

SharkNet  
Systembeschreibung  
Version 0.0.3

Dustin Feurich

23. Dezember 2017



# Inhaltsverzeichnis

<b>1</b>	<b>Überblick</b>	<b>5</b>
1.1	Einleitung . . . . .	5
<b>2</b>	<b>Related Work</b>	<b>7</b>
<b>3</b>	<b>Bluetooth</b>	<b>11</b>
3.1	Aufgabe der Komponente . . . . .	11
3.2	Architektur . . . . .	11
3.2.1	Überlick . . . . .	11
3.2.2	Schnittstellendefinitionen . . . . .	12
3.3	Nutzung . . . . .	13
3.3.1	Code . . . . .	13
3.3.2	Deployment / Runtime . . . . .	13
3.4	Test . . . . .	13
3.5	Ausblick . . . . .	13
<b>4</b>	<b>WiFi</b>	<b>15</b>
4.1	Aufgabe der Komponente . . . . .	15
4.2	Architektur . . . . .	15
4.2.1	Überlick . . . . .	15
4.2.2	Schnittstellendefinitionen . . . . .	17
4.3	Nutzung . . . . .	18
4.3.1	Code . . . . .	18
4.3.2	Deployment / Runtime . . . . .	18
4.4	Test . . . . .	18
4.5	Ausblick . . . . .	18
<b>5</b>	<b>Sonstiges</b>	<b>19</b>



# Kapitel 1

## Überblick

### 1.1 Einleitung

Durch die rasante Entwicklung des Internet of Things (IoT) ist das Interesse an einen semantischen Datenaustausch spürbar gestiegen. Wurde in den letzten Jahrzehnten noch fast ausschließlich klassisch über die Zieladresse der Datenpakete geroutet, so werden jetzt auch die Metadaten dieser Datenpakete beim Routing zunehmend beachtet. Das Routing erfolgt hierbei also inhaltsbasiert und ermöglicht ein Routing nach den Interessen der Kommunikationsteilnehmer. Der Datenaustausch zwischen diesen Teilnehmern kann beim inhaltsbasierten Routing sowohl per klassischer Client-Server Architektur, als auch Peer-To-Peer (P2P) erfolgen. In dieser Arbeit wird der Datenaustausch über P2P erfolgen, was mehrere Vorteile bietet:

- Die Verbindungen zwischen Kommunikationsteilnehmern (Peers) können spontan aufgebaut werden, es wird keine Serverinfrastruktur benötigt.
- Die Daten liegen ausschließlich bei den Peers selbst. Da es keine Zwischenstation für die Datenpakete gibt, erhöht dies die Vertraulichkeit der Kommunikation immens.
- Nahezu alle Kommunikationsanwendungen verwenden das Internet um den Datenaustausch zu ermöglichen. Eine Verbindung mit dem Internet ist jedoch nicht zu jeder Zeit und an jedem Ort verfügbar. Weiterhin kann auch hier auf den Zwischenservern die Kommunikation gespeichert und an Dritte weitergegeben werden.



# Kapitel 2

## Related Work

Es gibt zahlreiche wissenschaftliche Paper, die Semantisches oder Inhaltsbasiertes Routing zum Thema haben. Viele diese Paper sind jedoch entweder schon mindestens zehn Jahre alt, oder beinhalten nicht exklusiv den Datenaustausch über Peer-To-Peer. Im Folgenden wird jeweils die Grundidee von vier Arbeiten vorgestellt, welche ausschließlich den semantischen Datenaustausch über P2P zum Inhalt haben.

Strassner et. al. präsentieren ein hybrides Routing, bei dem sowohl semantisch als auch traditionell geroutet wird. Die Peers bauen hierbei ein *small world* Netzwerk auf, bei dem jeder Peer viele kurze und nur wenige lange Verbindungen zu anderen Peers hat. Es werden zwei semantische Strukturen definiert - *node profiles* und *object profiles* - welche beide anhand von Metadaten beschrieben werden. Ein Interesse wird mit Hilfe des *node profiles* formuliert, das dann an die anderen Peers direkt geschickt wird. Interessiert sind die Peers an die Objekte. Durch eine semantische Ähnlichkeitsanalyse wird überprüft, ob ein Peer entweder direkt ein Objekt an den anfragende Peer liefert, oder ob er das *node profile* an andere Peers weiterleitet. Das *node profile* wird an den Peer weitergeleitet, bei dem die Ähnlichkeitsanalyse zwischen *node profile* und *object profile* am höchsten ist und sich außerdem physisch in Reichweite befindet.

David Faye et. al. stellen in Ihrer Ausarbeitung ein semantisches und abfrageorientiertes (Query) Routing vor. Die neuartige semantische Struktur ist hierbei die *expertise table*, in der mit Metadaten festgehalten ist, welcher Peer über welches Wissen verfügt. Anders als in Sharknet sind die Peers nicht gleichberechtigt, sondern in zwei Kategorien eingeteilt: normale Peers und Super-Peers. Ein Super-Peer verwaltet mehrere normale Peers und besitzt dafür eine *expertise table*. Sie reichen die Anfragen entweder an andere Super-Peers weiter oder lassen diese von normalen Peers auswerten. Ein Interesse wird mit Hilfe einer Anfrage gestellt, diese Anfrage wird durch den Routingalgorithmus an das relevante Ziel gesendet. Dies läuft folgendermaßen ab:

- Ein Peer formuliert sein Interesse mit einer Query und sendet diese an seinen zuständigen Super-Peer, der im Paper als *Godfather* bezeichnet wird.
- Der *Godfather* wertet nun mit der Query und den *expertise tables* aller verfügbaren anderen Super-Peers aus, an welche Super-Peers er die Query weiterreicht.
- Nachdem ein Super-Peer auf dieser Art eine Query erhalten hat, kann er diese nun entweder abermals an andere Super-Peers weiterleiten oder sie von einem seiner zugeordneten Peers ausführen lassen.
- Das Ergebnis der Ausführung wird nun an den eigentlichen Absender der Query zurückgeleitet.

