sicherheitsgründen 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* enthält dabei eine neuen Treiber für den *WNIC* indem der *Monitor Mode* und weitere Modi implementiert sind. Anschließend ist das Gerät für die Verwendung des MISS Hintergrunddienstes vorbeireitet.

[5] http://www.cyanogenmod.org

Kapitel 3

Architektur und Implementierung

In diesem Kapitel wird zunächst die grundlegende Architektur des verwendeten Dienstes erläutert und anschließend auf die Implementierung eingegangen. Dieses und nachfolgende Kapitel setzten voraus, dass Grundlagen der Programmierung in Java und Android bekannt sind. Sind diese nicht vorhanden, wird an dieser Stelle auf die Android-Entwicklerseite[1] 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 und es wird empfohlen auch dies ergänzend hierzu 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 kein Benutzeroberfläche benötigt, ist dieser auch nicht an den 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. Ebenfalls 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 sie beenden.

[1] http://developer.android.com

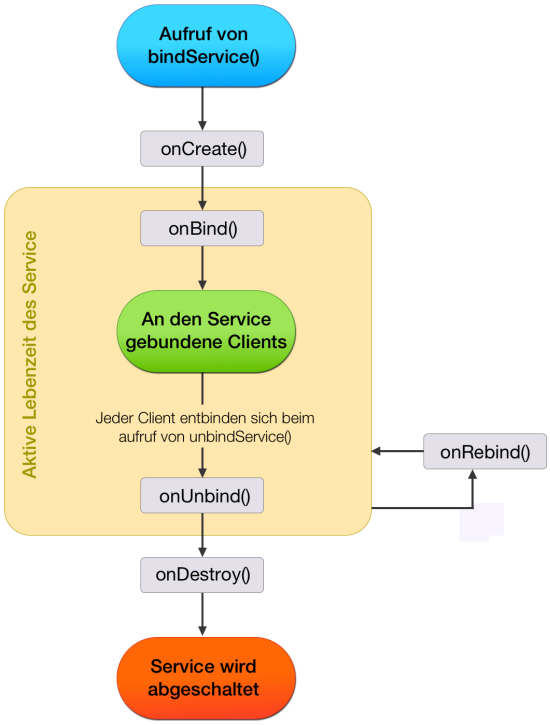


Abbildung 3.1: Lebenszyklus eines *Bound Service*

Da der in diesem Projekt genutzte Service nicht immer aktiv sein soll, wird hier auf eine besondere Art eines Service zurückgegriffen. Bei dieser Art von Hintergrunddienst handelt es sich um einen in *Bound Service* welcher von der selben Klasse wie ein normaler *Service* ist.
Man kann einen *Bound Service* als Server einer Client-Server-Schnittstelle betrachten. Dies ermöglicht den Android 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 mit der *Android Interface Definition Language (AIDL)* möglich wäre.
Um zwischen einem herkömmlichen und einem *Bound Service* zu unterscheiden, müssen nur verschiedene Methoden implementiert werden. Der Lebenszyklus eines *Bound Service* ist 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 entbunden haben.

3.1.2 Service Logik

Der Aufbau des Service, wie er Implementiert wurde 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 Anwendungen die sich gerade an den Service binden möchten.
Aus Effizienzgründen wird der Arbeiter-Thread nur gestartet wenn Geräte gesucht werden. Zu suchenden Geräte werden an 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, erfasst. Anschließend werden die gefunden Geräte mit den gesuchten Verglichen. Befindet sich ein gesuchtes unter den gefunden so benachrichtigt der Arbeiter-Thread seinen 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, das die Anwendung das gesuchte Gerät nicht mehr benötigt und dies per Nachricht an den Service übermittelt. Durch eine Nachricht an den Service wird das zuvor gescuhte Gerät entfernt.
Der Service überprüft bei jedem Empfang einer Nachricht ob der Arbeiter-Thread gestartet werden muss oder nicht. Werden Clients oder Stations gesucht wird der Thread gestartet. Sind keine Geräte mehr vorhanden wird der Thread beendet. Antwortet eine Anwendung auf den Fund eines seiner Gesuchten Geräte nicht, wird davon ausgegangen das die Anwendung unerwartet beendet wurde. Der Service entfernt die Anwendung und all ihrer Geräte welche im Zusammenhang mit ihr stehen.
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.

