



# Drone meshnetwerk simulatie

Software Design Document

Versie 2.0

Alten Nederland B.V.

Hogeschool van Arnhem en Nijmegen

HBO Technische Informatica - Embedded Software Developement

MWJ.Berentsen@student.han.nl

Studentnummer: 561399

Docent: J. Visch, MSc

Assessor: ir. C.G.R. van Uffelen

M.W.J. Berentsen

13 juni 2019

# Inhoudsopgave

<b>1</b>	<b>Inleiding</b>	<b>3</b>
1.1	Leeswijzer . . . . .	3
<b>2</b>	<b>Architectural Overview</b>	<b>5</b>
2.1	Hoofdgroepen . . . . .	5
2.2	Interfaces . . . . .	7
2.2.1	IInternetConnection . . . . .	7
2.2.2	IGatewayCommands . . . . .	7
2.2.3	IWirelessCommunication . . . . .	7
2.2.4	IMeshNetwork . . . . .	7
2.2.5	IMeshDebugInfo . . . . .	7
2.2.6	NRF24HighLevelInterface . . . . .	7
2.2.7	NRF24LowLevelInterface . . . . .	7
2.2.8	IDroneEngine . . . . .	8
2.2.9	IRoutingTechnique . . . . .	8
2.3	Ros topics en services . . . . .	8
2.3.1	RequestGatewayDroneFlight . . . . .	8
2.3.2	CasusRequest . . . . .	8
2.3.3	PowerSwitch . . . . .	8
2.3.4	WirelessMessageRequest . . . . .	8
2.3.5	AreaScanRequest . . . . .	8
2.3.6	RequestGPS . . . . .	9
2.3.7	RequestGatewayDroneFlight . . . . .	9
2.3.8	NodeDebugInfo . . . . .	9
2.3.9	NRF24 . . . . .	9
2.3.10	Location . . . . .	9
2.3.11	DroneInfo . . . . .	9
2.3.12	RestCommands . . . . .	9

<b>3 Detailed Design Description</b>	<b>10</b>
3.1 Deployment Diagram . . . . .	10
3.1.1 Ontwerpkeuzes met betrekking tot deployment . . . . .	10
3.2 Design Sub-Systeem Communicatie . . . . .	10
3.2.1 Component Diagram . . . . .	11
3.2.2 Interfaces . . . . .	13
3.2.3 Sequence Diagrams . . . . .	17
3.2.4 Activity Diagrammen . . . . .	21
3.2.5 Ontwerpkeuzes gemaakt voor het communicatie component . . . . .	29
3.3 Design Sub-Systeem Gazebo . . . . .	29
3.3.1 Component Diagram . . . . .	30
3.3.2 Interfaces . . . . .	32
3.3.3 Sequence Diagrams . . . . .	32
3.3.4 Activity Diagrammen . . . . .	35
3.3.5 Ontwerpkeuzes gemaakt voor het Gazebo component . . . . .	37
3.4 Design Sub-System Ros . . . . .	38
3.4.1 Component Diagram . . . . .	38
3.4.2 Sequence Diagrams . . . . .	45
3.4.3 Activity Diagrammen . . . . .	49
3.4.4 Ontwerpkeuzes gemaakt voor het Ros component . . . . .	51
3.5 Design Sub-System NRF24 . . . . .	52
3.5.1 Component Diagram . . . . .	52
3.5.2 Aansluiting NRF24 . . . . .	52
3.5.3 Sequence Diagrams . . . . .	53
3.5.4 Activity Diagrams . . . . .	56
3.5.5 Ontwerpkeuzes gemaakt voor het NRF24 component . . . . .	57
3.6 REST gateway . . . . .	58
3.6.1 Component diagram . . . . .	58
3.6.2 Interfaces . . . . .	58
<b>Literatuur</b>	<b>60</b>

# 1 — Inleiding

Het volgende verslag betreft de Software Requirements Specification voor de afstudeerstage van Maurice Berentsen (hierna: student). Dit document volgt het document: “*Software Design Description Template*” (Van Heesch, 2016)

Het beschrijft hoe de uiteindelijke applicatie eruit zal zien en wat de functionaliteit hiervan zal zijn. Op deze manier is het voor de student, maar ook voor de betrokken partijen, duidelijk wat er gerealiseerd zal worden. Dit document verduidelijkt de werking van de componenten en de onderlinge relaties.

## 1.1 Leeswijzer

Eerst zal een korte beschrijving gegeven worden van het doel van dit document. Daarna wordt er met een tabel een begrippenlijst toegelicht. In het tweede hoofdstuk is als eerste een package diagram te vinden waarin de functionaliteit van de verschillende componenten te vinden is en de manier waarop deze componenten met elkaar communiceren. Vervolgens is de algemene flow van het programma te zien, hierin wordt duidelijk welke stappen er onder water worden genomen als er een bepaalde actie wordt uitgevoerd. In hoofdstuk drie worden alle subsystemen duidelijk uitgewerkt en worden de ontwerpen van deze systemen uitgewerkt.

## Begrippenlijst

Term	Beschrijving
Acknowledgement	Term die gebruikt wordt voor het bevestigen van het ontvangst van een bericht.
Adapter	Een ontwerppatroon die gebruikt wordt om niet passende interfaces op elkaar aan te sluiten.
Broadcast	Het rond sturen van een bericht die iedereen mag ontvangen.
Byte	Een samenstelling van 8 bits.
Condensator	Een elektrische component om spanningsschommelingen af te vlakken.
Debug	Term die slaat op debugger. Een debugger bevat informatie die gebruikt wordt om gedrag te bestuderen binnen een applicatie.
Gateway	Een toegangspunt tot een netwerk.
Gazebo	Simulatiesoftware met ondersteuning voor physics.
GPS	Systeem voor locatie en tijd bepaling.
High level interface	Interface die functionaliteit aanbiedt van een low level interface op een versimpelde en geëncapsuleerde wijze.
Low level interface	Interface die dient als een zo direct mogelijk aanspreekpunt op de hardware.
nRF24l01+	Radio transceiver module werkzaam op de 2,4Ghz band.
Physics engine	Software die natuurwetten toepast op virtuele objecten.
Raspberry Pi model 2B+	Micro computer geschikt voor prototyping.
Ros	Robot operating system. Wordt gebruikt voor de transportlaag naar zowel virtuele als gesimuleerde robots.
Roservice	Een request response mechanisme gebruikt door Ros nodes.
Rostopic	Publicatie voor het gebruik van een subscriber patroon binnen Ros.
Router	Netwerkcomponent die berichten kan doorsturen.
Routingstechniek	Techniek die gebruikt voor het opbouwen van een pad binnen een netwerk.
RQT	Programma aangeboden door Ros voor aansluiting op Ros nodes.
SDF	XML format gebruikt voor het inladen van objecten en werelden in Gazebo
Seriële verbinding	Een communicatieverbinding het meest bekend van USB.
Simulatie software	Software die fysieke objecten nabootst in software.
Unittest	Geautomatiseerde test die functie test op resultaat.

Tabel 1.1: Begrippenlijst.

## 2 — Architectural Overview

In het package diagram [Figuur 2.1](#) op de volgende pagina is te zien dat bepaalde onderdelen voorzien zijn van een andere kleur. De groene kleur betekent dat het voorzien is door een externe library. De rode en blauw kleur zijn gegeneerd door Ros, waarbij de rode een rosservice zijn en de blauwe een rostopic. Op het hoogste niveau is het diagram te verdelen in zeven pakketten. De worden eerst beschreven. Na het package diagram zal elke interface kort toegelicht worden

### 2.1 Hoofdgroepen

**Communication** is het pakket die de beslissingen neemt voor de communicatie. Het heeft de intelligentie om het netwerk in kaart te brengen en te routeren. Daarnaast is het in staat om een beslissing te nemen om een verzoek te sturen om de drone te verplaatsen. Tenslotte bevat het meerdere berichten die gebruikt om het netwerk te kunnen onderhouden en op te bouwen.

**Raspberry** is het pakket die de software voor een Raspberry Pi onderscheid in twee rollen. Een Raspberry Pi kan de rol van gateway hebben of de rol van router in het meshnetwerk.

**Gazebo** is de plek waar de virtualisatie van de drone plaats vindt. Er kunnen virtuele router of gateway drones gemaakt worden die voorzien zijn van een virtuele Raspberry Pi ingeladen met de juiste software en een virtuele drone motor. Deze worden als model plugin gekoppeld aan de drone in een gesimuleerde wereld waar Gazebo vervolgens physics op de drones kan uitvoeren.

**Vector** is een package waar alleen een Vector3 template class in zit. Deze wordt gebruikt om te rekenen met punten in de wereld.

**Ros** is de groep waar de simulatie van de communicatie plaats vindt. Er is een virtuele NRF24 om te communiceren met andere nodes in het netwerk. De aanwezige WirelessSignalSimulator zorgt dat dit realistisch gebeurt door alleen communicatie uit te voeren als dit volgens de voorwaarden mag. Dit bepaald de simulator met informatie die het krijgt van Gazebo en de NRF. Tenslotte is er een DroneManager aanwezig die een interface aanbiedt via Ros voor de ontwikkelaar om verbinding te maken met de gateways uit het netwerk.

**REST gateway** is het deel die verantwoording neemt voor het kunnen aanbieden van een REST service naar de gateway toe.

**NRF24** is het gebruikte component om draadloze communicatie mogelijk te maken. Het is voorzien van een driver in de vorm van een low level interface en biedt zich aan via een high level interface voor communicatie.



## 2.2 Interfaces

### 2.2.1 IInternetConnection

Deze high level interface wordt gebruikt voor het leggen van een verbinding naar een extern punt buiten het netwerk. Omdat de gateway zich alleen hoeft te verbinden met een punt bestaat deze interface alleen uit een connect en een disconnect.

### 2.2.2 IGatewayCommands

Deze interface wordt gebruikt voor het ontvangen van commando's via een verbinding met een extern punt buiten het netwerk. Op dit moment is er alleen een functie aanwezig voor het verzoeken van een drone verplaatsing

### 2.2.3 IWirelessCommunication

In deze interface wordt de functionaliteit gesteld waar een draadloos communicatiemiddel aan moet voldoen. Het bevat functies om een antenne te starten en te stoppen. Het versturen van een bericht naar een specifiek punt of het zenden naar elke punt in de buurt. Er is een functie aanwezig om debugging te gebruiken. Tenslotte moet een aansluitend component met deze interface moeten kunnen teruggeven of deze aan of uit staat.

### 2.2.4 IMeshNetwork

Deze interface wordt aangeboden om een aangesloten draadloos communicatie middel de mogelijkheid te geven berichten door te kunnen geven aan het [MeshnetworkComponent](#) en het ID van dit component kenbaar te maken.

### 2.2.5 IMeshDebugInfo

Deze interface wordt aangeboden om een verzameling variabelen openbaar te maken die nuttig zijn als debug informatie.

### 2.2.6 NRF24HighLevelInterface

Deze high level interface is de aangeboden interface van de NRF24l01+ en voldoet aan de interface [IWirelessCommunication](#). Het praat via de low level interface met de RF24 library.

### 2.2.7 NRF24LowLevelInterface

De low level interface is het directe aansluitpunt met de RF24 library



### 2.2.8 IDroneEngine

Een drone engine interface representeert de aansluiting met een drone. Hierin moet het mogelijk zijn om een doel coördinaat te sturen naar de drone om zich naartoe te verplaatsen. Daarnaast moet het mogelijk zijn voor een drone om de huidige positie terug te geven.

### 2.2.9 IRoutingTechnique

Deze interface bevat functies voor het uitvoeren van een routingstechniek. Het heeft functies voor het starten en onderhouden van het netwerk. Daarnaast zijn er functies hoe gereageerd moet worden op het vinden en verliezen van andere netwerkpunten. Er zijn functies aanwezig om een adres op te halen waar naartoe gezonden moet worden om een punt te bereiken. Ook kan er opgehaald worden welke punten zijn aangesloten, hoeveel punten dit zijn en hoeveel directe aansluitingen er zijn. Tenslotte is er functie aanwezig die aangeroepen wordt zodra een drone zich verplaatst zodat hier juist op gereageerd kan worden.

## 2.3 Ros topics en services

Zoals al eerder benoemd wordt er gebruik gemaakt van de transportlaag van Ros. Dit wordt gedaan in de vorm van Ros topics en services. Hieronder worden de messages en services kort toegelicht.