Einen anderen Ansatz mit komplett gleichberechtigten Peers stellt Antonio Carzaniga et. al. vor, bei dem parallel zwei Protokolle ausgeführt werden. Dies umfasst zum einen das *Broadcast Routing Protocol* und zum anderen das *Content-based Routing Protocol*. Das Broadcast Protokoll ist für das physische Versenden der Nachrichten zwischen den Peers verantwortlich und baut eine Spanning-Tree Topologie auf. Die Nachricht wird zunächst ohne Einschränkung an alle Peers geschickt, die erreichbar sind. Das eigentliche Routing geschieht durch das *Content-based Routing Protocol*. Folgende semantische Strukturen werden benutzt:

- Eine *Message* besteht aus typisierten Attributen
- Ein *predicate* ist eine Disjunktion von Konjunktionen von Bedingungen (constraints), die sich auf einzelne Attribute beziehen
- Die *content-based forwarding table* enthält die von den Peers gesetzten *predicates*

Eine Funktion wertet anhand der *forwarding table* aus, an welche Peers die Nachricht weitergeleitet werden soll. Zusätzlich wird durch das Broadcast Protokoll ermittelt, welche Peers sich physisch in Reichweite befinden. Die Nachricht wird nun alle Peers geschickt, die in beiden Mengen vorkommen. Diese Funktionsweise ähnelt SharkNet, da in der Anwendung die Nachrichten ebenfalls per Broadcast verschickt werden. Die semantische Auswertung erfolgt in SharkNet jedoch durch Profile, die vom Nutzer dynamisch festgelegt werden können und nicht durch eine sich automatisch aufbauende Tabelle.

Luca Mottola et. al. haben eine sich selbst reparierende Baumtopologie entworfen, mit der inhaltsbasiertes Routing in mobilen Ad Hoc Netzwerken realisiert werden kann. Laut Mottola et. al. benötigt eine Topologie in Form eines Baums bei ad hoc Netzwerken eine stetige Selbstreparatur, die durch das dynamische Entfernen und Hinzufügen von mobilen Geräten notwendig sei. Diese Topologie wird während der Programmausführung auf den



Peers stetig angepasst, um auch bei einem häufigen Peerwechsel weiterhin benutzbar zu sein. Die Baumstruktur ist dabei für das inhaltsbasierte Routing essentiell. Das Routing erfolgt über das publish-subscribe Prinzip, wobei die Peers Nachrichten zu den Themen bekommen, die sie für die sie sich angemeldet (subscribt) haben.

Der wesentliche Unterschied zwischen den vorgestellten Veröffentlichungen und dieser Arbeit sind einerseits die Eingangs- und Ausgangsprofile, mit denen natürliche Personen eingehende und ausgehende Nachrichten semantisch filtern können und andererseits die Präsentation einer konkreten mobilen Applikation, die diese Art des Routings verwirklicht. Außerdem unterscheiden sich die dafür verwendeten semantischen Strukturen deutlich von anderen Veröffentlichungen.

Da diese Arbeit jedoch nicht nur das semantische Routing, sondern mit Sharknet auch ein dezentrales Netzwerk realisiert, soll an dieser Stelle kurz das bereits bekannte dezentrale soziale Netzwerk Diaspora vorgestellt werden.

Jeder Benutzer kann in Diaspora einen eigenen Server benutzen, welche als Pod bezeichnet werden. Diese Pods beinhalten die Benutzerdaten und werden vom Besitzer des Pods verwaltet. Der umfassende Datenschutz ist bei Diaspora jedoch nur dann gegeben, wenn jeder Benutzer auch einen eigenen Webserver benutzt, um damit seinen Pod zu hosten. In der Realität wird häufig aber kein eigener Webserver benutzt, außerdem ist die direkte Kommunikation zwischen den Pods nur eingeschränkt möglich. So lassen sich zum Beispiel keine Kontaktlisten von anderen Pods crawlen, auch wenn diese sie zur Verfügung stellen würden. Dies hat zur Folge, ein Teil der Benutzer sich ausschließlich mit anderen Pods verbinden, die dann zu Sammelpods werden.



# Kapitel 3

## Bluetooth

### 3.1 Aufgabe der Komponente

Die über SharkNet abgeschickten Nachrichten werden über Bluetooth übertragen. Die Komponente ist dabei ausschließlich für die kabellose Übertragung von Daten bzw. Nachrichten verantwortlich, die Ortung von potentiellen Kommunikationspartnern erfolgt über die Wifi-Direct Komponente. Auch die Filterung von bereits bekannten oder semantisch uninteressanten Nachrichten wird nicht innerhalb dieser Komponente, sondern innerhalb der Semantischen Routing Komponente vorgenommen.

Da es in SharkNet neben normalen Chats auch Gruppenchats und einen semantischen Broadcast gibt, erfordert der Datenaustausch mit Bluetooth kein Pairing der miteinander kommunizierenden Geräte. Dies trägt maßgeblich zur Benutzerfreundlichkeit bei, da insbesondere beim semantischen Broadcast sonst ständig Anfragen zum Pairing auf dem Gerät erscheinen würden und vom Benutzer zusätzliche Interaktionen erforderlich wären.

### 3.2 Architektur

#### 3.2.1 Überblick

Im folgenden UML-Klassendiagramm sind alle Bestandteile der Bluetooth Komponente von SharkNet abgebildet.

