SharkNet
Systembeschreibung
Version 0.0.3

Dustin Feurich

19. Dezember 2017

Inhaltsverzeichnis

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 Kommunkationsanwendungen 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

Bluetooth

2.1 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 Kommunikationspartern 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.

2.2 Architektur

2.2.1 Überlick

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

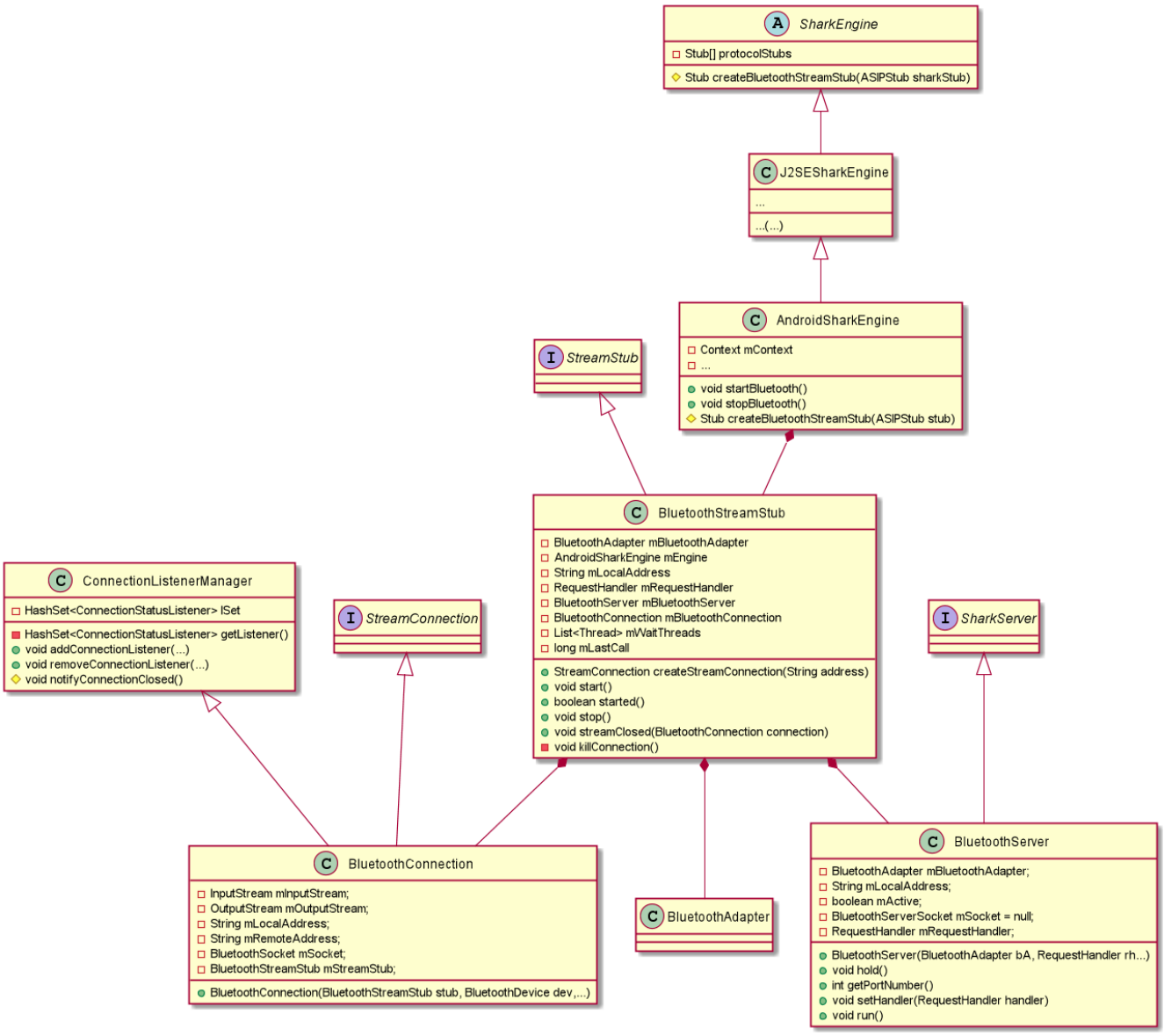

Abbildung 2.1: Die Bluetooth Klassen im Überblick

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.