### 2.3.1 RequestGatewayDroneFlight

Via deze aangeboden service is het mogelijk om een verzoek te sturen naar de gateway voor een verplaatsing. In het verzoek moet het nodeID en locatie zitten.

### 2.3.2 CasusRequest

Deze service maakt het mogelijk om een casus posities door te sturen naar de drones.

### 2.3.3 PowerSwitch

Via deze service kan een [VirtualNRF24](#) aan of uit gezet worden.

### 2.3.4 WirelessMessageRequest

De [VirtualNRF24](#) gebruikt deze service om een NRF24 bericht te versturen naar een andere NRF24. De response geeft aan of het zenden lukte.

### 2.3.5 AreaScanRequest

Deze service kan aangeroepen worden om de id's van alle nodes binnen het bereik van een node terug te krijgen. Dit is nodig om een algemene zending naar iedereen binnen bereik mogelijk te maken.

### 2.3.6 RequestGPS

Deze service wordt aangeboden door de [DroneEngine](#) en biedt de mogelijkheid om de huidige locatie van een drone op te vragen.

### 2.3.7 RequestGatewayDroneFlight

Dit topic wordt gebruikt om locatieverplaatsingverzoeken op te publiceren.

### 2.3.8 NodeDebugInfo

Elke [VirtualNRF24](#) maakt wanneer dit verzocht wordt een debug topic aan. Hierop wordt informatie gepubliceerd wat van toegevoegde waarde is voor ontwikkelaars.

### 2.3.9 NRF24

Een [VirtualNRF24](#) luistert naar dit topic om zo berichten te kunnen ontvangen.

### 2.3.10 Location

Het location topic representeert de verbinding die een [MeshnetworkComponent](#) zou hebben met een drone. Dit topic wordt gebruikt om doelen op te versturen waar de drone naartoe moet.

### 2.3.11 DroneInfo

Dit topic is onmisbaar voor de [WirelessSignalSimulator](#). De informatie die hierop gepubliceerd wordt bevat informatie over de locatie van een [VirtualNRF24](#) of die aan staat en naar welk [NRF24](#) topic deze luistert.

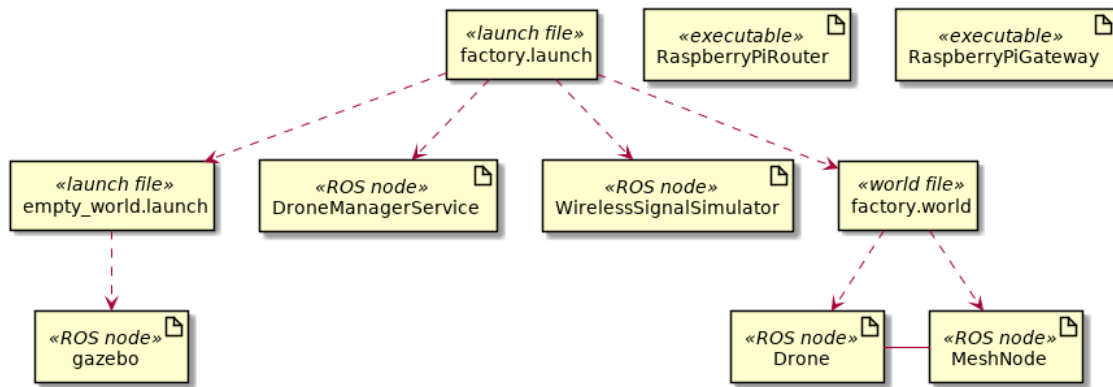
### 2.3.12 RestCommands

Deze interface wordt via een Rest service aangeboden. Hiermee wordt het mogelijk om vanuit een extern punt commando's naar de gateway te versturen.

## 3 — Detailed Design Description

### 3.1 Deployment Diagram

In dit deployment diagram staan de executables (Ros nodes) die opgestart worden door de verschillende launch files, daarnaast zijn ook de twee executables meegenomen die gebruikt worden op de Raspberry Pi's



Figuur 3.1: Deployment diagram drone meshnetwerk.

#### 3.1.1 Ontwerpkeuzes met betrekking tot deployment

##### Geen aangeboden mogelijkheid om drones los op te starten

Er is voor gekozen om alleen via het factory.launch bestand de drones te kunnen starten. De factory.launch moet gebruikt worden, omdat deze factory.world aanroept welke op zijn beurt de plugin start die de drones aanmaakt. In factory.world is het configureerbaar hoeveel gateway en router drones aangemaakt moeten worden. Deze keuze is gemaakt, omdat elke drone een uniek id moet hebben in de simulatie. Dit moet omdat elke motor zijn aansluiting publiceert met dit id.

Om de gebruiker ervan te ontmoedigen is er daarom ook geen SDF bestand aanwezig waarmee normaal objecten in Gazebo geladen worden.

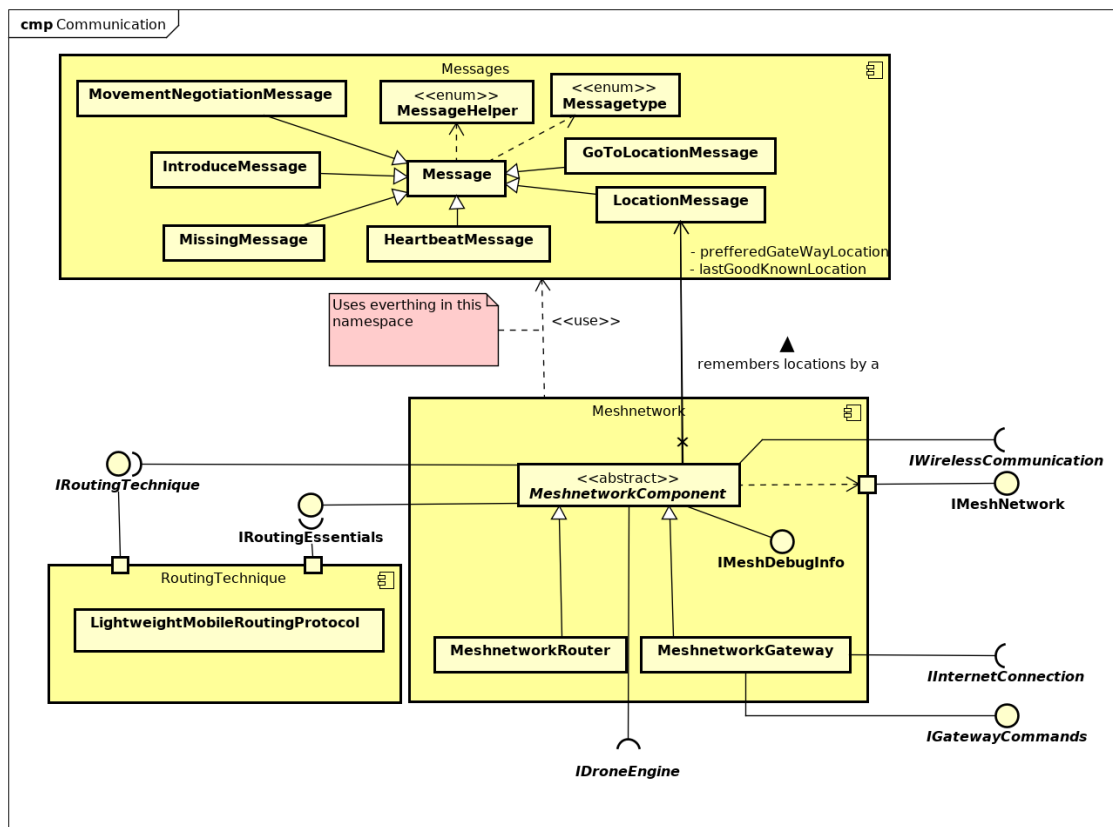
### 3.2 Design Sub-Systeem Communicatie

Het sub-systeem communicatie is verantwoordelijk voor de communicatie door het gebruik van mesh technologie. Het subsysteem kan dit niet alleen en werkt daarom samen met

andere subsystemen, welke zichtbaar zijn in het ???. Het component communicatie bestaat ook weer uit kleinere componenten welke weergegeven staan in het [Component diagram communicatie](#).

### 3.2.1 Component Diagram

In de onderstaande afbeelding 3.2 staat het component diagram van de communicatie weergegeven. Het is een extractie uit het ??



Figuur 3.2: Component diagram communicatie.

#### Meshnetwork

Dit component bevat de basis componenten voor het opbouwen van een meshnetwork.

#### MeshnetworkComponent

Een MeshnetworkComponent is de basis van elk component in het meshnetwork. Het vereist routingstechniek via de [IRoutingTechnique](#), een aansluiting naar een drone via [IDroneEngine](#) en een vorm van draadloze communicatie via [IWirelessCommunication](#). Hij maakt gebruik van het component [Messages](#) om te communiceren met andere MeshnetworkComponenten.

#### **MeshnetworkRouter**

Een meshnetwork router is een [MeshnetworkComponent](#) die in staat is verbinding met andere nodes op te bouwen. Zijn doel is om altijd verbinding te hebben met een [MeshnetworkGateway](#). Als hij dit te lang niet heeft kan hij zich verplaatsen door aanspraak te maken op de DroneEngine. Wanneer hij zich in een groep van andere routers bevindt zal hij eerst onderling onderhandelen wie er zich moet verplaatsen.

#### **MeshnetworkGateway**

De gateway is een [MeshnetworkComponent](#) die in staat is een verbinding op te bouwen naar een punt buiten het meshnetwork. In de huidige situatie kan dit via een internetverbinding die loopt via de interface [InternetConnection](#).

#### **Internet**

Dit component bevat een high level interface om verbinding te maken en te verbreken met het internet. Daarnaast biedt het een interface aan om berichten te ontvangen.

#### **Messages**

Dit component betreft een verzameling van bericht samenstellingen die gebruikt worden voor de communicatie van het meshnetwork.

#### **Message**

Een Message is de basis van elke bericht en bevat tenminste de volgende informatie: Maker, zender, berichttype, ontvanger, geadresseerde.

#### **GoToLocationMessage**

Dit bericht bevat een locatie waar een drone zich naartoe moet verplaatsen.

#### **HeartbeatMessage**

Dit bericht wordt gebruikt om de verbinding met anderen te onderhouden. Dit bericht maakt een hop per keer dat deze doorgestuurd wordt.

#### **IntroduceMessage**

Een introductie bericht wordt gebruikt door een node om zich voor te stellen aan alle andere nodes die dichtbij zijn.

#### **LocationMessage**

Dit bericht wordt gebruikt om de huidige locatie van een node door te sturen naar een andere node.

#### **MissingMessage**

Zodra een [MeshnetworkComponent](#) de verbinding verliest met een node gebruikt hij dit bericht om andere nodes daarover te informeren.

#### **MovementNegotiationMessage**

Om onderling te onderhandelen tussen de nodes wie er actie moet ondernemen wordt dit bericht gebruikt.

#### **RoutingTechnique**

Dit component voorziet het meshnetwerk van een routingstechniek wat dus inhoudt dat dit het component is die de communicatieroutes opbouwt naar andere punten in het netwerk.

#### **LightweightMobileRoutingProtocol**

Deze techniek is een hybride meshnetwerk routeringstechniek waarbij een node de burens die deze heeft ziet als een kind. Als een node een nieuw kind heeft vertelt hij dit aan zijn burens. Omdat een node door zijn burens ook gezien wordt als kind registreren deze zijn nieuwe kind dus als kleinkind. Hierdoor hoeft een node alleen maar te zoeken aan wie welk kind hij een bericht hoeft door te geven. De complexiteit van de routingstechniek neemt per stap in het netwerk af bij nodes met meerdere kinderen. Als een drone zich verplaatst heeft wist deze het de opgebouwde geheugen van kinderen en kleinkinderen.

#### **Wireless**

Dit component voorziet een [MeshnetworkComponent](#) van een high level interface voor draadloze communicatie. Als er een draadloos component wordt aangesloten moet deze hierop aangesloten worden.

### **3.2.2 Interfaces**

Het communicatie component biedt zowel intern als extern interfaces aan. Deze worden hieronder omschreven.

#### **Extern aangeboden interface beschrijvingen**

Het communicatie component heeft drie extern aangeboden interfaces. De interface `IMeshNetwork` wordt aangeboden om berichten voor het meshnetwerk te kunnen ontvangen vanaf een aangesloten draadloos communicatie component. De interface `IGatewayCommands` wordt gebruikt voor het ontvangen van aansturing vanuit een extern punt naar een gateway. Tenslotte biedt het `MeshnetworkComponent` nog een interface aan met debug-informatie genaamd `IMeshDebuginfo`. Omdat deze laatste interface alleen maar getters bevat is besloten deze niet uitgebreid te behandelen.