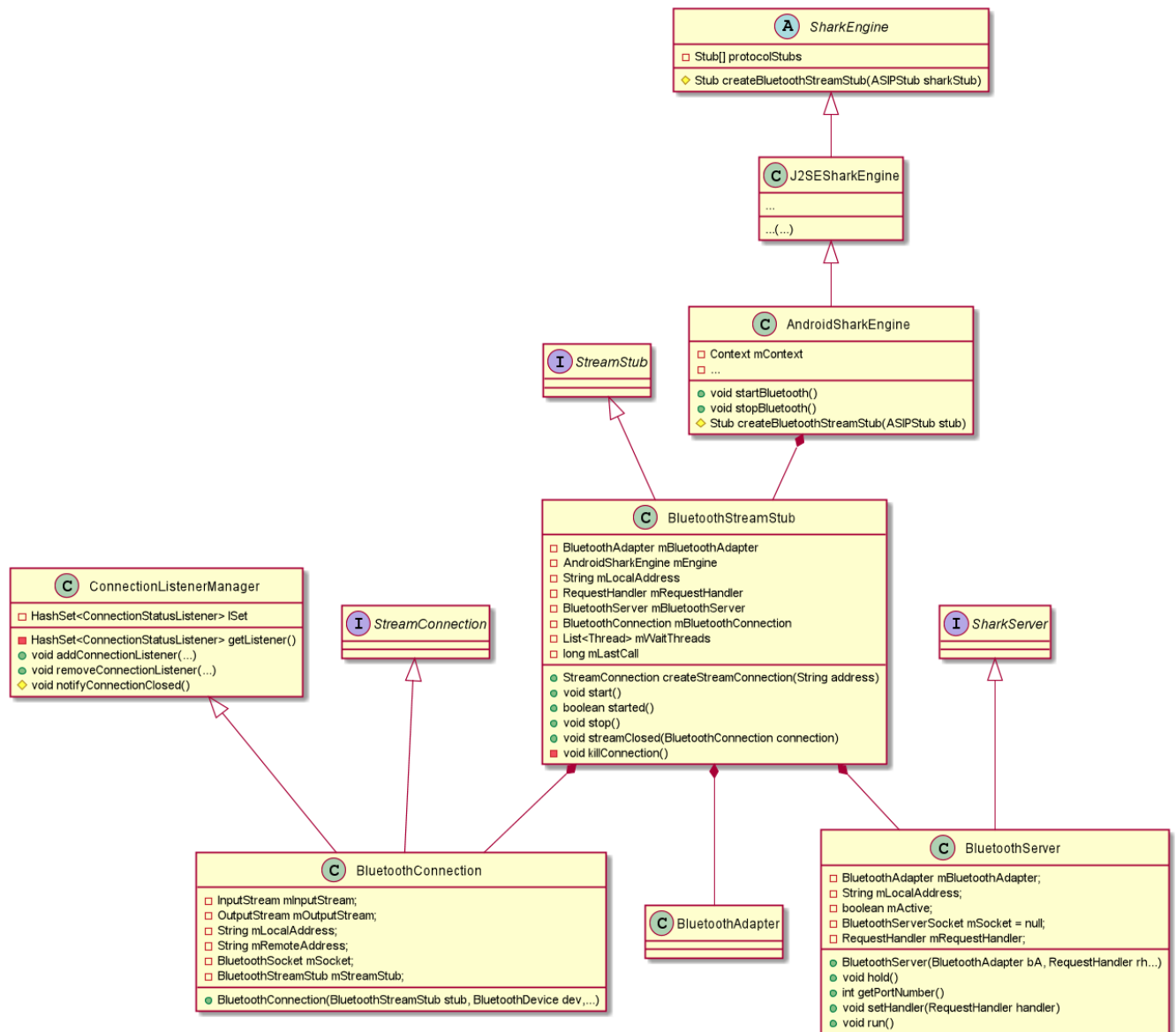


Abbildung 3.1: Die Bluetooth Klassen im Überblick

Im Zentrum dieser Hierarchie steht die Klasse *BluetoothStreamStub*. Eine Instanz dieser Klasse befindet sich als Attribut in der Klasse *AndroidSharkEngine*, von der aus alle Protokolle wie NFC, Wifi-Direct oder Bluetooth gesteuert werden. Sie stellt daher auch Methoden wie *startBluetooth()* oder *stopBluetooth()* bereit.

### 3.2.2 Schnittstellendefinitionen

Anhand der Klassenhierarchie der Bluetooth-Komponente lässt sich erkennen, dass die folgenden drei Schnittstellen implementiert werden:

- *StreamStub*: Mit Hilfe von Implementierungen dieses Interfaces können streamba-

sierte Ende-zu-Ende Verbindungen zwischen zwei Geräten hergestellt werden. Die Klasse *BluetoothStreamStub* öffnet und schließt daher die Verbindungen zu anderen Geräten per Bluetooth.

- *StreamConnection*: Das Shark Framework definiert mit dem Interface *StreamConnection* das Verhalten einer streambasierten Verbindung zweier Geräte. Dieses Interface ist nicht zu verwechseln mit gleichnamigen Interface von Java ME. Klassen wie *BluetoothConnection*, welche dieses Interface implementieren, bauen in ihren jeweiligen Konstruktor die Verbindung mit ihrem jeweiligen Protokoll auf. In der Klasse *BluetoothConnection* erfolgt dies über das Bluetooth-Protokoll RFCOMM.
- *SharkServer*: Eine dieses Interface implementierende Klassen wartet bei der bestehenden Verbindung auf Datenpakete, nimmt diese an und leitet sie an einen *RequestHandler* weiter. Die Klasse *BluetoothServer* nimmt daher die Datenpakete an, die per bestehender Bluetoothverbindung eintreffen.

## 3.3 Nutzung

### 3.3.1 Code

Der Code dieser Komponente kann hier <https://github.com/SharedKnowledge/SharkNet-Api-Android/tree/master/api/src/main/java/net/sharksystem/api/shark/protocols/bluetooth> betrachtet werden. Wie auch die anderen Implementierungen von Übertragungsprotokollen, befindet sich auch die Bluetooth-Implementierung im Projekt *SharkNet-Api-Android* im Package *protocols*.

### 3.3.2 Deployment / Runtime

## 3.4 Test

## 3.5 Ausblick

Es ist empfehlenswert, die von Android gestellten Bluetooth Klassen durch die dazu äquivalenten Bluetooth Low Energy (BLE) Klassen entweder zu ersetzen oder zumindest eine Alternative zu dem klassischen Bluetooth Package zu bieten. BLE verbraucht weniger Akkuleistung als das klassische Bluetooth, kann dafür aber nur eine geringere Menge an Daten pro Verbindung unterstützen. Da die mit SharkNet verschickten Nachrichten

auch trotz der semantischen Annotationen nur wenige Kilobyte benötigen, stellt dies für SharkNet kein Hindernis dar.