2.2.2 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 Konstruktur 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 *Request Handler* weiter. Die Klasse *BluetoothServer* nimmt daher die Datenpakete an, die per bestehender Bluetoothverbindung eintreffen.

2.3 Nutzung

2.3.1 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*.

2.3.2 Deployment / Runtime

2.4 Test

2.5 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 3

WiFi

3.1 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.

3.2 Architektur

3.2.1 Überlick

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

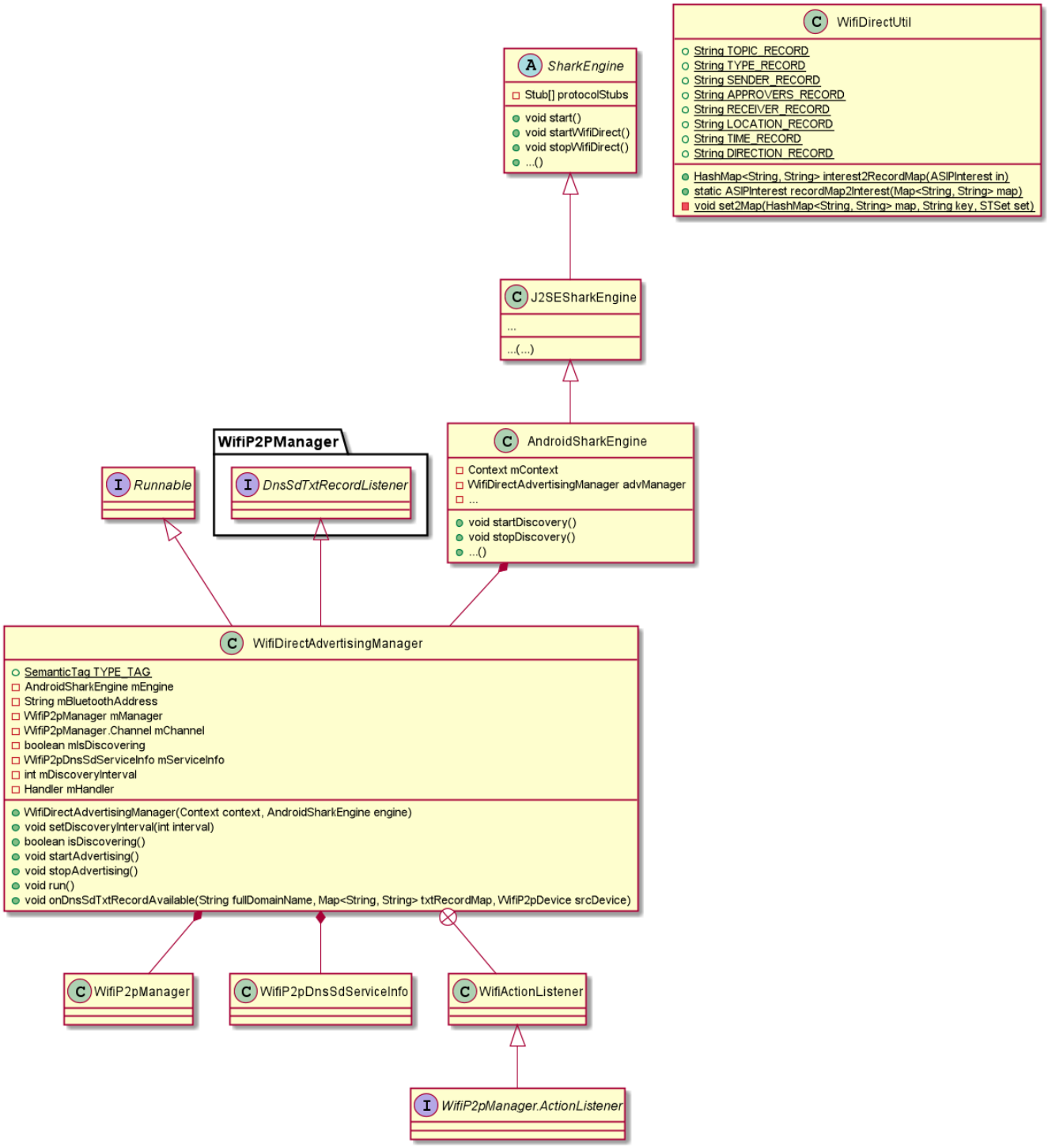

Abbildung 3.1: Die WiFi-Direct Klassen im Überblick

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.

3.2.2 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 3.1: 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 3.2: 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 3.3: Peer Semantic Tag

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

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/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*.