**IMeshNetwork** De volgende functies worden aangeboden door de interface IMeshNetwork.

```
void OnMsg( const uint8_t* message );

const uint8_t getNodeID( ) const;
```

**OnMsg** Deze functie geeft een antenne toegang tot een meshcomponent om berichten door te geven. Het bericht wordt doorgegeven met een adresverwijzing naar de array waar het bericht in staat

*preconditie:* Het meshcomponent heeft een thread gestart voor bericht afhandeling.

*postconditie:* Het opgegeven bericht is behandeld door het meshcomponent.

**getNodeID** Deze functie geeft de antenne door welk node id het meshcomponent bezit.

*preconditie:* Het meshcomponent heeft een node id

*postconditie:* Het id van het meshcomponent is doorgegeven.

**IGatewayCommands** De volgende functie wordt aangeboden door de interface IGatewayCommands.

```
void GoalRequestToDrone( const uint8_t ID, const float latitude ,
                        const float longitude, const uint16_t height );
```

**SendGoalRequestToDrone** Deze functie laat de aangesloten gateway een verzoek doen tot het verplaatsen van een drone. Er is nog geen ingebouwde functionaliteit om terug te geven of dit verzoek ook is aangekomen bij de drone.

*preconditie:* De gateway die het verzoek ontvangt heeft een route tot de drone die het verzoek moet ontvangen.

*postconditie:* De drone heeft het verzoek ontvangen en zal zich daar naartoe gaan verplaatsen

### Intern aangeboden interface beschrijvingen

Intern worden er twee interfaces aangeboden. Het MeshnetworkComponent biedt een interface IRoutingEssentials aan waar basale communicatie functies in zitten om routingstechnieken mogelijk te maken. De andere interface IRoutingTechnique wordt door het subcomponent RoutingTechnique aangeboden.

**IRoutingEssentials** De volgende functies worden aangeboden door de interface IRoutingEssentials.

```
void searchOtherNodesInRange( )

bool sendHeartbeat(uint8_t other);
```

**searchOtherNodesInRange** Deze functie wordt aangeroepen om het component te verzoeken om te kijken of er andere nodes binnen bereik zijn.

**preconditie:** Meshcomponent is voorzien van een draadloos communicatiemiddel die gestart is.

**postconditie:** Meshcomponent heeft een inventarisatie gemaakt van alle nodes binnen bereik.

**sendHeartbeat** Door het aanroepen van deze functie wordt er een heartbeat verzonden van het component naar een node met het opgegeven ID in de parameter.

**preconditie:** Geadresseerde is bekend bij de routingstechniek.

**postconditie:** Routingstechniek geeft terug welk adres de volgende stap is voor het bericht.

**IRoutingTechnique** De volgende functies worden aangeboden door de interface IRoutingTechnique.

```
uint8_t getDirectionToNode( const uint8_t node );

void startRouting( );

void maintainRouting( );

void NodeMovedLocation( );

uint8_t cantCommunicateWithNode(const uint8_t node);

uint8_t OtherCantCommunicateWithNode(const uint8_t other, const uint8_t node);

void canCommunicateWithNode(const uint8_t node);

void OtherCanCommunicateWithNode(const uint8_t other, const uint8_t node);

const uint16_t getTableSize( );

const uint16_t getAmountOfChildren( );

const std::set< uint8_t > getSetOfChildren( );

const bool empty( );
```

**getDirectionToNode** Dit is de functie waar de aanvragende interface het meeste belang bij heeft. Deze functie verwacht een id van de geadresseerde node als parameter. De routingstechniek zal vervolgens uitzoeken naar welke node een bericht doorgegeven moet worden om het bericht aan te laten komen bij de geadresseerde. Dit adres wordt terug gegeven als een return waarde.

**preconditie:** Geadresseerde is bekend bij de routingstechniek.

**postconditie:** Routingstechniek geeft terug welk adres het vervolg adres is voor het bericht.



**startRouting** Deze functie wordt aangeroepen zodra de routingstechniek moet beginnen met het opbouwen van communicatie routes.

**preconditie:** De communicatie waarover de routing gebeurt is beschikbaar

**postconditie:** De node zich aangemeld bij andere nodes en kan adressen opslaan.

**maintainRouting** Deze functie wordt herhaaldelijk aangeroepen in de software. Het roept de routingstechniek aan om een onderhoud te plegen aan de opgebouwde routing informatie. In dit geval controleert de routingstechniek of de aansluitende nodes nog bestaan.

**preconditie:** De routingstechniek is gestart met routeren en heeft een lijst van beschikbare nodes.

**postconditie:** Er is onderhoud uitgevoerd aan de lijst van beschikbare nodes.

**NodeMovedLocation** Deze functie wordt aangeroepen zodra de drone zich verplaatst heeft. Het is aan de routingstechniek om hier juist op te reageren. In de huidige implementatie worden alle routes als ongeldig gezien waardoor de tabel geleegd wordt.

**preconditie:** Er is geen preconditie.

**postconditie:** De routingstechniek heeft de tabel bij juist bijgewerkt.

**cantCommunicateWithNode** Deze functie wordt aangeroepen als er geen connectie gelegd kan worden met een node. Deze functie geeft 0 terug als er een alternatieve route mogelijk is naar de node.

**preconditie:** De node is aanwezig in de route tabel.

**postconditie:** De route wordt verwijderd of als niet beschikbaar beschouwd.

**OtherCantCommunicateWithNode** Deze functie wordt aangeroepen als een andere node aangeeft dat hij geen connectie kan maken met een node. Deze functie geeft terug op hoeveel routes de niet bereikbare node voorkomt.

**preconditie:** De node is aanwezig in de route tabel.

**postconditie:** De route wordt verwijderd of als niet beschikbaar beschouwd.

**canCommunicateWithNode** Zodra een node bevestiging heeft dat deze verbinding kan maken wordt deze functie aangeroepen. Als parameter wordt het adres meegegeven.

**preconditie:** Er is bevestiging dat een node een directe verbinding heeft met een andere node.

**postconditie:** De verbonden node wordt toegevoegd als direct route punt.

**OtherCanCommunicateWithNode** De functie wordt aangeroepen als een andere node doorstuurt dat hij bevestiging heeft dat hij verbinding heeft met een nieuwe node in het netwerk. Als parameter wordt het adres meegegeven van zowel de node die aangeeft dat hij een verbinding heeft gevonden als het adres die hij gevonden heeft.

**preconditie:** Er is bevestiging dat een node een verbinding heeft met een andere node.

**postconditie:** De gevonden node wordt toegevoegd als route punt in het netwerk.

**getTableSize** Deze functie geeft terug hoeveel routes er zich bevinden in de opgebouwde routing tabel.

**preconditie:** Geen preconditie.

**postconditie:** Alle routes zijn geteld en de functie geeft dit aantal terug.

**getAmountOfChildren** In deze functie wordt het aantal gevonden direct aansluitende nodes geteld en terug gegeven.

**preconditie:** Geen preconditie.

**postconditie:** Het aantal direct aansluitende nodes is geteld en dit aantal is terug gegeven.

**getSetOfChildren** Hier wordt een set gemaakt met id's van direct aansluitende nodes.

**preconditie:** Geen preconditie.

**postconditie:** Er is een set met de id's van alle nodes waar een directe verbinding mee gevonden is.

**empty** Deze functie geeft een boolean terug welke aangeeft of de tabel leeg is. Dit is een aparte functie, omdat dit scheelt in de complexiteit ten opzichte van het tellen van een tabel en die vergelijken met 0.

**preconditie:** Geen preconditie

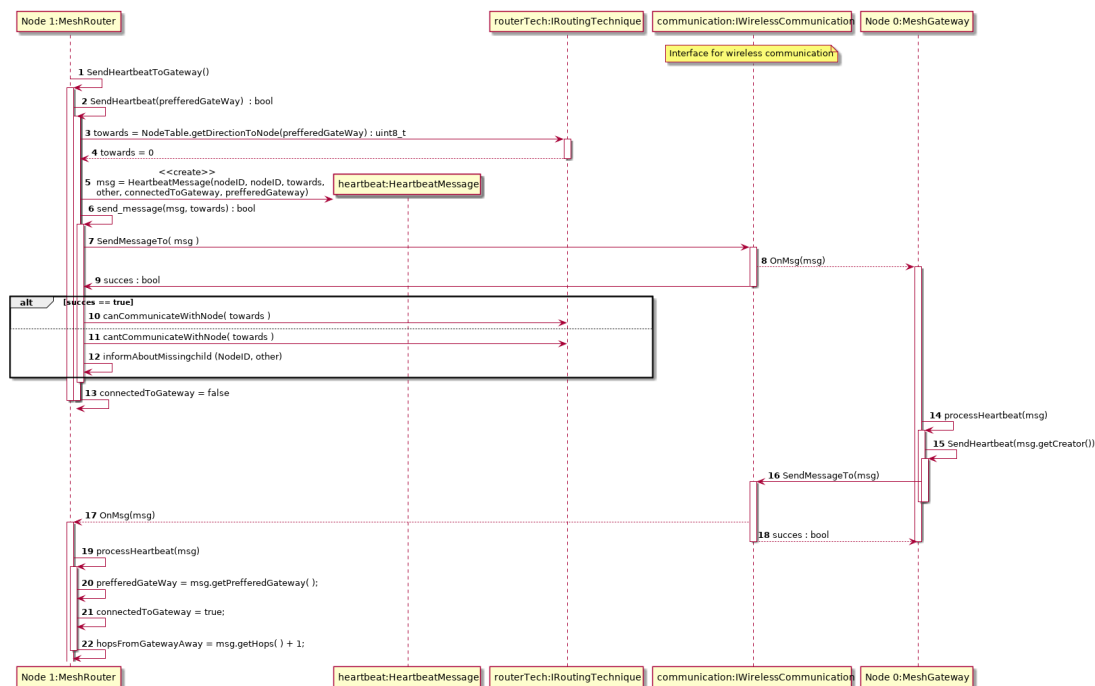
**postconditie:** Er is een controle gedaan of de lijst leeg is en dit wordt terug gegeven.

#### 3.2.3 Sequence Diagrams

Hier wordt met sequence diagrammen communicatie wegen omschreven die gebruikt worden tussen verschillende nodes in het netwerk.

##### Heartbeat van router naar gateway met directe verbinding

Om duidelijk te maken hoe een bericht van een node naar een gateway komt, wordt het scenario van een heartbeat uitgewerkt naar een direct verbonden router. Deze weg van communicatie wordt voor alle directe berichten hetzelfde uitgevoerd. Dat maakt het overbodig om voor elk type bericht een sequence diagram uit te werken



Figuur 3.3: Sequence diagram van een heartbeat bericht van een router naar een gateway met een directe onderlinge verbinding.

In diagram 3.3 is de flow zichtbaar van een router die een heartbeat bericht verstuurd naar een gateway en daar ook weer response op krijgt. Op het moment dat een router het bericht krijgt om een heartbeat te versturen naar de gateway, zal hij dit altijd doen naar de gateway waar hij een voorkeursverbinding (preferredGateway) mee heeft. Hij zal aan de aangesloten routingstechniek vragen of er een route mogelijk is naar de router en aan wie hij zijn bericht dan moet afgeven. In het geval van deze sequence diagram heeft de router een directe verbinding met de gateway. In stap 4 is te zien dat het teruggegeven adres 0 is, het adres van de gateway.