# Kapitel 4

## WiFi

### 4.1 Aufgabe der Komponente

Über die WiFi-Direct Komponente vermitteln die Peers ihre Kontaktdaten an alle verfügbaren Peers in der Nähe. Dies geschieht über den Expose Befehl des ASIP Protokolls, bei dem ein ASIP-Interesse an die Wissensbasis von anderen Peers gesandt wird. Dies beinhaltet unter anderem die Bluetooth MAC-Adresse, mit der dem Peer dann anschließend Nachrichten per Bluetooth geschickt werden können. Das Verschicken der Bluetooth Mac-Adresse via WiFi-Direct ermöglicht es daher, dass für die darauf folgende Bluetooth-Verbindung kein Pairing benötigt wird.

Die Komponente ist der elementare Bestandteil des Peer-Radars, der alle sich in der Nähe befindlichen Peers anzeigt und die Kommunikation mit diesen erlaubt. Das Radar ist wiederum dafür erforderlich, neue Chats mit Peers anzulegen oder einen semantischen Broadcast ohne Bluetooth-Pairing zu ermöglichen.

### 4.2 Architektur

#### 4.2.1 Überblick

Im folgenden UML-Klassendiagramm sind alle Bestandteile der WiFi-Direct Komponente von SharkNet abgebildet.

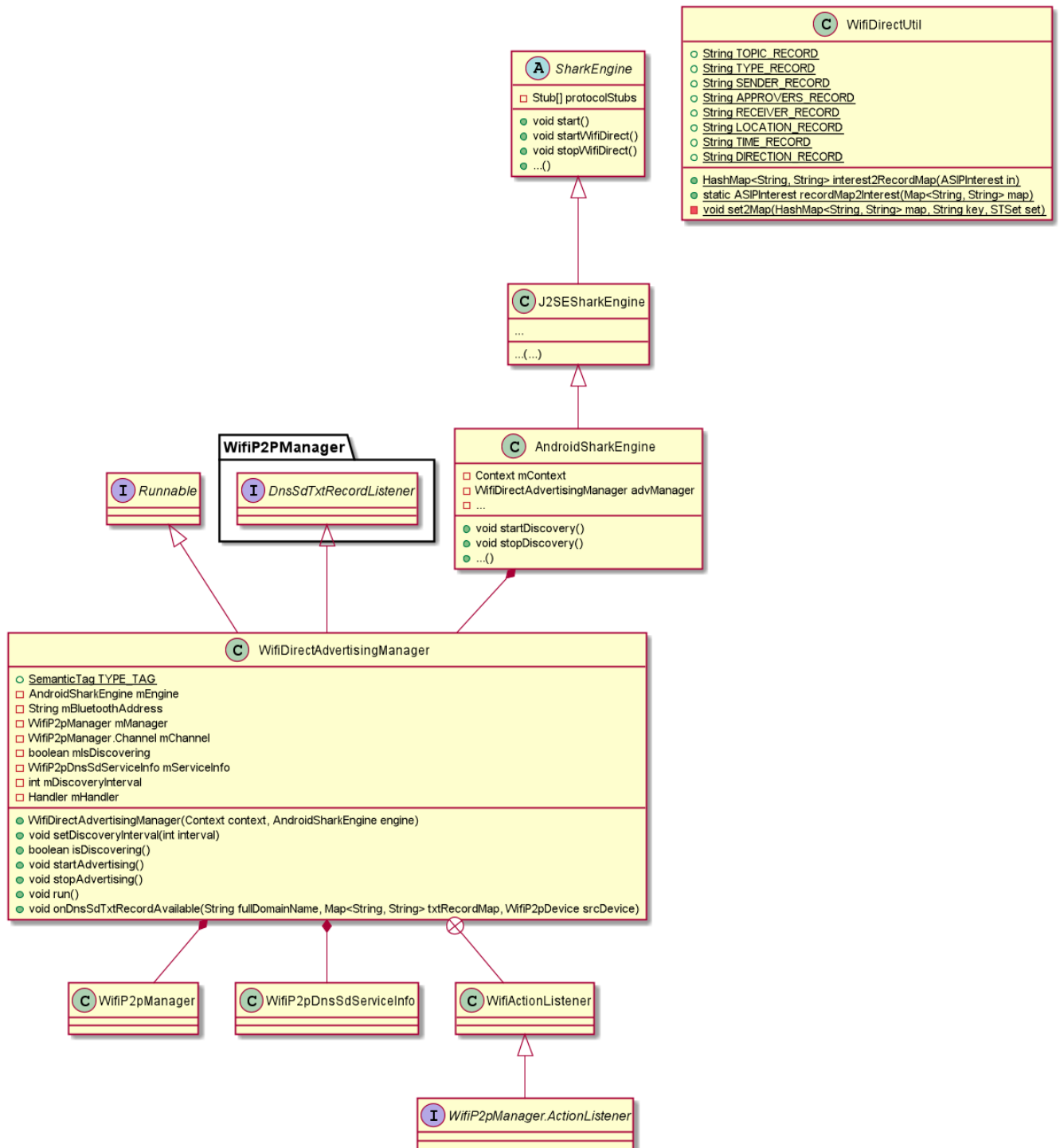


Abbildung 4.1: Die WiFi-Direct Klassen im Überblick

Im Zentrum dieser Hierarchie steht die Klasse *WifiDirectAdvertisingManager*. Eine Instanz dieser Klasse befindet sich als Attribut in der Klasse *AndroidSharkEngine*, von der aus alle Protokolle wie NFC, Wifi-Direct oder Bluetooth gesteuert werden. Über die Engine kann daher auch das Radar per *startDiscovery()* Methode gestartet oder über



die *stopDiscovery()* Methode beendet werden. Das Starten oder Stoppen der kompletten WiFi-Komponente erfolgt dagegen in der Klasse *SharkEngine*, die den Ausgangspunkt der Vererbungshierarchie darstellt.