3.3.2 Deployment / Runtime

3.4 Test

3.5 Ausblick

Kapitel 4

Sonstiges

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

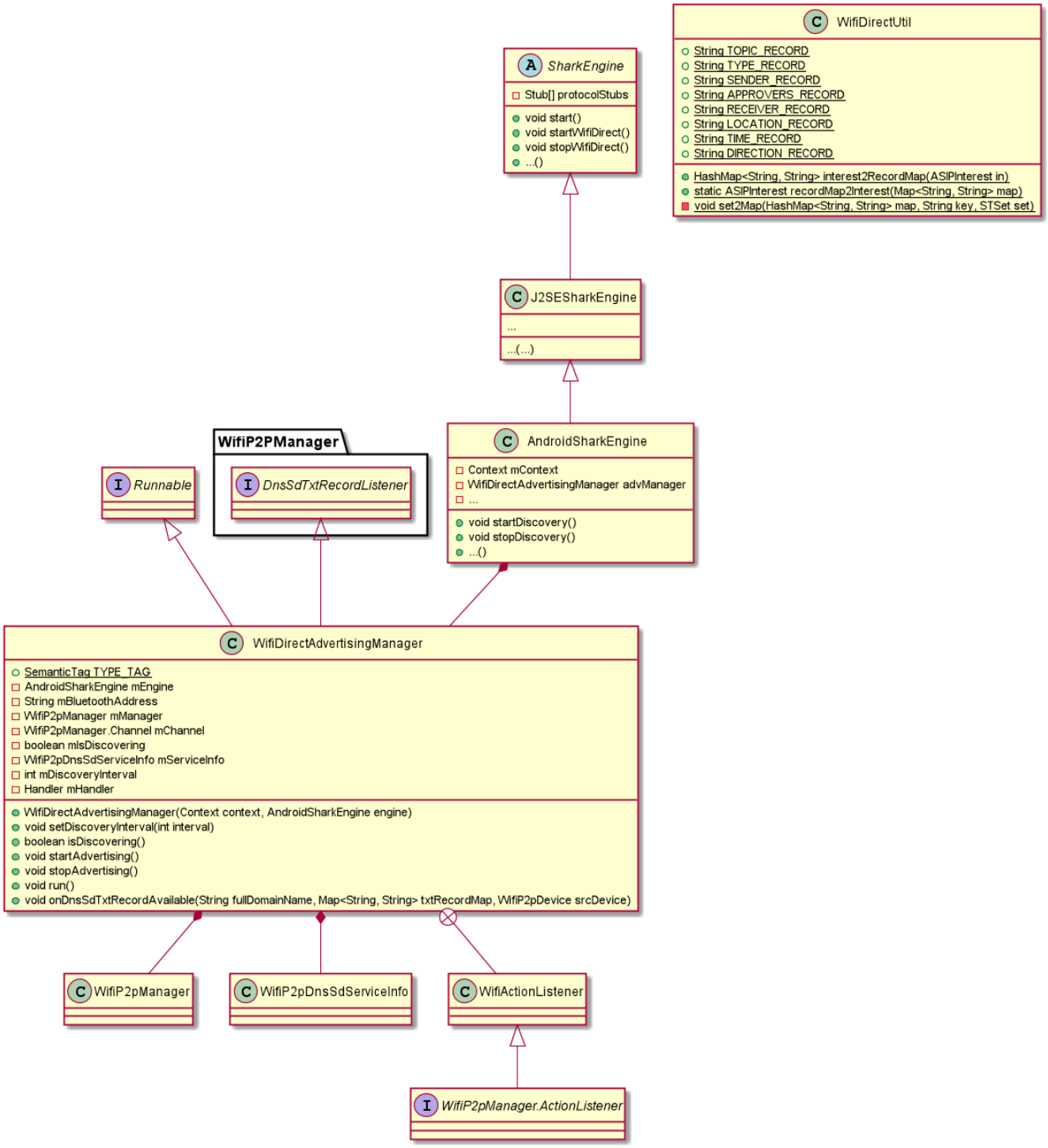

Abbildung 4.1: Die WifiDirect Klassen im Überblick