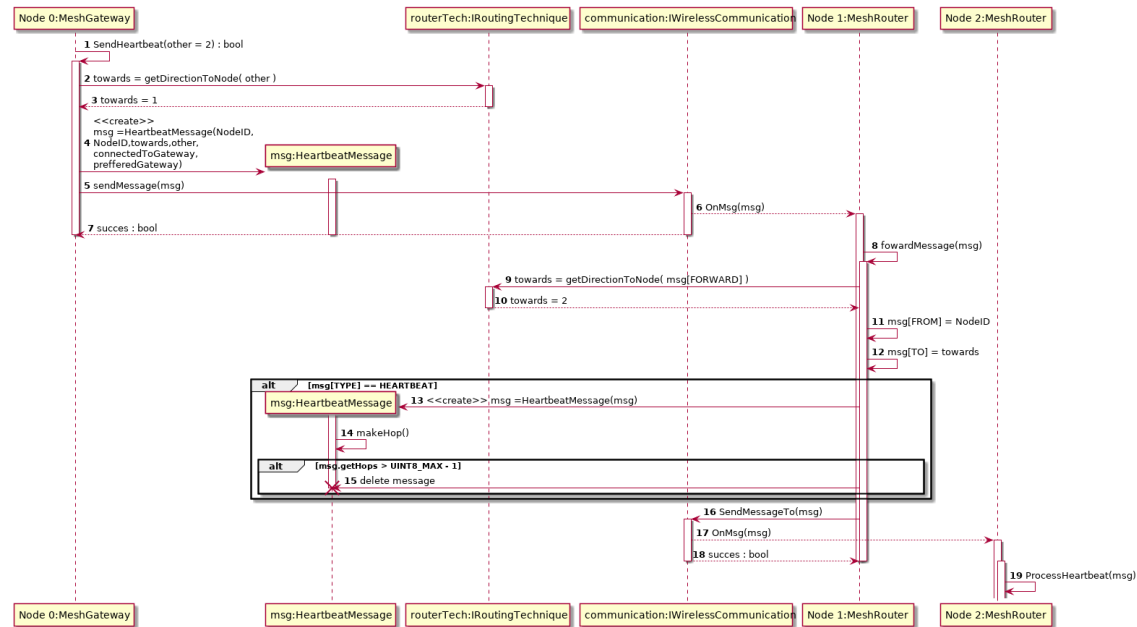
Vervolgens maakt de router een heartbeat bericht aan die het via de [IWirelessCommunication](#) interface verstuurd naar de Gateway. De router geeft door aan de routingstechniek in stap 10 en 11 of het zenden wel of niet gelukt is op basis van de succes feedback uit stap 9. Als het niet lukt geeft hij dit ook door aan zijn aangesloten punten in stap 12. De werking hiervan is terug te vinden in [Informatieverspreiding bij verliezen contact met andere node](#). Als laatste voordat de router zijn functie verlaat zet hij in stap 13 de boolean die bijhoudt of er verbinding is met de gateway (connectedToGateway) op false. Dit gebeurt omdat het component verwacht dat er reactie komt van de gateway die het weer op true zet.

Bij stap 8 is het gelukt om het bericht te versturen naar de gateway die daar via zijn eigen [IWirelessCommunication](#) interface het bericht ontvangt. Er is voor gekozen om deze interface, maar één keer weer te geven in de tabel om het overzichtelijk te houden. De reactie van een gateway op een heartbeat is altijd dat er een heartbeat wordt teruggestuurd naar de zender. De functie SendHeartbeat(uint8\_t) in stap 15 werkt gelijk aan die van stap 2 en is daarom niet opnieuw uitgewerkt.

Stap 17 is het moment dat de router weer response heeft van de gateway. Hij stelt met de inhoud van het bericht het ID van de gateway in als preferredGateway, zet de boolean

connectedToGateway weer op true en registreert hoeveel hops het bericht nodig had om aan te komen.

### Heartbeat van gateway naar een router via een andere router



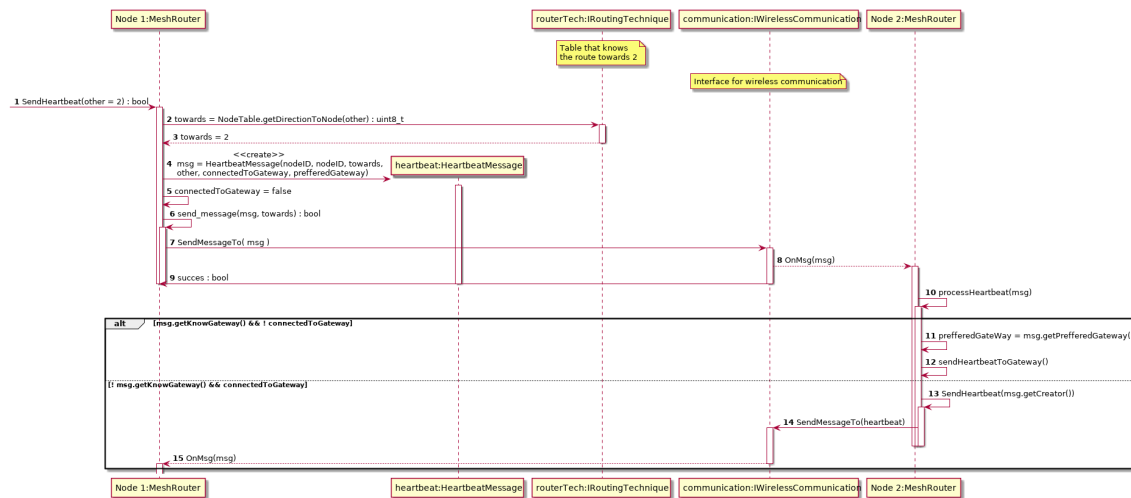
Figuur 3.4: Sequence diagram van een heartbeat bericht van een gateway naar een router via een andere router.

Diagram 3.4 laat zien hoe een heartbeat verstuurd wordt vanaf een gateway naar een router via een andere router. Er is gekozen om de flow van een heartbeat bericht te laten zien, omdat deze bij een forward ook een hop extra krijgt en daarmee het meest complex is. Elke node heeft een eigen instantie van een routingstechniek en draadloze communicatie, in deze diagram is gekozen om dezelfde te gebruiken om de leesbaarheid goed te houden.

Bij stap 3 is zichtbaar dat de routingstechniek bij aanvraag voor een route naar node 2 een adres teruggegeven wordt van node 1, omdat via deze weg het bericht doorgegeven moet worden. Vanaf stap 8 is zichtbaar wat er gebeurt als een bericht doorgestuurd moet worden. Bij stap 9 wordt er eerst gekeken of er een route is naar de geadresseerde en aan wie het bericht dan doorgegeven moet worden. Node 1 heeft een directe verbinding met node 2 dit wordt dan ook als adres doorgegeven in stap 10. Vervolgens past de meshrouter het bericht aan dat zodat de zender en ontvanger weer juist staan. De waarden waar in staat wie de geadresseerde en maker van het bericht zijn blijven onaangepast. Als het bericht van het type heartbeat is wordt deze apart genomen om er in stap 14 een hop bij op te tellen. Als een heartbeat bericht meer hops heeft gemaakt dan dat er in de waarde van hop past, wordt het bericht verwijderd en de flow gestopt.

In stap 16 wordt het bericht doorgestuurd naar de geadresseerde. Als het bericht een heartbeat was wordt deze doorgestuurd anders wordt het aangepaste bericht verstuurd.

### Heartbeat van een router naar een andere router



Figuur 3.5: Sequence diagram van een router die een heartbeat stuurt naar een router.

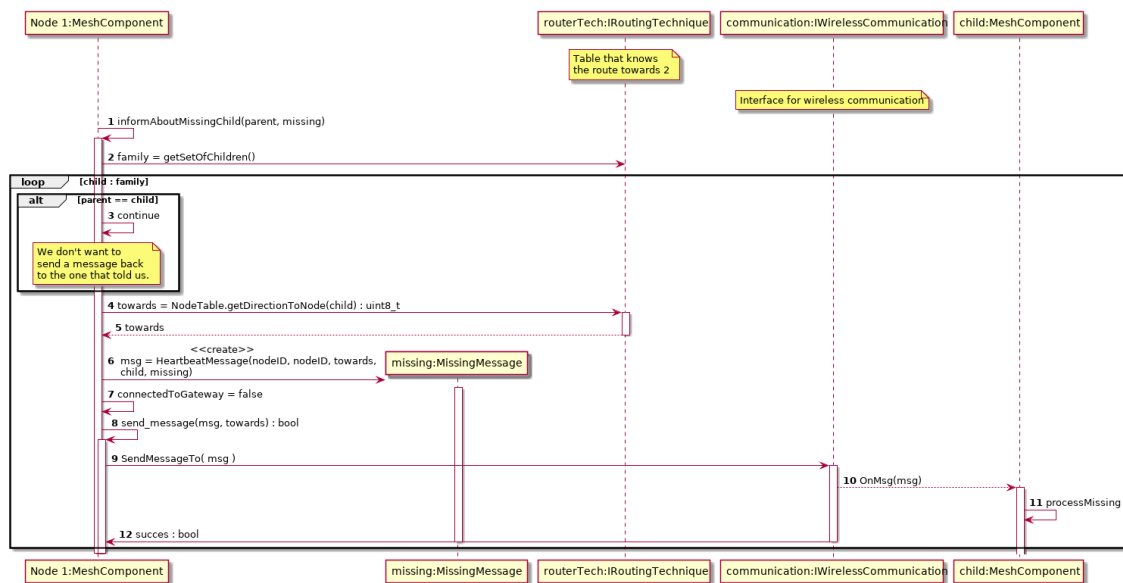
In de bovenstaande sequence diagram 3.5 wordt de flow behandeld van een heartbeat van een router naar een andere router. De keuze om dit in een sequence diagram uit te werken is om te laten zien dat er niet in elke scenario een reactie bericht verwacht hoeft te worden.

In stap 11 en 12 is bijvoorbeeld de afhandeling te zien als de zendende router verbinding heeft met een gateway en de ontvangende router niet. In dit scenario gaat de router geen bericht terugsturen, maar stelt hij de gateway die de zendende router heeft in als voorkeur en probeert een heartbeat te versturen naar deze gateway. Als het scenario omgedraaid is en de zendende router heeft geen verbinding met een gateway, maar de ontvangende router wel stuurt hij juist wel een heartbeat terug. De zendende router zal hierdoor dezelfde stappen gaan uitvoeren als stap 11 en 12 waardoor hij weer een verbinding opbouwt met een gateway.

Dit houdt dus ook in dat als beide routers geen verbinding hebben met een gateway, of allebei wel, ze geen response sturen op een heartbeat.

### Informatieverspreiding bij verliezen contact met andere node

Op het moment dat een node doorheeft dat er een verbinding verloren is met een andere node gaat hij de op hem aangesloten nodes daarover informeren. Dit zodat ieder andere node zijn routingstabel kan updaten



Figuur 3.6: Sequence diagram van een component die zijn aangesloten nodes op de hoogte brengt van een verloren node.

De eerste stap die een meshcomponent onderneemt is het raadplegen van de routerings-techniek of er direct aangesloten nodes zijn. Per aangesloten node verstuurt het component een bericht dat er een andere node geen verbinding meer wil maken. Een uitzondering is als de direct aangesloten node degene is die het component geïnformeerd heeft over het feit van de vermissing. Die ontvangt geen bericht, omdat deze al op de hoogte is.

### 3.2.4 Activity Diagrammen

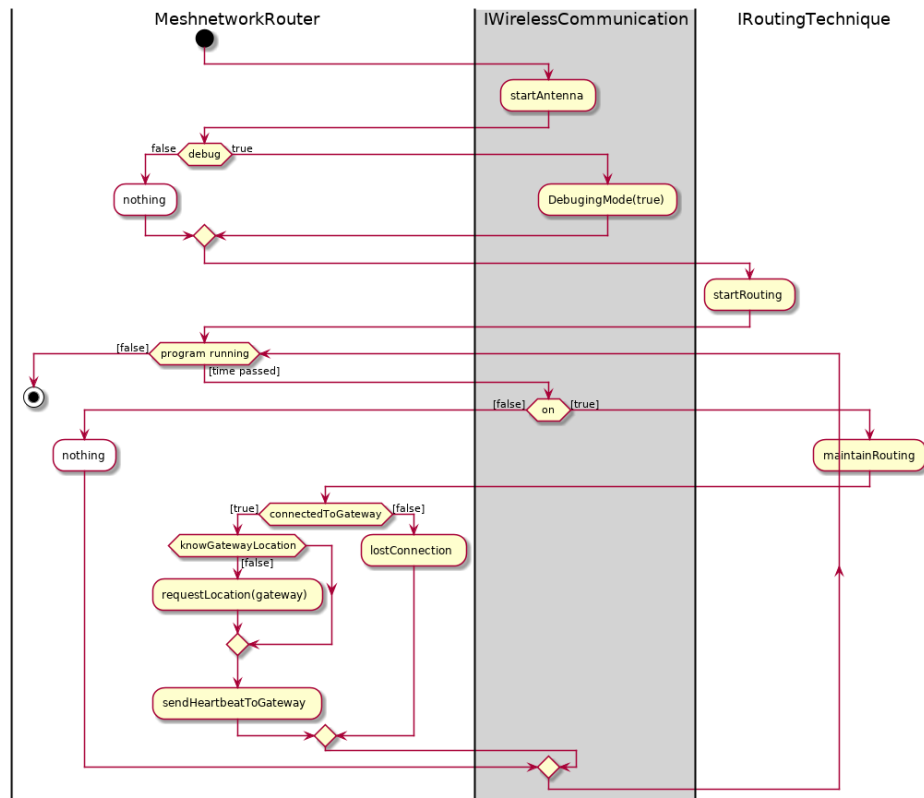
Met de onderstaande activity diagrammen wordt de flow van de communicatie applicatie uitgelegd. In het component zijn twee soorten applicaties beschikbaar een router en een gateway. Deze worden eerst apart behandeld daarna zal de rest van het component aan bod komen.