Die Klasse *WifiDirectUtil* bietet statische Methoden an, mit denen ASIP-Interessen in Hashmaps umgewandelt werden können und umgekehrt. Dies ist notwendig, da die von Android gestellte Basisklasse *WifiP2PManager* bei der Anmeldungen von Services keine ASIP-Interessen, sondern Hashmaps als Parameter akzeptiert.

### 4.2.2 Schnittstellendefinitionen

Wie im vorherigen Unterkapitel erläutert liefern die beiden Methoden *startDiscovery()* und *stopDiscovery()* die Funktionalität, um Peers zu finden und andere Peers über das eigene Interesse in Kenntnis zu setzen.

Bei Aufruf der *startDiscovery()* Methode wird innerhalb der Engine ein neuer *WifiDirectAdvertisingManager* angelegt und anschließend dessen *startAdvertising()* Methode aufgerufen. Innerhalb der *startAdvertising()* Methode wird sich nun auf der dritten Schicht des OSI-Modells begeben, wie der folgende Codeausschnitt zeigt:

Listing 4.1: Peer Semantic Tag

```

1  HashMap<String , String> map = WifiDirectUtil.interest2RecordMap(
    interest );
2  mServiceInfo = WifiP2pDnsSdServiceInfo.newInstance( "_sbc" , "_presence
    . _tcp" , map );
3  mManager.addLocalService( mChannel , mServiceInfo , new
    WifiActionListener( "Add_LocalService" ) );
4  mManager.clearServiceRequests( mChannel , new WifiActionListener( "Clear
    _ServiceRequests" ) );
5  WifiP2pDnsSdServiceRequest wifiP2pDnsSdServiceRequest =
    WifiP2pDnsSdServiceRequest.newInstance();
6  mManager.addServiceRequest( mChannel , wifiP2pDnsSdServiceRequest , new
    WifiActionListener( "Add_ServiceRequest" ) );
```

Nachdem in der erste Zeile eine Hashmap auf dem Interesse erzeugt worden ist, wird diese Hashmap in Zeile zwei als Parameter für die Erzeugung einer Service Information benutzt. Anschließend wird dem *WifiP2PManager* ein neuer lokaler Service hinzugefügt, wobei dieser Service die zuvor erzeugte Service Information enthält. Nachdem etwaige vorherige Service Requests beseitigt worden sind, wird der neue WifiP2P Service Request hinzugefügt. Dadurch wird nun an alle Geräte in der Nähe, die auf WifiP2P Service Requests warten, dieser zur Verfügung gestellt.

Neben dem Hinzufügen von Services, müssen diese aber auch empfangen und ausgewertet werden. Dies ist der Grund, warum der *WifiDirectAdvertisingManager* das Interface *Runnable* implementiert. In der dadurch implementierten Methode *run()* werden die von anderen Geräten gesendeten Service Requests empfangen.

Listing 4.2: Peer Semantic Tag

```

1 mManager.discoverServices(mChannel, new WifiActionListener("Discover_
    Services"));
2 mHandler.postDelayed(this, mDiscoveryInterval);

```

Sollte ein Service gefunden und erfolgreich eine Peer-To-Peer Verbindung zwischen zwei Geräten aufgebaut werden können, wird nun die aus Listing x.x bekannte Hashmap an das Gerät gesendet, welches den Service gefunden (discovered) hat. Dabei wird automatisch die Methode *onDnsSdTxtRecordAvailable* aufgerufen, welche die empfangene Hashmap in ein ASIP-Interesse umwandelt und dann der Engine weiterreicht.

Listing 4.3: Peer Semantic Tag

```

1 ASIPInterest interest = WifiDirectUtil.recordMap2Interest(
    txtRecordMap);
2 mEngine.handleASIPInterest(interest);

```

## 4.3 Nutzung

### 4.3.1 Code

Der Code dieser Komponente kann hier <https://github.com/SharedKnowledge/SharkNet-Api-Android/tree/master/api/src/main/java/net/sharksystem/api/shark/protocols/wifidirect> betrachtet werden. Wie auch die anderen Implementierungen von Übertragungsprotokollen, befindet sich auch die WiFi-Direct-Implementierung im Projekt *SharkNet-Api-Android* im Package *protocols*.

### 4.3.2 Deployment / Runtime

## 4.4 Test

## 4.5 Ausblick

# Kapitel 5

## Sonstiges

In der folgenden Grafik sind alle Bestandteile der WifiDirect Komponente von SharkNet abgebildet.