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

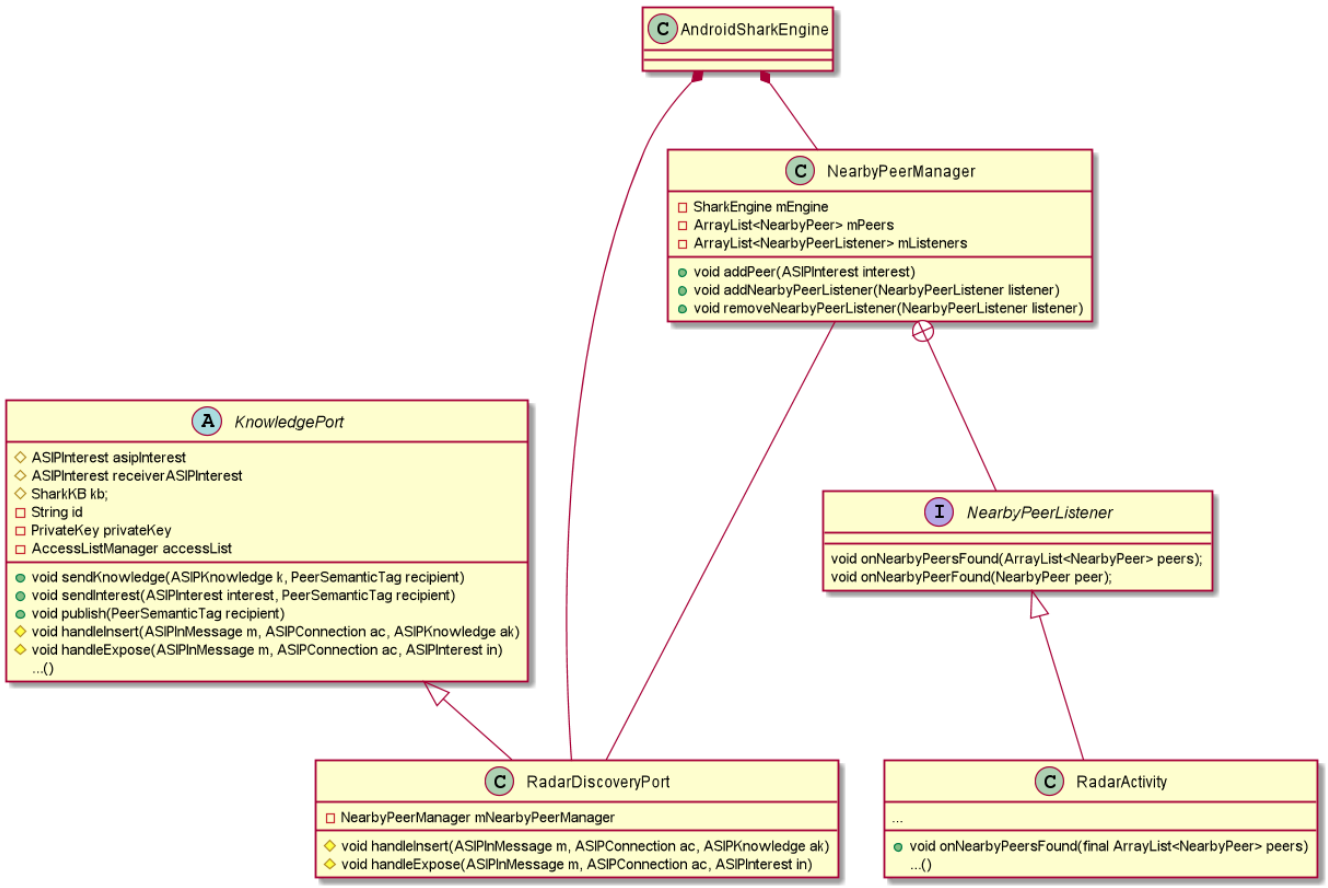

Abbildung 4.2: Die Radar Klassen im Überblick

Im folgenden Aktivitätsdiagramm wird das Versenden von Nachrichten per Broadcast abgebildet