Voor elk activity diagram telt dat er onder het diagram een begeleidende tekst is toegevoegd.

#### Activity diagrammen router

Eerst wordt de algemene flow van de router applicatie toegelicht vervolgens zullen protocollen worden toegelicht die optreden in bepaalde situaties.

**Meshnetwerk router applicatie** De onderstaande flow zichtbaar in [Figuur 3.7](#) laat de algemene flow zien van de router applicatie.



Figuur 3.7: Activity diagram meshnetwerk router applicatie.

Zodra het programma start wordt als eerste de antenne aan gezet. Wanneer aangegeven wordt ook de debug stand van de antenne geactiveerd. Nadat deze is opgestart wordt kenbaar gemaakt aan de routeringstechniek dat deze kan starten met het opbouwen van routes.

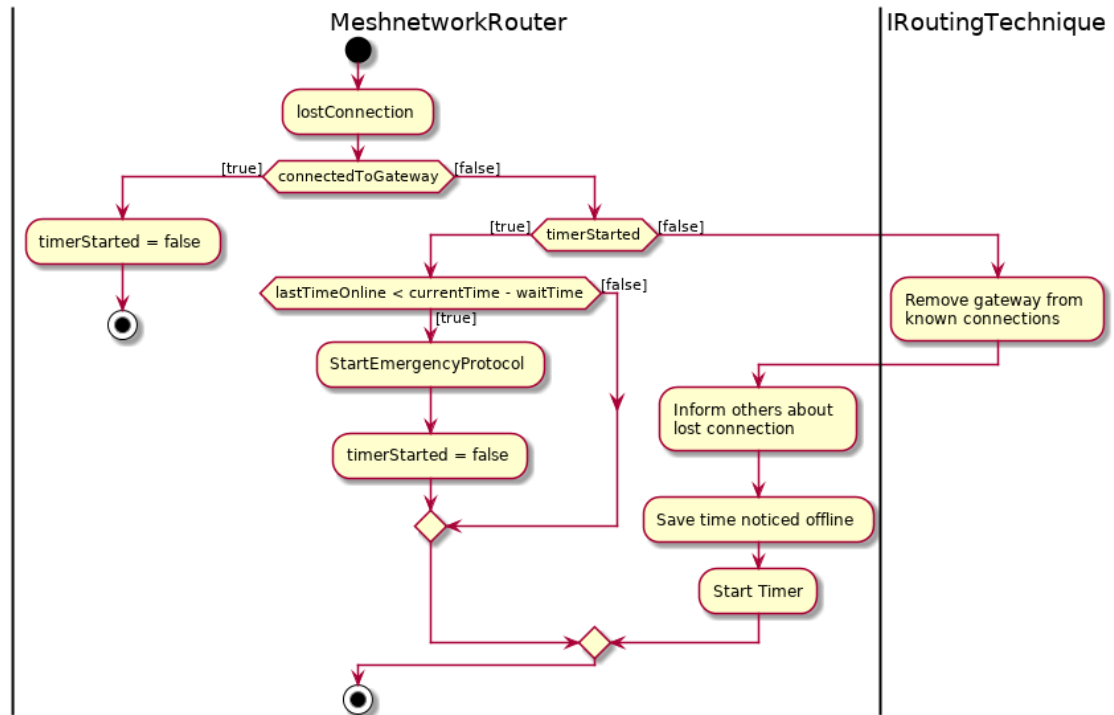
Nu deze twee zijn opgestart kan de basis lus van het programma beginnen. Zolang het programma draait wordt deze lus herhaald.

Elke keer zodra het programma op dit punt aankomt wacht het 10 (instelbaar) seconden. Hierna controleert de router of de aangesloten antenne nog aan staat. Als dit niet zo is begint de lus opnieuw. Als de antenne wel aan staat wordt er een verzoek gedaan aan de routeringstechniek om onderhoud te plegen.

Vervolgens zijn er twee paden mogelijk waarbij er gekeken wordt of er verbinding is met een gateway. Als er geen verbinding is wordt het protocol opgestart voor een verloren verbinding. Hoe dit protocol werkt, wordt apart toegelicht in het hierop volgende diagram.

Wanneer er wel verbinding is met een gateway wordt er gekeken of de router ook op de hoogte is van de locatie van de router. Als hij dit nog niet weet vraagt hij deze op bij de gateway. Tenslotte wordt er een heartbeat verstuurd naar de gateway.

**Meshnetwork router protocol bij verloren verbinding** In het diagram getoond in [Figuur 3.8](#) wordt het protocol uitgelegd die een router uitvoert zodra deze zijn verbinding heeft verloren met een gateway.



Figuur 3.8: Activity diagram verloren verbinding protocol.

Het protocol start op met het nogmaals controleren of de gateway geen verbinding heeft. Als er nog steeds geen verbinding is wordt er gekeken of er al een timer gestart is die bijhoudt hoelang er al geen verbinding meer is.

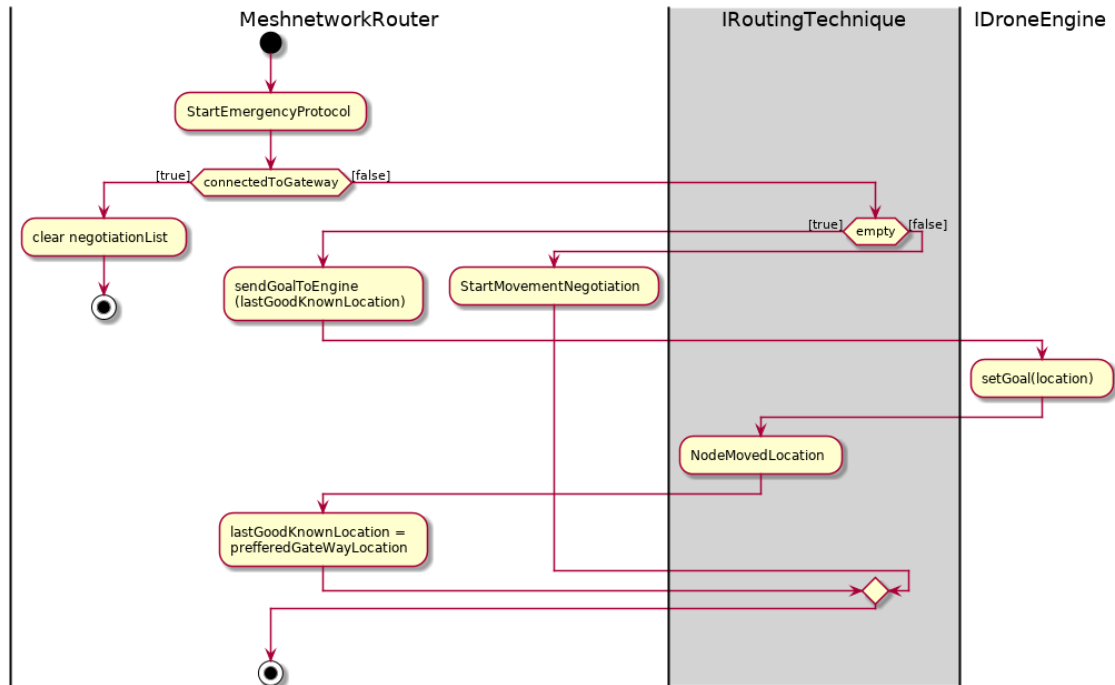
Als er nog geen timer gestart is wordt de routingstechniek op de hoogte gebracht dat de verbinding verloren is met de gateway. Vervolgens worden de direct aangesloten nodes ook op de hoogte gebracht van de verloren verbinding. Tenslotte wordt er een timer gestart door een boolean om te schakelen. Hierna wordt de huidige tijd opgeslagen waarna de functie wordt verlaten.

Als er al wel een timer gestart is wordt er gekeken of de tijd die een node zonder verbinding mag zitten al verstreken is. Als deze tijd nog niet verstreken is wordt de functie verlaten.

Wanneer de tijd wel verstreken is wordt het noodprotocol opgestart die in de volgende paragraaf beter wordt toegelicht. Zodra dit protocol is uitgevoerd wordt de timer weer uitgezet zodat deze in een volgende lus weer opnieuw gaat lopen.



**Meshnetwork router noodprotocol** In [Figuur 3.9](#) wordt het uitgevoerde protocol toegelicht die wordt uitgevoerd wanneer een node te lang zonder verbinding zit.



Figuur 3.9: Activity diagram noodprotocol.

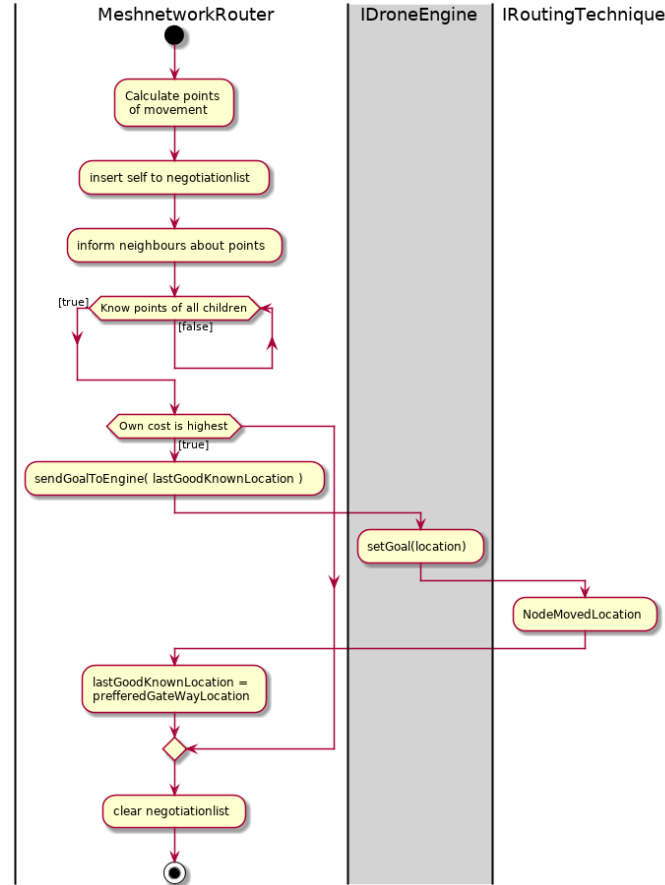
Het noodprotocol begint met een laatste controle of de node ondertussen alweer verbinding heeft, als dit zo is wordt het protocol niet uitgevoerd en de onderhandelingslijst leeg gehaald.

Wanneer er nog steeds geen verbinding is gelegd met een gateway wordt het protocol gestart. Eerst wordt de routingstechniek geraadpleegd of er nog andere nodes verbonden zijn met deze node.

Als dat niet zo is wordt er een verzoek gestuurd naar de drone om zich te gaan verplaatsen naar de locatie die als laatste gemarkeerde is als goede locatie. Nadat deze verplaatst is wordt de locatie van de gateway ingesteld als laatst bekende goede locatie. Hierna wordt de routingstechniek op de hoogte gesteld dat er een verplaatsing heeft plaats gevonden. Als dit allemaal is uitgevoerd wordt dit protocol gesloten.

Wanneer er wel andere nodes verbonden zijn met deze node dan moeten deze onderling gaan onderhandelen wie er een verplaatsing moet gaan uitvoeren om zo efficiënt mogelijk het netwerk proberen te herstellen. Deze werking is apart uitgelegd in de hier opvolgende activity diagram [3.10](#). Nadat deze onderhandelingen zijn afgelopen wordt het noodprotocol afgesloten zodat het netwerk een kans heeft om zichzelf te herstellen.

**Meshnetwork router verplaatsingsonderhandeling** Figuur 3.10 laat de flow zien die een enkele node neemt wanneer deze start met onderhandelen over wie er een verplaatsing moet uitvoeren.