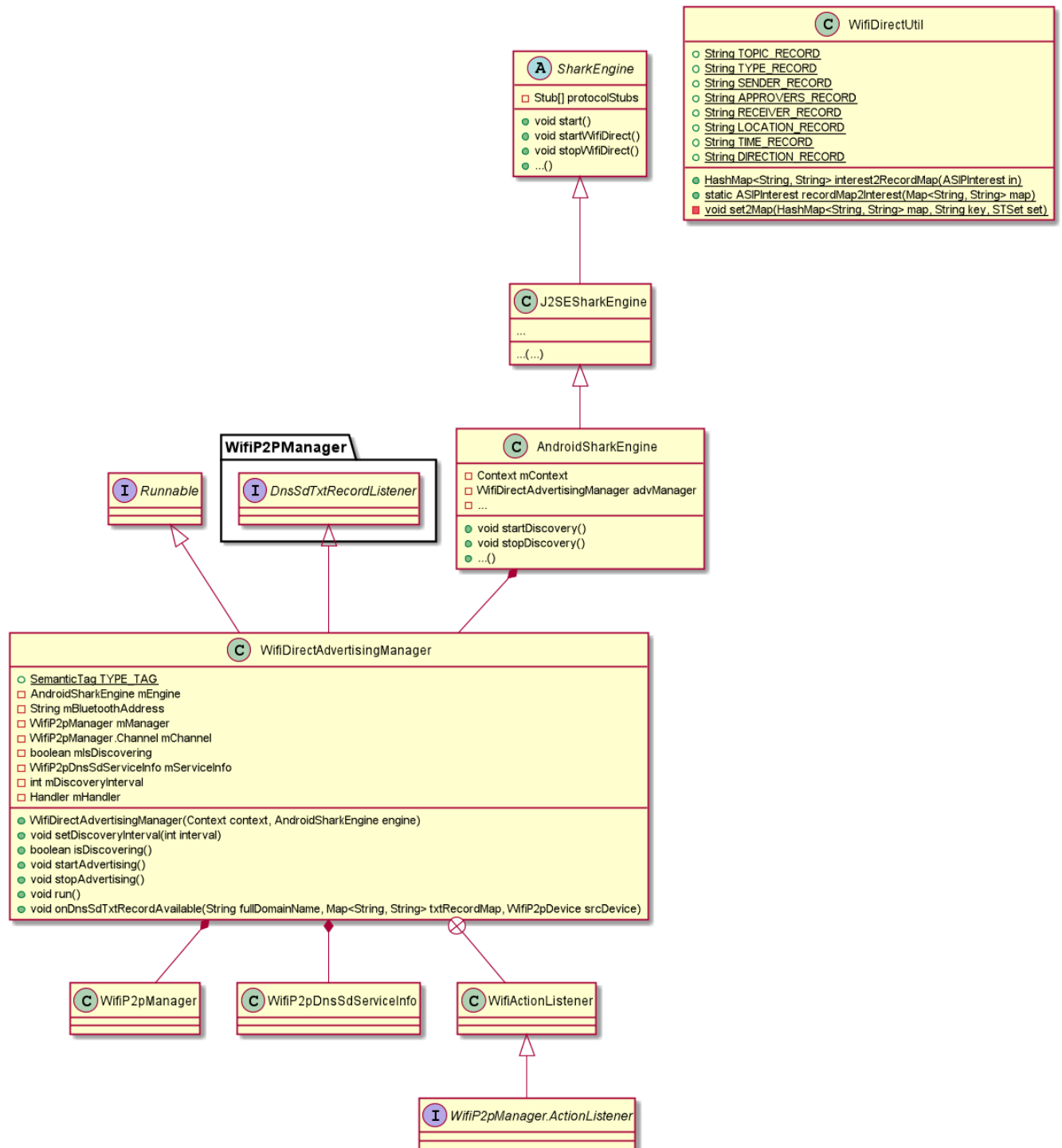


Abbildung 5.1: Die WifiDirect Klassen im Überblick

In der folgenden Grafik sind alle Bestandteile der Radar Komponente abgebildet.

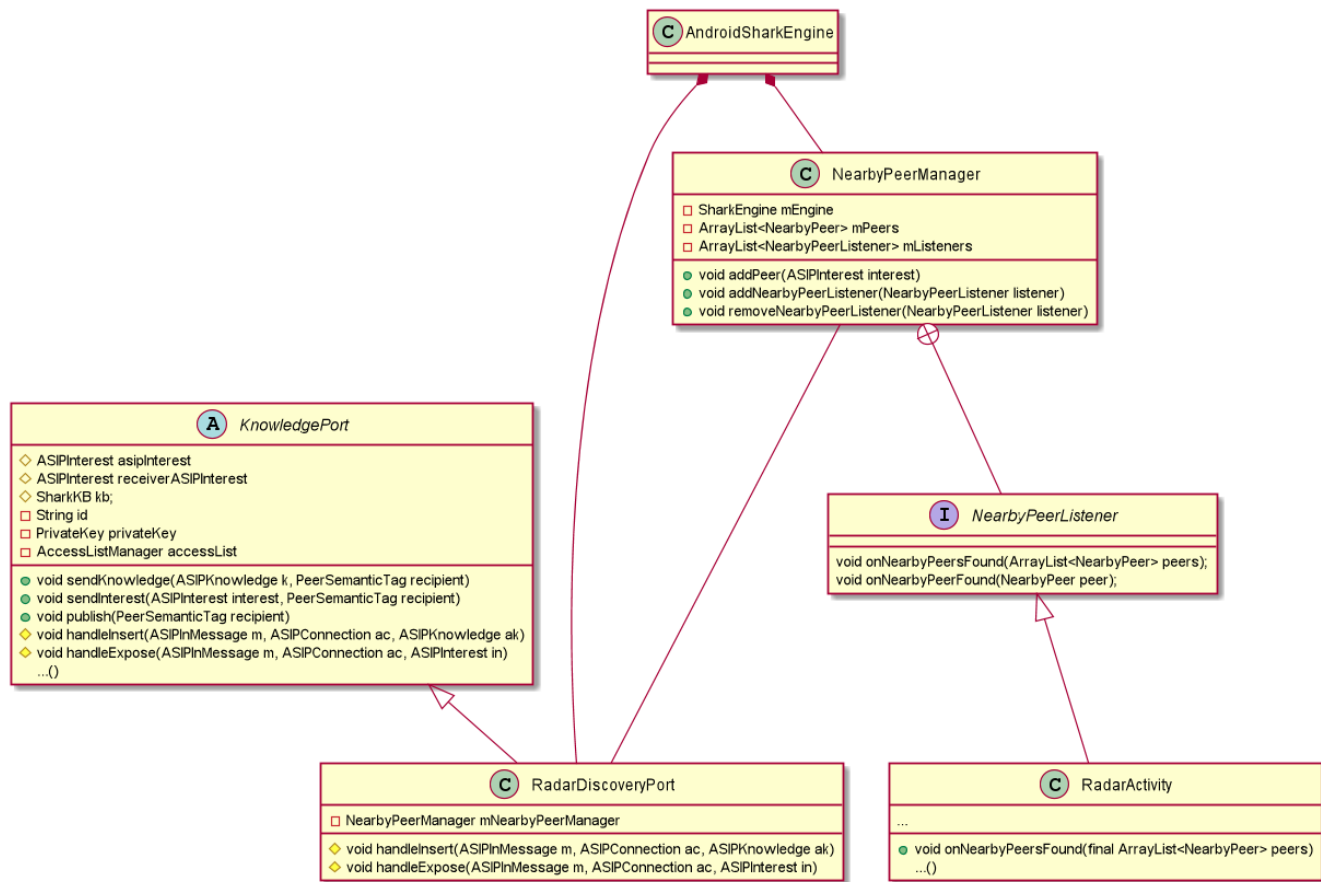


Abbildung 5.2: Die Radar Klassen im Überblick

Im folgenden Aktivitätsdiagramm wird das Versenden von Nachrichten per Broadcast abgebildet