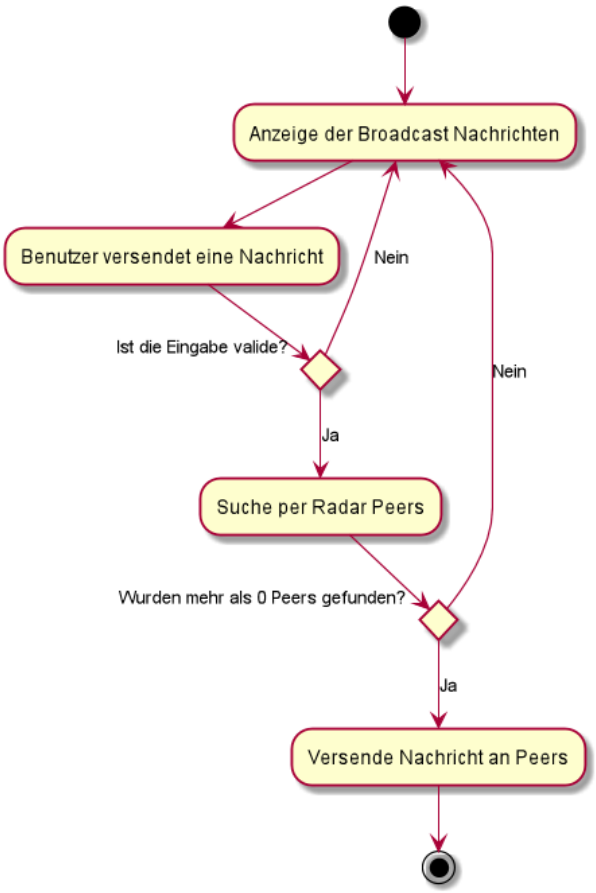

Abbildung 4.3: Versenden von Nachrichten per Broadcast in SharkNet

Im folgenden Aktivitätsdiagramm wird das Empfangen von Nachrichten per Broadcast abgebildet