Figuur 3.10: Activity diagram verplaatsingsonderhandeling.

De flow begint met het berekenen hoeveel punten de beweging van de node scoort. In de opgeleverde applicatie worden de verplaatsingspunten op twee factoren gebaseerd. Het aantal punten die een node krijgt staat gelijk aan de afstand die het heeft hemelsbreed met de gateway.

Hiervoor wordt Pythagoras gebruikt om de afstand te berekenen wat zichtbaar is in [Vergelijking 3.1](#)

$$afstand = \sqrt{(x_{gateway} - x_{postitie})^2 + (y_{gateway} - y_{postitie})^2 + (z_{gateway} - z_{postitie})^2} \quad (3.1)$$

In deze vergelijking is de positie van een drone vertaald naar een cartesische waarde daarom worden er x,y,z waardes gebruikt die slaan op de latitude, longitude en height waardes van de drone.

Een voorwaarde is wel dat de locatie waar de node zich naartoe wil gaan verplaatsen niet bezet wordt door een andere node. Als dit zo is worden de punten op -1 gezet, waardoor die automatisch niet meer mee doen, omdat het minimale aantal punten 0 is.

Hierna voegt de node zichzelf toe in de onderhandellijst en maakt hij zijn score bekend aan zijn aangesloten nodes. Zodra de node dit bekend heeft gemaakt wacht hij tot de andere nodes dit ook hebben gedaan.

Als dit zover is kijkt de node of hij de hoogste score heeft in een lijst.

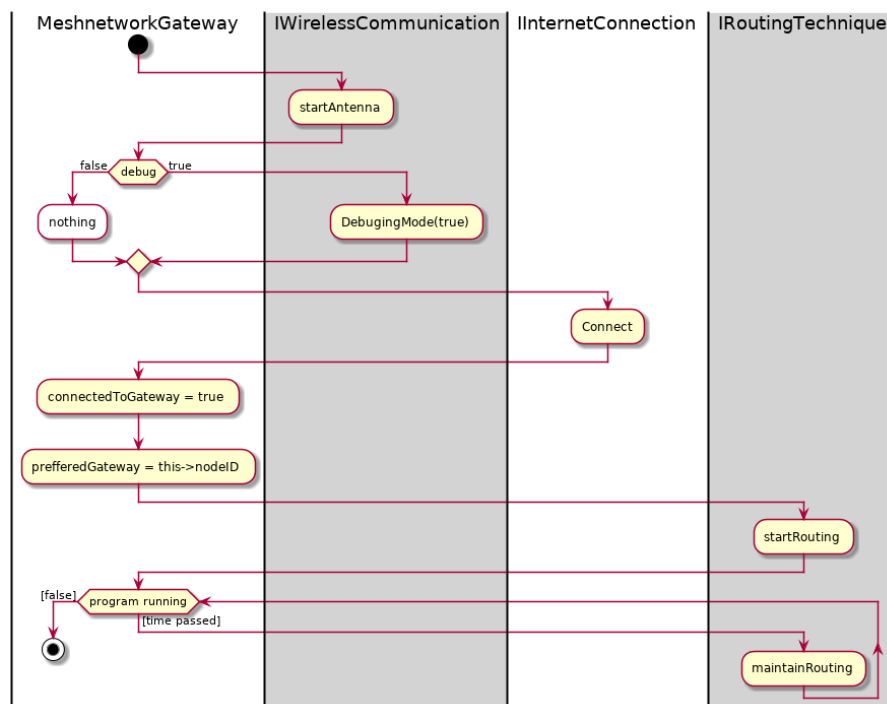
Wanneer een node de hoogste score heeft verzoekt hij de drone om zich te verplaatsen naar de laatst bekende goede locatie. Ook maakt de node dit bekend aan de routingstechniek. Vervolgens stelt de node de locatie van de gateway in als laatste bekende goede locatie.

Tenslotte maakt de node altijd aan het einde van de onderhandeling de lijst waar alle onderhandel waardes in staan leeg.

### Activity diagrammen gateway

In het volgende stuk worden de activity diagrammen behandeld die specifiek gaan over de gateway applicatie. Eerst wordt de algemene flow van de applicatie behandeld en daarna flows binnen de applicatie

**Meshnetwork gateway applicatie** Deze paragraaf beschrijft de algemene flow van de gateway applicatie.



Figuur 3.11: Activity diagram meshnetwork Gateway applicatie.

Zodra de gateway begint start hij meteen de antenne op van de draadloze communicatie. Als debug aan staat verzoekt hij deze stand van de antenne om aan te gaan. Na het starten van de antenne zoekt de gateway een verbinding op met het internet via de aangesloten [IInternetConnection](#) om zo het aansturen vanuit een extern punt van het meshnetwork mogelijk te maken.

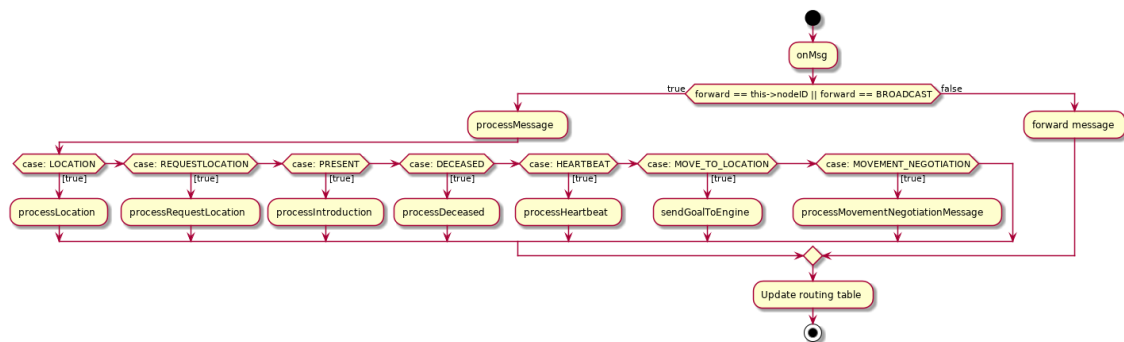
Daarna stelt hij de variabelen in die gebruikt worden in het netwerk om routers een weg naar de router toe te laten vinden. Voor nu worden de waarde dat er een verbinding is met de gateway altijd op waar gezet. In de toekomst kan dit nog veranderd worden door te kijken of er een internetverbinding is opgezet.

Tenslotte begint de gateway met een verzoek naar de routingstechniek om te starten met routeren. Waarna de gateway alleen nog periodiek een verzoek zal sturen om de routing te onderhouden.

### Meshnetwork algemene applicatie

Dit volgende stuk beschrijft de flows in de applicatie die zowel de router als de gateway volgen.

**Meshnetwork flow bij ontvangst bericht** Deze paragraaf beschrijft acties die de applicatie onderneemt bij het ontvangen van een bericht.



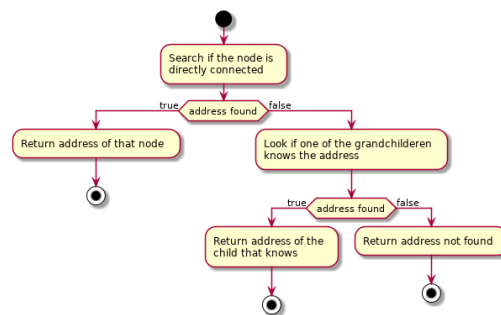
Figuur 3.12: Activity diagram meshnetwork flow ontvangen bericht.

In het activity diagram is te zien dat er bij het ontvangen van een bericht eerst wordt gekeken of het bericht geadresseerd is voor de ontvangende node. Als die niet zo is dan wordt het bericht doorgestuurd naar een volgend punt.

Wanneer het wel geadresseerd is voor de ontvangende node dan gaat het de functie in om het bericht te verwerken. Hoe het bericht verwerkt wordt is afhankelijk van het type die het bericht bij zich draagt. Dit wordt gedaan met een switch case.

Voordat de functie verlaten wordt, zal altijd de routingstechniek geïnformeerd worden over de communicatie.

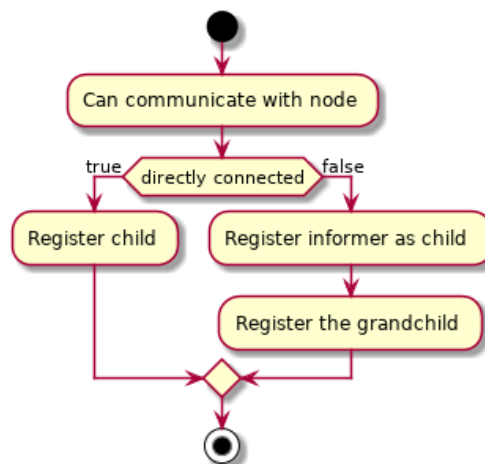
**Routingstechniek voor het vinden van een pad tot het adres** In het communicatie component is een routingstechniek aanwezig voor het vinden van een route tot een pad. Hoe deze techniek het volgende adres vindt in de route wordt getoond in het volgende activity diagram.



Figuur 3.13: Activity diagram vinden pad tot adres.

In het diagram is te zien dat er drie uitgang situaties zijn. Eerst zal de techniek kijken of het adres behoort tot een direct aangesloten node en als dat zo is geeft hij dit terug. Als er direct aangesloten node is gevonden kijkt het systeem of een van de aangesloten nodes het adres kent. Wanneer een aangesloten node het adres kent geeft de routingstechniek het adres van de aangesloten node terug. Tenslotte als niemand het adres kent wordt dit terug gegeven.

**Routingstechniek actie bij bewijs van contact** Als een node bewijs heeft dat er een verbinding is geeft hij dit door aan de routingstechniek die verwerkt dat in de huidige implementatie op de volgende manier.



Figuur 3.14: Activity diagram registeren van contact.

Op het moment dat er contact gelegd is dan wordt er gekeken of dit direct contact was of dat er een aangesloten node een nieuw contact heeft gevonden. Als het direct contact was komt er een nieuwe route aftakking door het toevoegen van een kind van de node. Als er indirect een nieuw contact wordt gevonden dan wordt degene die dit aangeeft geregistreerd als nieuw punt als deze nog onbekend is. Vervolgens wordt het adres die gevonden is toegevoegd aan de set van degene die de vondst aangeeft.

### 3.2.5 Ontwerpkeuzes gemaakt voor het communicatie component

In het volgende stuk worden de keuzes gemaakt voor het component toegelicht.

#### Directe aanroep van verplaatsing voor de routingstechniek

Zodra het meshnetwork component besloten heeft dat hij zich gaat verplaatsen roep hij direct de functie aan waarmee de [IRoutingTechnique](#) geïnformeerd wordt tot een verplaatsing. Deze keuze is gemaakt zodat de routingstechniek meteen weet dat alle routes die hij heeft opgebouwd ongeldig zijn geworden. Het alternatief was om deze aanroep pas te doen op het moment dat een drone geland is. Deze keuze is niet overwogen, omdat de huidige [IDroneEngine](#) nog geen ondersteuning heeft van het doorgeven van een staat van de drone. Op het moment dat de aanroep bij het landen dus zou moeten gebeuren moet er ook een state machine in de drone engine geïmplementeerd worden.

#### Locaties onthouden door het hele bericht op te slaan

Op het moment dat een meshnetwork component een locatie wil onthouden van de gateway of het component die een stap terug staat in het netwerk, doet hij dit door het gehele bericht van een locatieverzoek te onthouden. Hierdoor is er een klein stuk opgeslagen data die niet gebruikt wordt, maar er blijft op een overzichtelijke manier wel data onthouden die er toe doet. Zo kan er in het bericht gezien worden van wie er een locatie is opgeslagen en ook hoe oud dit bericht is. In een toekomstige applicatie kan het nodig zijn om met de leeftijd van het bericht andere beslissingen te maken.

#### Het gebruik van een messagehelper enum

Een enum in C++ heeft de eigenschap dat elk nummer er maar één keer in mag voorkomen. Deze eigenschap wordt gebruikt voor het overzichtelijk toewijzen van plekken in de array van de payload. Door dit te doen kan er tijdens het ontwikkelen gemakkelijk gebruikt gemaakt worden van de namen in de enum. Mochten er ooit plekken veranderen in het bericht hoeft dit ook maar op één plek aangepast te worden. Tevens kan de enum ook gebruikt worden voor het definiëren van het maximale aantal bytes die een array mag bevatten.