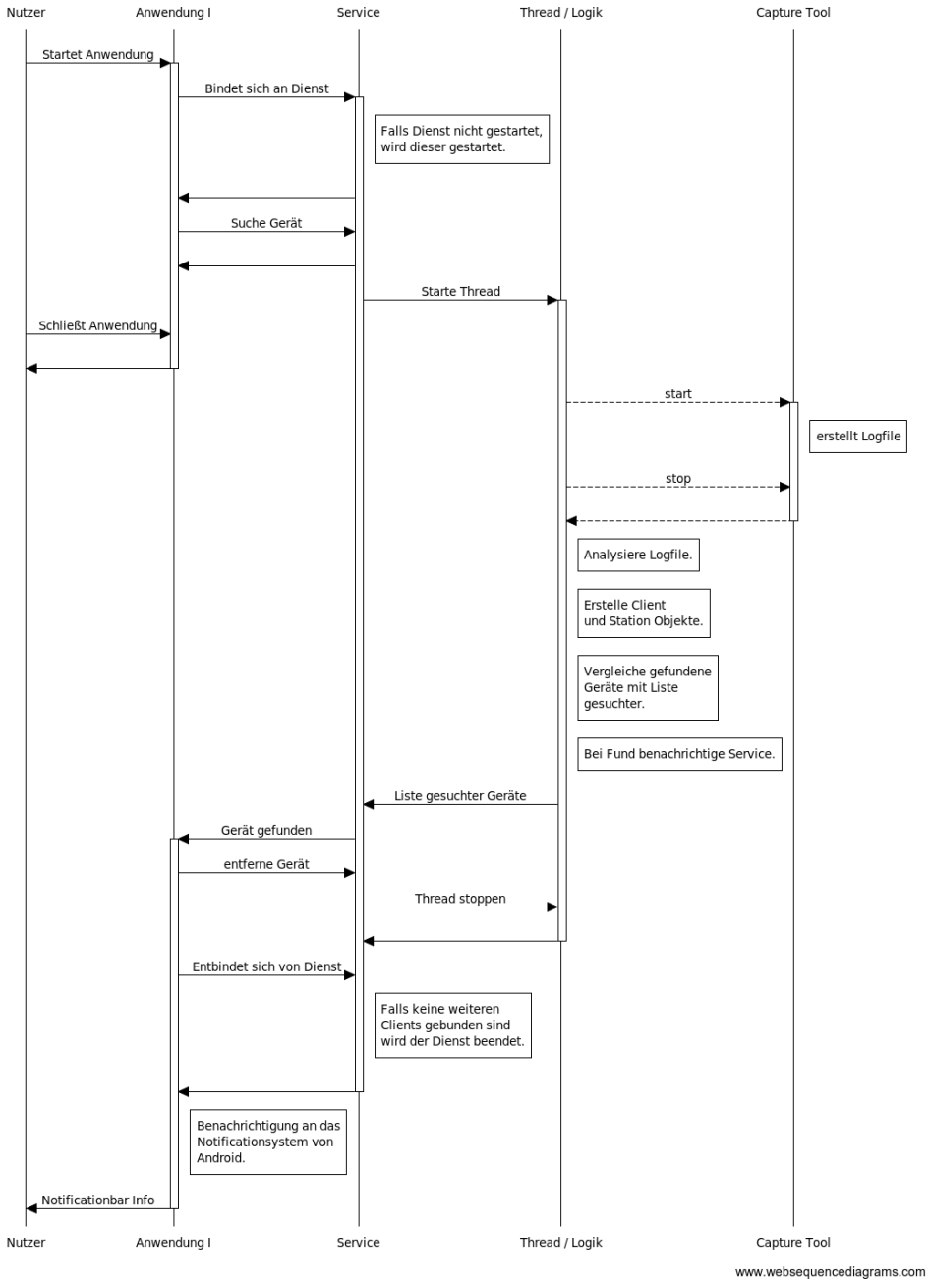


Abbildung 3.2: Anwendungsfall und Lebenszyklus des MISS als UML-Sequenzdiagramm.

3.2 Implementierung

Für die Implementierung der voran gezeigten Architektur wurde das Eclipse ADT[2] verwendet. Als Zielgerät wurde das Samsung Galaxy GS1 mit Cyanogen 7 gewählt. Hierbei war zu beachten, das bei der Programmierung nur Eigenschaften bis API Level 10 genutzt wurden. 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 in *assets* gespeichert. Dies hat den entscheidenden Vorteil, dass die programmatisch generiert werden können und nicht nach der Installation von Nutzer im Programmverzeichnis abgelegt werden müssen.
Ist die Luftschnittstelle auf einen 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
- startCapture.sh
- stopCapture.sh

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. Unter anderem 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.

[2] http://developer.android.com/sdk/index.html

Klassenname	Beschreibung
MainActivity.java	Wird nur bei der Installation geöffnet und erzeugt alle nötigen Skripte.
MISService.java	Nimmt Anwendungsanfrage entgegen und erzeugt und kontrolliert den Arbeiter-Thread.
ScanLogic.java	Arbeiter-Thread der mithilfe der Skripte alle Geräte findet und mit zu suchenden Vergleicht.
FileParser.java	Erstellt Client und Station Objekte anhand des Logfiles.
Client.java	Objekt welche alle für Clients erfassbaren Daten enthält.
Station.java	Objekt welche alle für Stations erfassbaren Daten enthält.
ScanOrder.java	Enthält die Liste von gesuchten Stations und Clients.