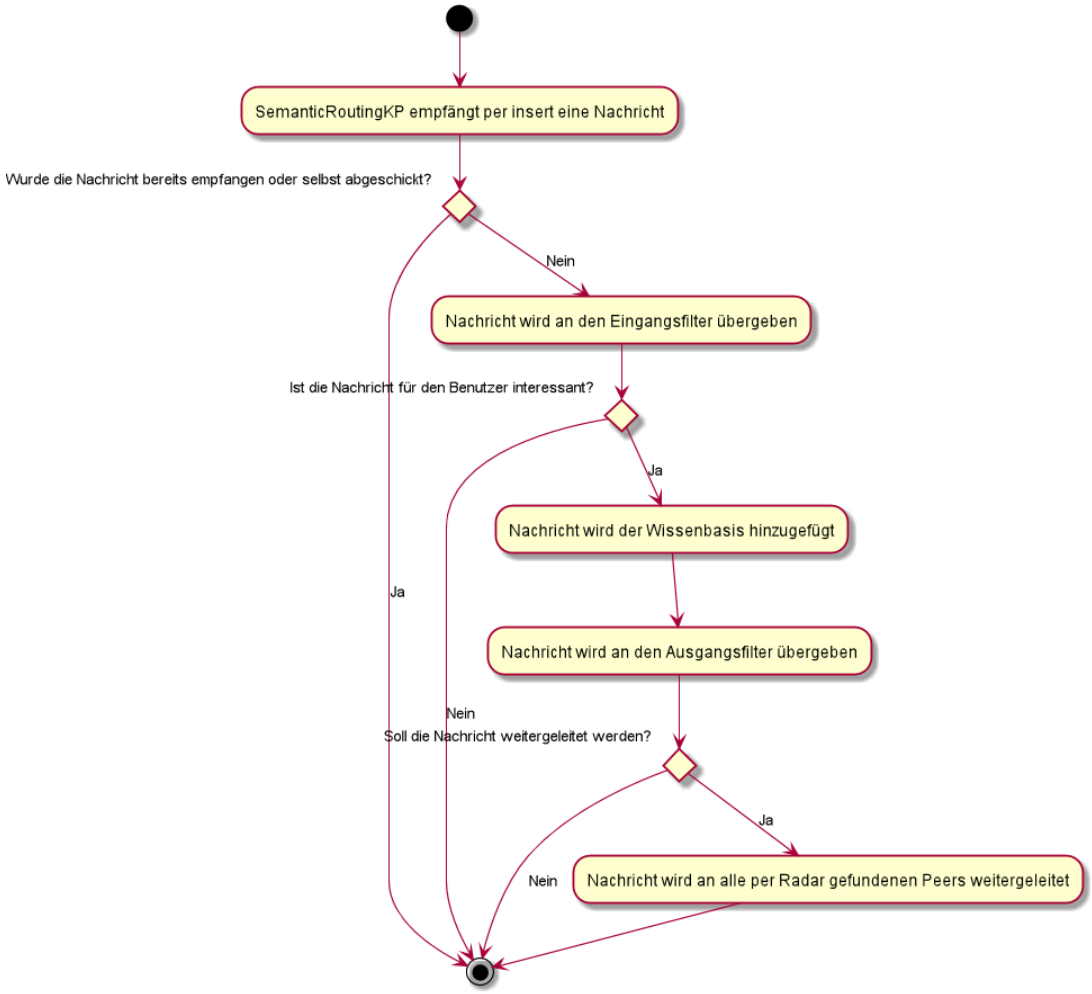

Abbildung 4.4: Empfangen von Nachrichten per Broadcast in SharkNet

Im folgenden Aktivitätsdiagramm wird Filterung von Nachrichten per Eingangsfilter abgebildet