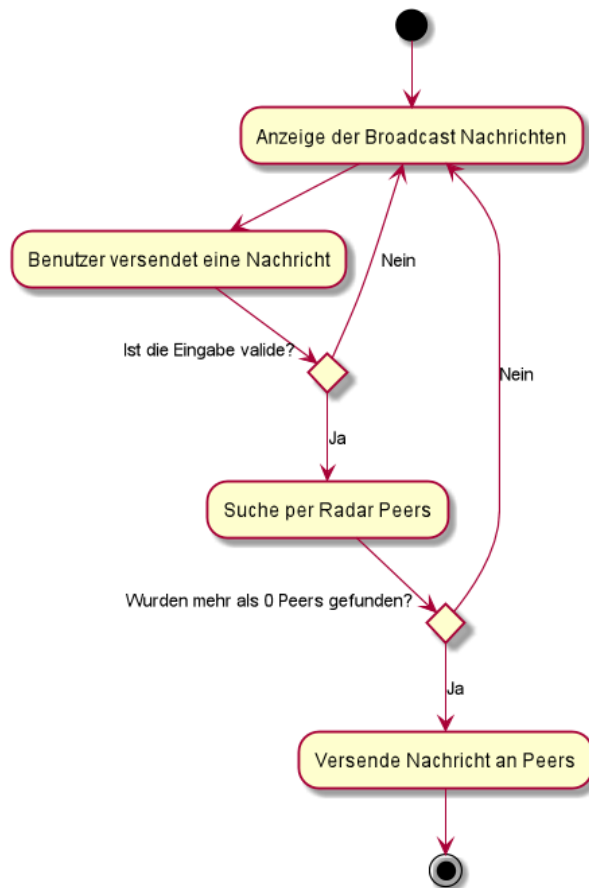


Abbildung 5.3: Versenden von Nachrichten per Broadcast in SharkNet

Im folgenden Aktivitätsdiagramm wird das Empfangen von Nachrichten per Broadcast abgebildet

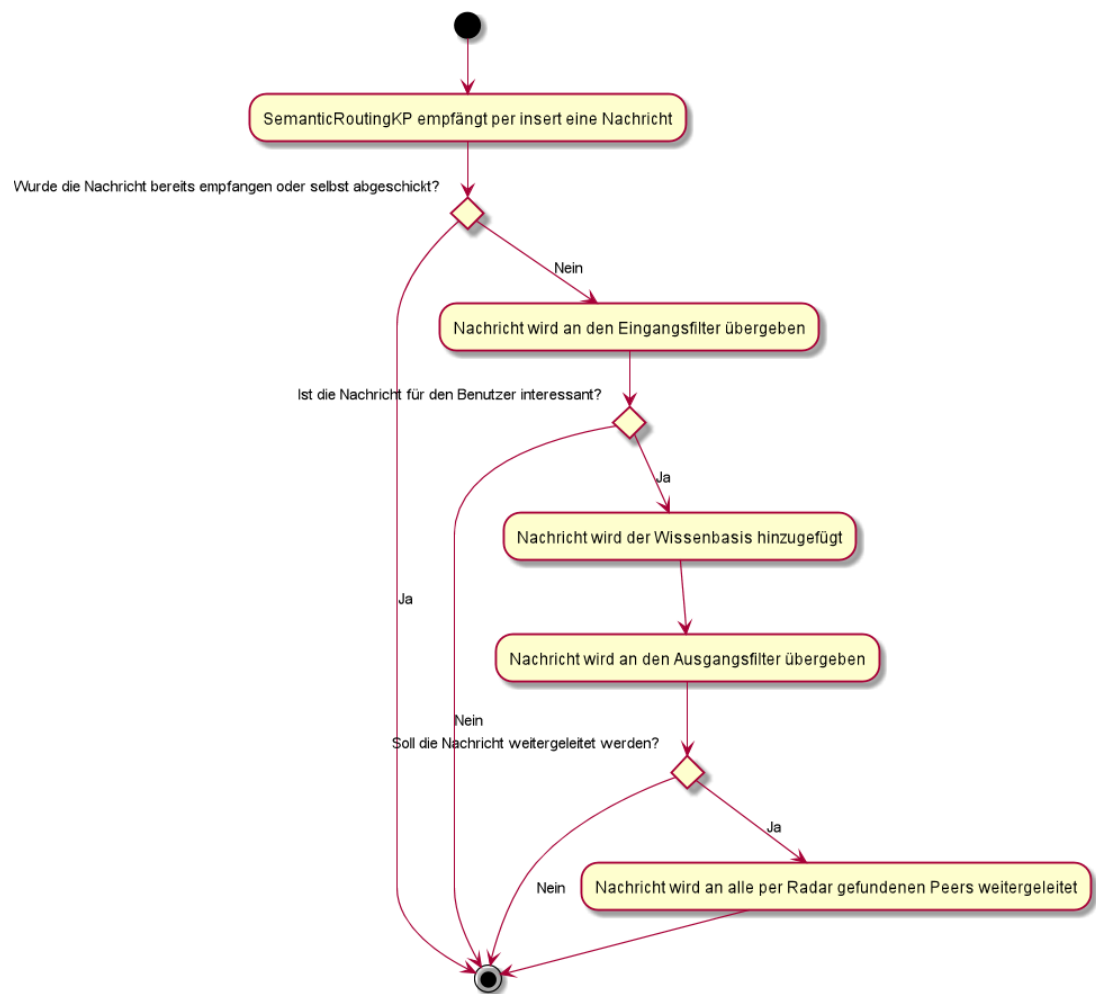


Abbildung 5.4: Empfangen von Nachrichten per Broadcast in SharkNet

Im folgenden Aktivitätsdiagramm wird Filterung von Nachrichten per Eingangsfiler abgebildet