#### Extra toevoeging van bestemmingsinformatie in het bericht

In het bericht wordt ook opgeslagen aan wie het bericht verstuurd moet worden naast de informatie voor wie het bericht bestemd is. Hier is voor gekozen zodat er in alle systemen duidelijk is voor wie het bericht is bedoeld is. Zo kan er beter getest worden, is debug informatie duidelijker en kan deze informatie tussen systemen uitgewisseld worden.

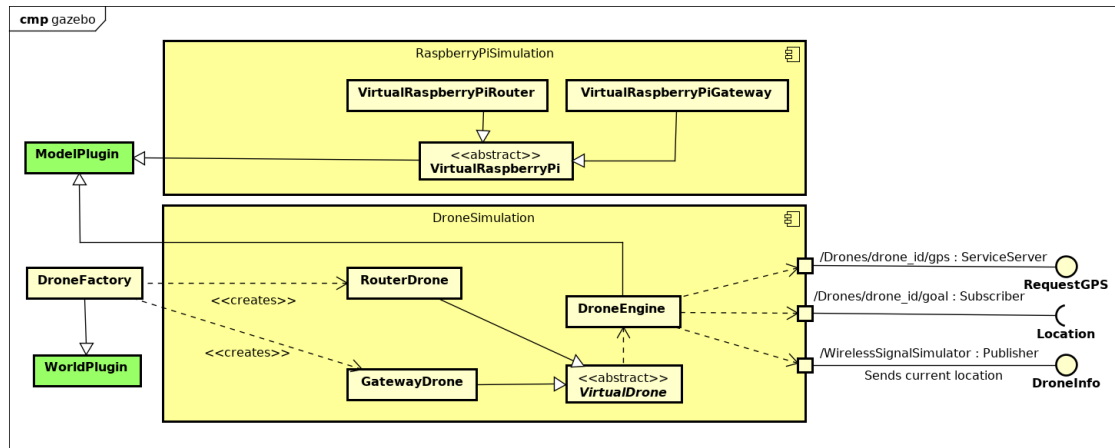
Het alternatief was om berichten binnen het systeem in een struct op te slaan waar de informatie samen met het bericht opgeslagen kon worden.

## 3.3 Design Sub-Systeem Gazebo

Het sub systeem Gazebo is het component die verantwoordelijk is voor de simulatie van de drones. Het is zowel verantwoordelijk voor het initialiseren van de virtuele hardware als

aan elkaar koppelen van deze hardware. Gazebo zal uiteindelijk ook verantwoordelijk zijn voor het toepassen van physics op de virtuele drones.

### 3.3.1 Component Diagram



Figuur 3.15: Component diagram Gazebo.

#### RaspberryPiSimulation

In de dit component wordt een Raspberry Pi gesimuleerd, deze simulatie is puur functioneel. Het heeft geen effect op de clock snelheid deze is gelijk aan de computer waarop de software draait.

#### VirtualRaspberryPi

Dit abstracte component wordt gebruikt ter generalisatie van een Raspberry Pi. Omdat deze Raspberry Pi altijd wordt toegevoegd aan een ander model erft dit component over van ModelPlugin.

#### VirtualRaspberryPiRouter

Deze [VirtualRaspberryPi](#) is voorzien van een applicatie die de Raspberry Pi zich laat gedragen als een router in het meshnetwerk.

#### VirtualRaspberryPiGateway

Deze [VirtualRaspberryPi](#) is voorzien van een applicatie die de Raspberry Pi zich laat gedragen als een gateway in het meshnetwerk.

#### DroneSimulation

Dit component is verantwoordelijk voor het simuleren van drones.

## DroneEngine

Elke [VirtualDrone](#) is voorzien deze drone engine. Dit component maakt het mogelijk voor een drone om zich in een rechte lijn door lucht zich te verplaatsen waarbij het opstijgen en de landing verticaal wordt uitgevoerd. Daarnaast biedt dit component via een rosservice het deel van de interface [IDroneEngine](#) aan om de huidige locatie op te vragen. Via een rostopic kan er een doel gestuurd worden om het andere deel van de net genoemde interface te voorzien. Tenslotte stuurt dit component bij elke verplaatsing de huidige locatie door naar de [WirelessSignalSimulator](#).

## VirtualDrone

Een virtuele drone is een abstract component die alle variabelen bevat die nodig zijn om een drone in de wereld te injecteren. Deze variabelen betreffen een drone id, locatie en verwijzing naar de Gazebo wereld.

## GatewayDrone

Deze drone is een [VirtualDrone](#) die in de SDF omschrijving wordt voorzien van een [VirtualRaspberryPiGateway](#) plugin.

## RouterDrone

Deze drone is een [VirtualDrone](#) die in de SDF omschrijving wordt voorzien van een [VirtualRaspberryPiRouter](#) plugin.

## DroneFactory

Dit component is verantwoordelijk voor het produceren van drones. Het is een World-Plugin die aan de hand van meegegeven parameters gateway en router drones aanmaakt. Een dronefactory is afhankelijk van de SDF omschrijving in een .world bestand die er in factory.world er als volgt uitziet:

```
<plugin name=" DroneFactory" filename=" libDroneFactory .so">
  <amountOfRouterDrones>17</amountOfRouterDrones>
  <amountOfGatewayDrones>1</amountOfGatewayDrones>
  <UseServer>0</UseServer>
  <Debug>1</Debug>
</plugin>
```

**amountOfRouterDrones** Hiermee wordt het aantal in te laden router drones bepaald.

**amountOfGatewayDrones** Hiermee wordt het aantal in te laden gateway drones bepaald.

**UseServer** Wanneer deze op true staat wordt de eerste gateway voorzien van een Rest server.



**Debug** Deze boolean bepaalt of de drones wel of geen debugging topic zullen hebben.

### 3.3.2 Interfaces

In de vorm van Ros transport types voldoet het Gazebo component aan extern aangeboden interfaces. Door gebruik te maken van een ServiceServer [RequestGPS](#) en een subscriber op het topic [Location](#) die samen de functionaliteit bieden benodigd voor de interface [IDroneEngine](#). Door informatie te publiceren op het topic [DroneInfo](#) biedt het informatie per drone aan over locaties.

#### **/Drones/drone\_id/gps**

Elke drone engine maakt een service aan met de naam “/Drones/drone\_id/gps” waarbij de drone.id een variabele is. De structuur van de service ziet er als volgt uit.

```
float32 latitude
float32 longitude
int16 height
```

Deze service heeft geen parameters waaraan voldaan moet worden. Als response geeft het de huidige locatie terug van de drone.

#### **/Drones/drone\_id/goal**

Door te luisteren naar dit topic is het mogelijk om doelen te sturen naar de drone. Dit gebeurt door Location berichten te sturen naar de drone welke de volgende structuur hebben.

```
float32 latitude
float32 longitude
int16 height
```

Bij ontvangst stelt de DroneEngine de ontvangen locatie in als doel.

#### **/WirelessSignalSimulator/**

Tenslotte publiceert de drone informatie op dit topic voor de [WirelessSignalSimulator](#). Hiervoor gebruikt de drone het bericht DroneInfo die de volgende structuur heeft:

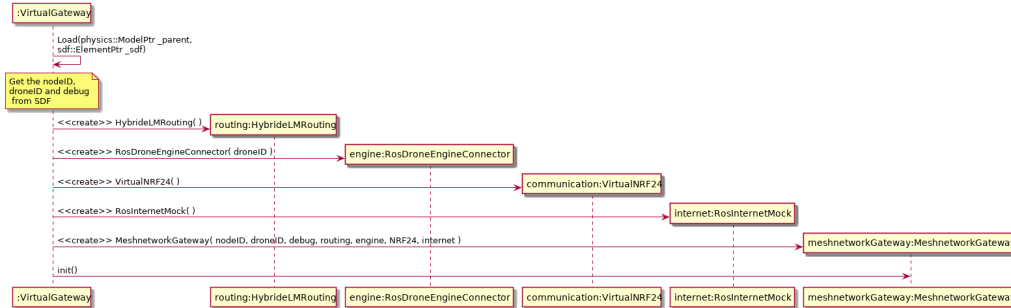
```
uint8 nodeID
float32 [3] position
string sub
bool on
```

De DroneEngine vult alleen de informatie uit het bericht die hij bekend heeft. Dit houdt dus in dat het ID van de drone verstuurd wordt en de positie waar die op dit moment is.

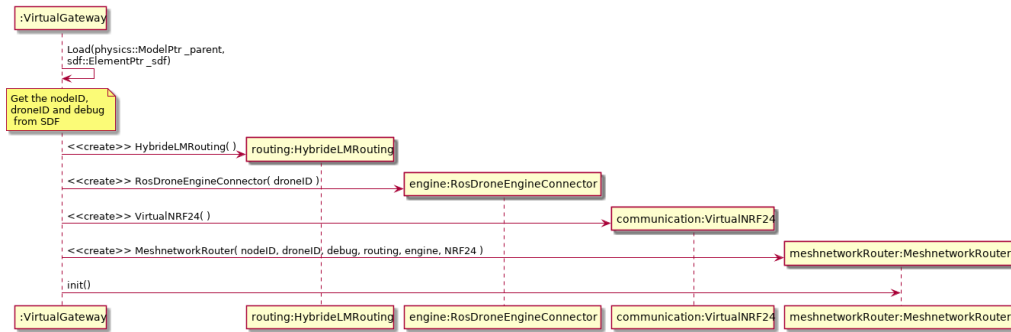
### 3.3.3 Sequence Diagrams

In de onderstaande diagrammen wordt de sequentie weergegeven die uitgevoerd wordt in het component Gazebo. Eerst worden de virtuele RaspberryPi's behandeld waarna de dronefactory vervolgens aan bod komt.

**Opstarten virtuele RaspberryPi** Doordat er maar een minimaal verschil zit tussen het opstarten van een virtuele router Raspberry Pi en een virtuele gateway Raspberry Pi worden deze samen behandeld.



Figuur 3.16: Sequence diagram initialiseren van een virtuele gateway Raspberry Pi.

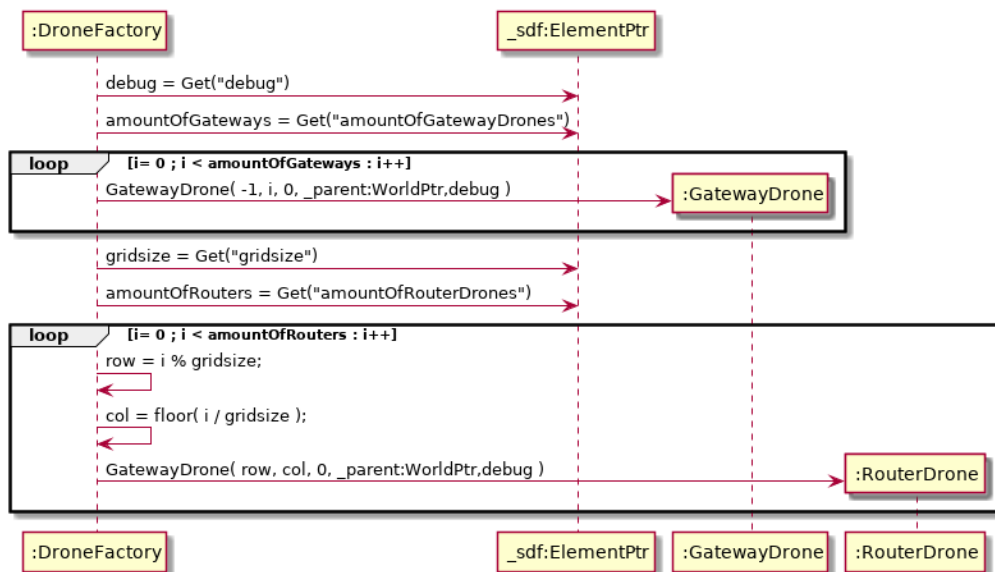


Figuur 3.17: Sequence diagram initialiseren van een virtuele router Raspberry Pi.

In beiden gevallen begint het opstarten van de virtuele Raspberry Pi met het uitlezen van SDF waarden waar de parameters NodeID, DroneID en debug uit gehaald wordt. Vervolgens wordt er een instantie van de routingstechniek, DroneEngine connector en virtuele NRF24 aangemaakt. In het geval van de gateway wordt er ook nog een RosInternetMock aangemaakt.

Tenslotte wordt er een meshnetwork gateway of router aangemaakt die een verwijzing meekrijgt naar de zojuist aangemaakte componenten. Als laatste stap wordt het meshnetworkcomponent aangeroepen om te starten met een init functie.