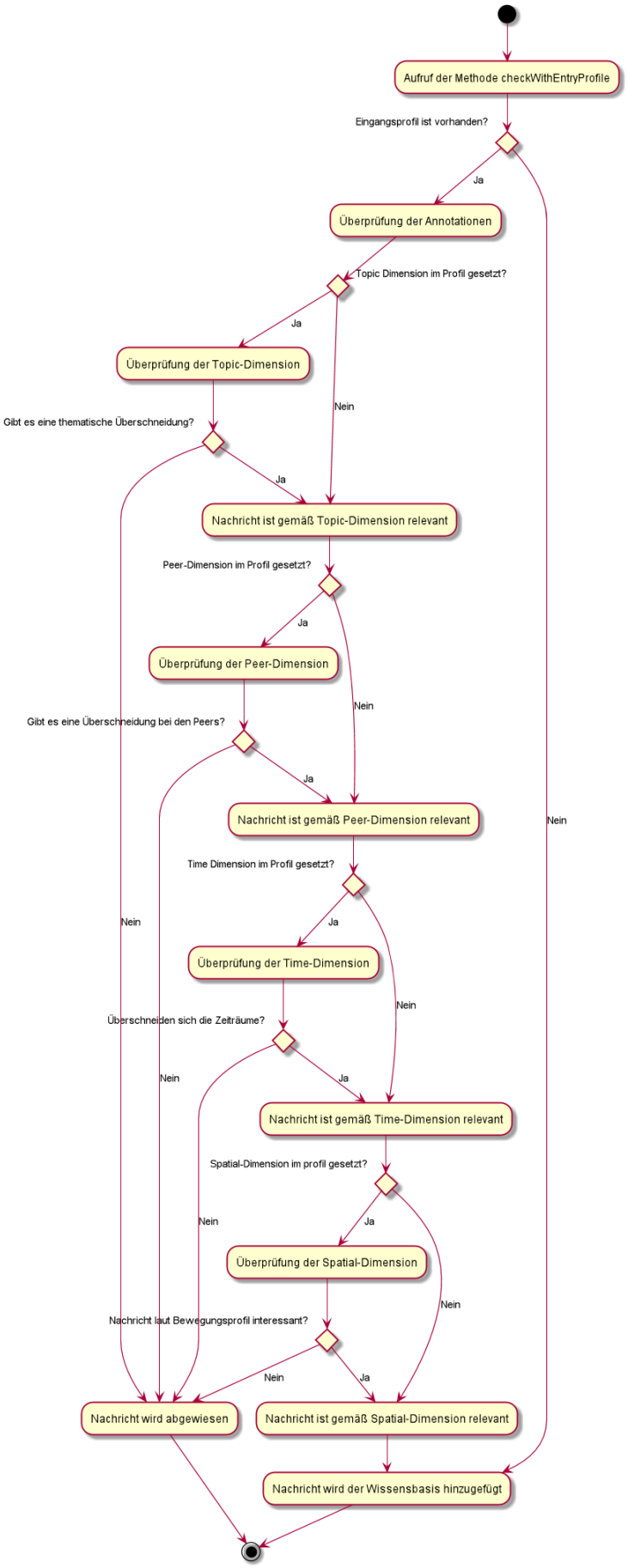

Abbildung 4.5: Filterung von Nachrichten per Eingangsfilter in SharkNet

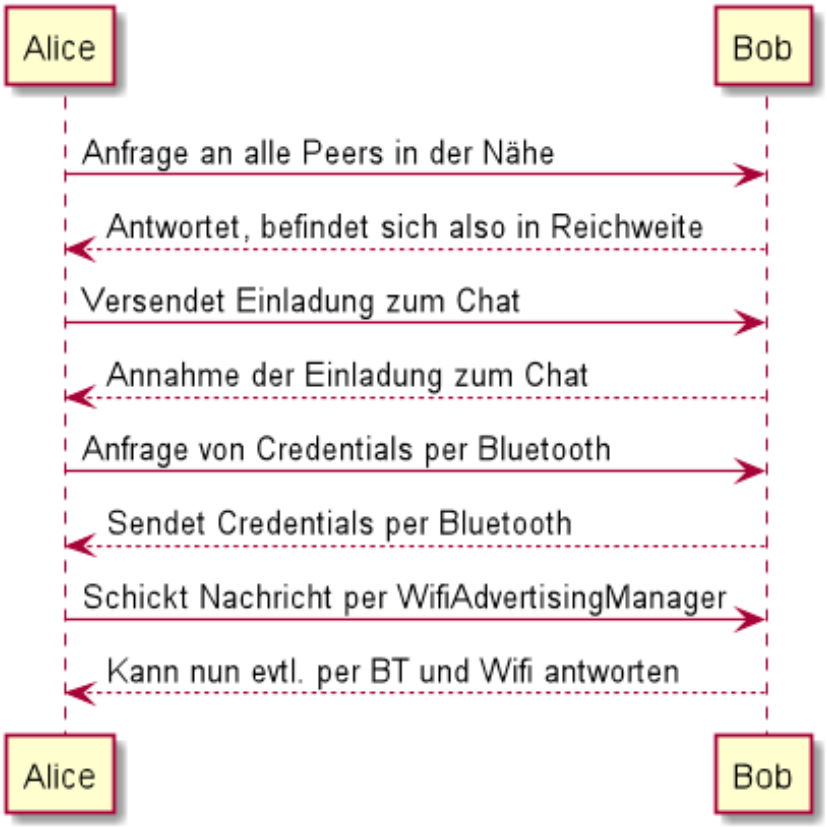

Abbildung 4.6: Kommunikation per Chat