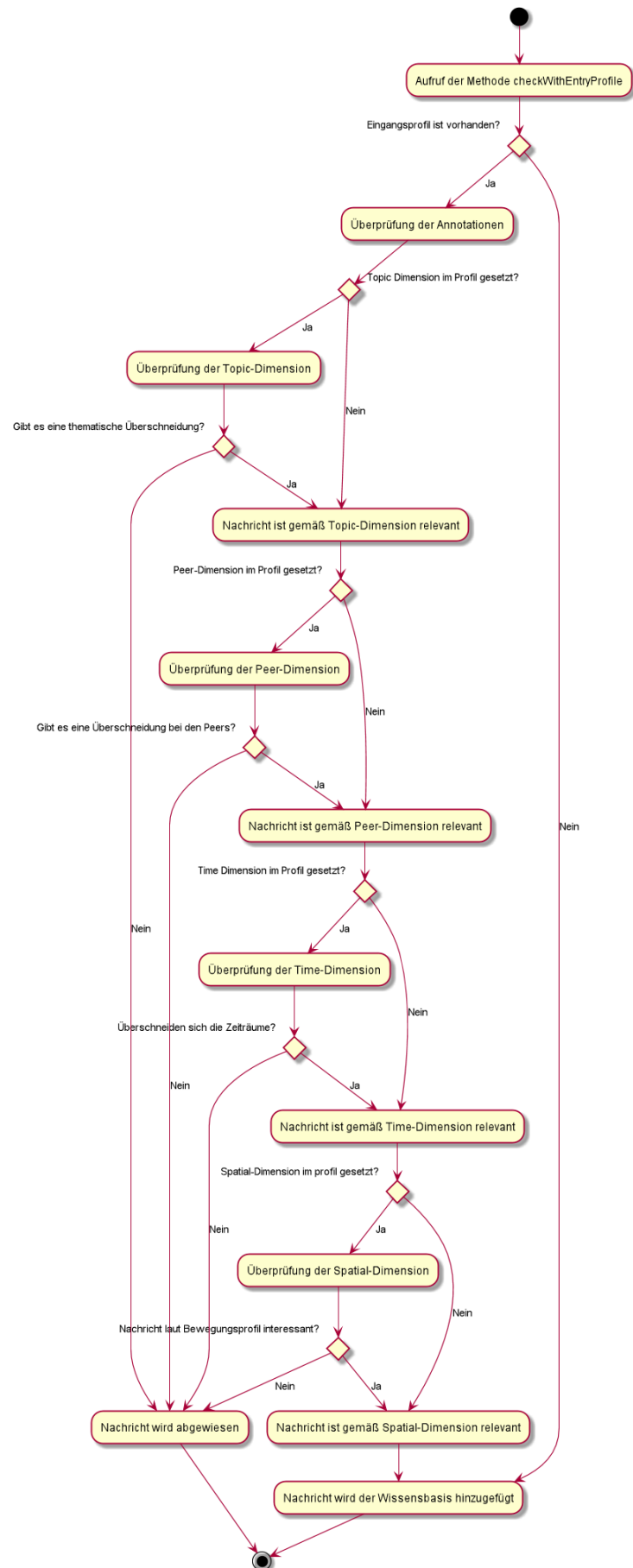


Abbildung 5.5: Filterung von Nachrichten per Eingangsfilter in SharkNet

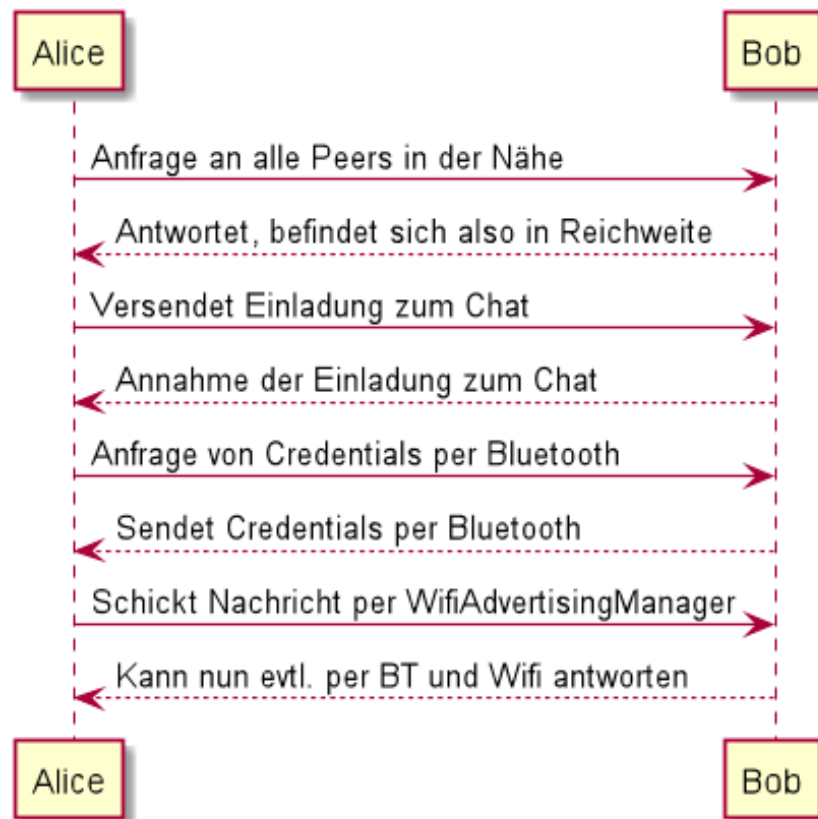


Abbildung 5.6: Kommunikation per Chat