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 kann die beiliegende Anwendung, *PAR* welche in Abschnitt 3.3 erläutert wird, genutzt werden.
Die Basisfunktionen die der Service nach außen hin bietet werden im nächsten Kapitel 3.2.1 erklärt.

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 zu Service erfolgreich aufgebaut wurde und 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 *Service-Connetion* 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 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
- MSG_UNREGISTER_APPLICATION
- MSG_ADD_CLIENT
- MSG_ADD_STATION
- MSG_REMOVE_CLIENT
- MSG_REMOVE_STATION
- MSG_FOUND_DEVICE

Um auch eine Antworten verarbeiten zu können, muss wie in Listing 3.3 Zeile 2 & 3 ein *Messenger* Objekte erstellt werden. Die Klasse *IncomingHandler* erbt von*Handler*, wobei die Methode *handleMessage()* überschrieben werden muss, um die von Entwickler gewünschten Aktionen durchführen zu können. Eine Implementierung von *Handler* könnte dann 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 gesendet damit der Service das Gerät aus seiner Liste von suchenden entfernt.
Um Nachrichten zu senden wurde in Listing 3.4 die Methode *sendMessageToService* genutzt die in Listing 3.5 abgebildet ist.

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 den Service MISS zu Nutzen und zu Testen wurde im Laufe des Projekts die Anwendung PAR geschrieben. 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 schreiben. In der Abbildung 3.3 sind Auszüge aus der Anwendung zu sehen.

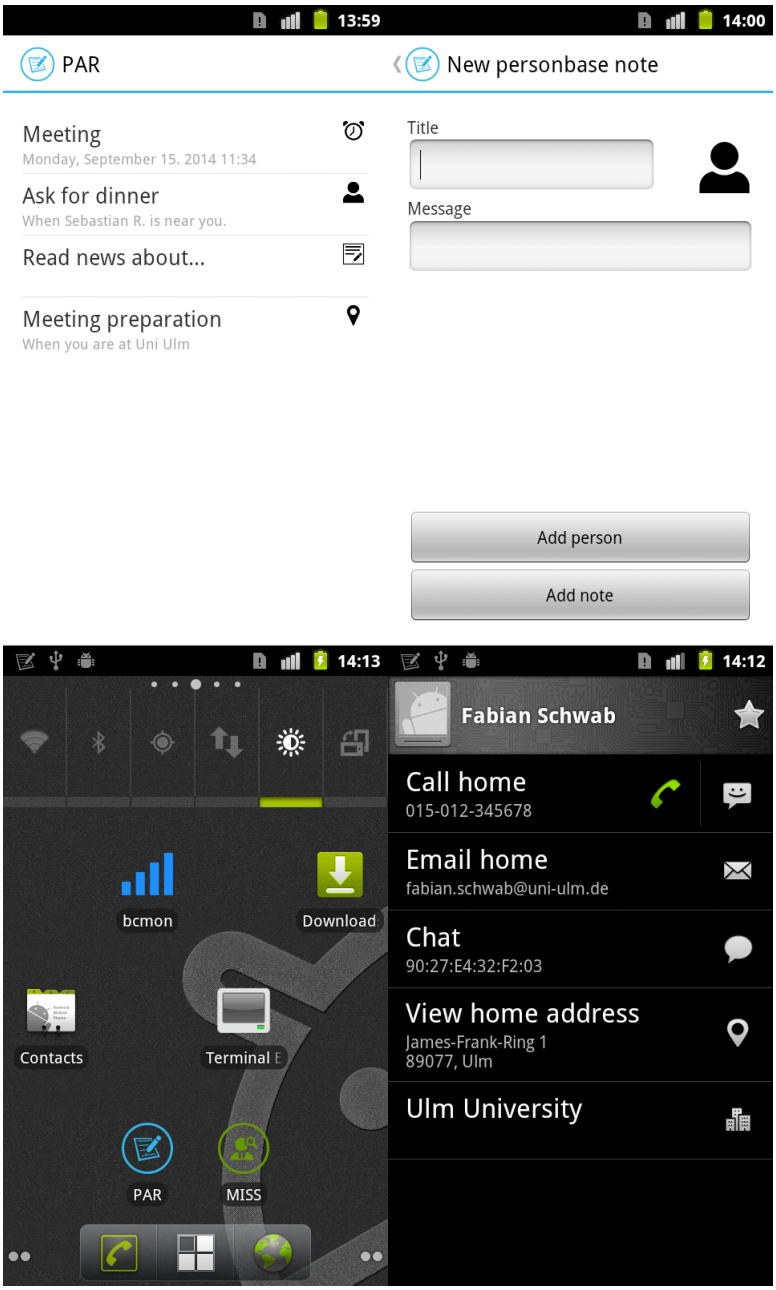


Abbildung 3.3: Screenshots der Anwendung PAR und Notifikation in der Statusleiste.