**Drone factory** In de volgende sequentie wordt het opstarten van een dronefactory behandeld.



Figuur 3.18: Sequence diagram over het opstarten van een dronefactory.

Bij het opstarten van een drone factory wordt de SDF geraadpleegd hoeveel gateway aangemaakt moeten worden. Vervolgens zal de factory per gateway een instantie aanmaken die zichzelf via een SDF omschrijving injecteert in Gazebo. Deze SDF omschrijving wordt later in deze paragraaf verder toegelicht, tegelijk met die van een router drone.

Nadat de factory alle gateway drones heeft gemaakt worden de router drones gemaakt. Voordat hij de drones aanmaakt, raadpleegt de factory het SDF bestand nogmaals om de variabele gridsize in te stellen. Gridsize bepaalt hoe breed elke kolom maximaal mag zijn. Nadat de RouterDrones zijn aangemaakt is het factory programma klaar en zal het niets meer uitvoeren tijdens de simulatie.

**SDF omschrijvingen drones** Een drone inject bij de constructor een SDF omschrijving van zichzelf in Gazebo. Deze omschrijvingen worden nu behandeld.

Een SDF omschrijving van een router drone ziet er als volgt uit:

```

<sdf version = '1.6 '>
<model name = 'router_drone '>
  <static>1</static>
  <pose>x y z 0 0 0</pose>
  <link name = 'link '>
    <inertial>
      <pose>x y z 0 0 0</pose>
    </inertial>
    <collision name = 'collision '>
      <geometry>
        <box size = '0.5 0.5 0.5'></box>
      </geometry>
    </collision>
    <visual name = 'visual '>
      <geometry>
        <box size = '0.5 0.5 0.5'></box>
      </geometry>
      <material>
  
```

```

<script>
  <uri>file://media/materials/scripts/Gazebo.material</uri>
  <name>Gazebo/Green</name>
</script>
</material>
</visual>
</link>
<plugin name="MeshnetworkRouter" filename="libMeshnetworkRouter.so">
  <DroneID> droneID </DroneID>
  <nodeID> droneID </nodeID>
  <Debug> debug </Debug>
</plugin>
<plugin filename = 'libDroneEngine.so' name ='DroneEngine'>
  <DroneID> droneID </DroneID>
</plugin>
</model >
</sdf>

```

Een SDF omschrijving van een gateway drone ziet er bijna hetzelfde uit behalve dat deze aanspraak maakt op de plugin “MeshnetworkGateway” in plaats van “MeshnetworkRouter”. De gebruikte kleur paars is in plaats van groen.

Belangrijke punten uit deze omschrijving zijn:

**static:** Door deze op waar te zetten wordt de drone niet meegenomen in de physics engine.

**x y z:** Deze worden gebruikt voor het positioneren van de drone.

**Material:** Hierin wordt een script aangeroepen om een kleur toe te wijzen aan de drone.

**plugin name="MeshnetworkRouter" filename="libMeshnetworkRouter.so"** Deze plugin zorgt ervoor dat een drone voorzien wordt van een [VirtualRaspberryPiRouter](#).

**plugin name="MeshnetworkGateway" filename="libMeshnetworkGateway.so"** Deze plugin zorgt ervoor dat een drone voorzien wordt van een [VirtualRaspberryPiGateway](#). Deze plugin is niet zichtbaar in het bovenstaande SDF voorbeeld, maar wordt wel gebruikt in het SDF format voor een gateway.

**plugin filename = 'libDroneEngine.so' name ='DroneEngine'** Deze plugin zorgt ervoor dat een drone wordt voorzien van een [DroneEngine](#).

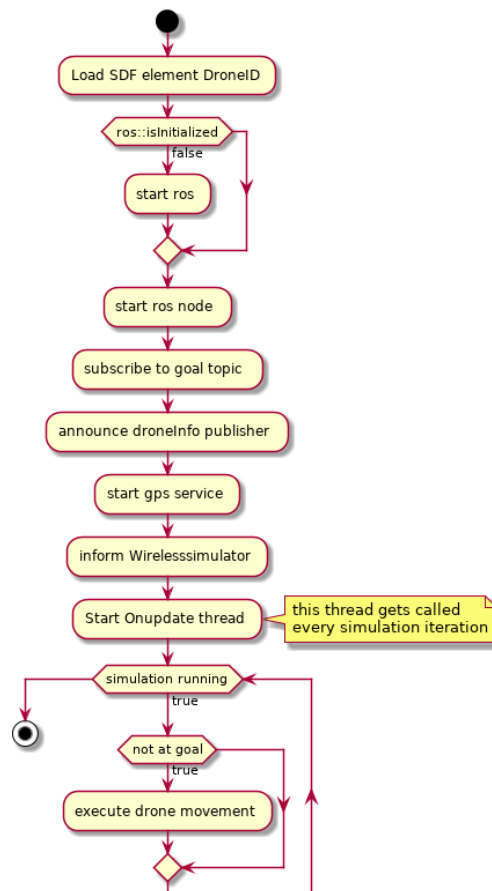
**DroneID:** Dit zal het ID zijn van de drone zelf en wordt aan de motor gekoppeld, deze moet uniek zijn tijdens de simulatie. Anders zullen er meerdere drones aangestuurd worden met een enkel id.

**nodeID:** Dit zal het ID zijn van het aangesloten netwerkcomponent. Ook deze moet uniek zijn in de huidige implementatie.

**Debug:** Op het moment dat deze aanstaat, zal een netwerkcomponent zijn debug stand gebruiken.

### 3.3.4 Activity Diagrammen

**DroneEngine applicatie** Elke drone is voorzien van een DroneEngine, dit component zorgt ervoor dat de drone in de simulatie kan voortbewegen op basis van doelen. Daarnaast biedt het de optie om de locatie van een drone op te vragen om zo een GPS na te bootsen.

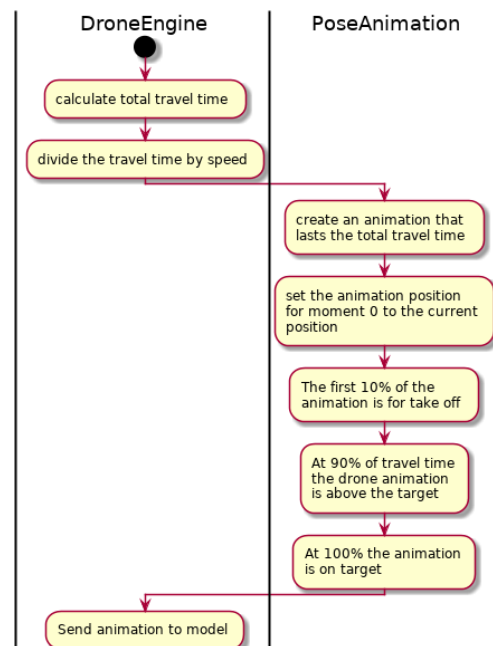


Figuur 3.19: Activity diagram initialiseren van de DroneEngine.

Op het moment dat een DroneEngine gestart wordt is het ophalen van de DroneID het eerste wat het component doet. Vervolgens controleert het component of Ros al gestart is. Wanneer deze nog niet gestart is wordt deze alsnog opgestart. Vervolgens wordt er een eigen rosnode gecreëerd. Deze rosnode wordt vervolgens gebruikt om te luisteren naar het topic waarvan de drone zijn doel locatie binnen krijgt. Hierna start de node een topic met droneinformatie waar de locatie van de drone en het id in staat. Vervolgens wordt er ook de GPS service gestart. Nu dit allemaal opgestart is wordt er een bericht gestuurd naar de wireless simulator met de huidige drone informatie.

Om het initialiseren af te maken wordt er tenslotte een thread gestart die elke simulatie cycle wordt aangeroepen. Deze thread controleert per cycle of de drone op zijn huidige doel is. Als deze dat niet is wordt er een functie aangeroepen om de drone naar zijn doel toe te laten bewegen. Hoe deze animatie werkt, wordt in het hierop volgende activity diagram uitgelegd

**Uitvoeren van een beweging van de drone** Voor het uitvoeren van een beweging maakt de drone gebruik van een animatie. Dit is mogelijk, omdat de DroneEngine een modelplugin is waardoor het aanspraak kan maken op de pose van de aangesloten drone.



Figuur 3.20: Activity diagram vinden pad tot adres.

Op het moment dat er een beweging verzocht wordt, begint de drone met het berekenen wat de directe afstand is tussen de drone en het doel. Zodra de afstand berekend is wordt de afstand gedeeld door de snelheid. Deze berekening geeft de tijd aan de beweging zal kosten.

De eerste stap van de animatie is het instellen waar de animatie begint op tijdstip 0, in deze implementatie is dat op de locatie waar de drone al is. Vervolgens moet de drone opstijgen dit duurt 10 procent van de totale tijd die de drone beweegt. De hoogte van het opstijgen heeft een verband met afstand, dit is namelijk de helft hiervan. Vervolgens wordt op 90 procent van de animatie een doel ingesteld om weer op dezelfde hoogte boven het doel te hangen. Tenslotte moet de animatie bij 100 procent landen op het ingestelde doel.

Als deze volledige animatie is ingesteld wordt dit verstuurd naar het model om uit te voeren.

### 3.3.5 Ontwerpkeuzes gemaakt voor het Gazebo component

In het volgende stuk worden de keuzes gemaakt voor het Gazebo component toegelicht.

#### Abstracte Raspberry Pi

In de software worden verschillende Raspberry Pi's in de wereld van Gazebo geladen. Omdat elke type Raspberry altijd een aparte plugin is zorgt deze overerving ervoor dat dit ook altijd zo is.

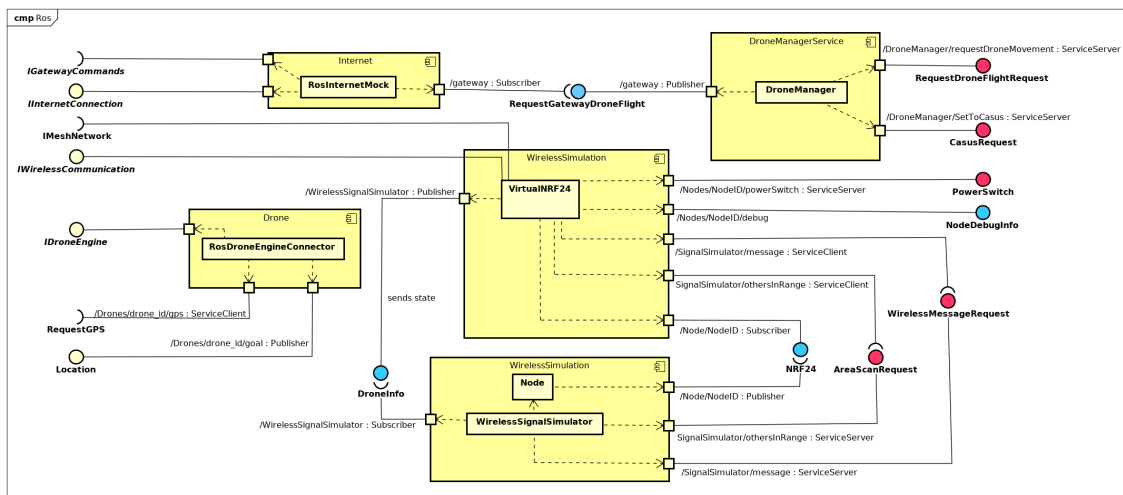
### Doelen doorgeven met een rostopic

Bij een echte drone is het vrij aannemelijk dat de aansluiting via een seriële verbinding verloopt. Het gedrag van een rostopic komt meer overeen met een seriële verbinding dan het alternatief, rosservice. In de veel implementaties van een seriële verbinding wordt namelijk altijd een verbinding open gehouden.

## 3.4 Design Sub-System Ros

Het volgende gedeelte zal gaan over het pakket Ros. Dit onderdeel is verantwoordelijk voor het simuleren van alle vormen van communicatie die gebruikt worden in het meshnetwerk. Het simuleert een NRF24 inclusief de draadloze communicatie. Het biedt een adapter tussen de [DroneEngine](#) en [MeshnetworkComponent](#). Tenslotte heeft het een onderdeel die zich voordoeft als internetcomponent.

### 3.4.1 Component Diagram



Figuur 3.21: Component diagram Ros.

#### RosDroneEngineConnector

Dit is een adapter tussen de [DroneEngine](#) en de interface [IDroneEngine](#). Die het mogelijk maakt om Ros te gebruiken om aanspraak te maken op de virtuele drone.

#### DroneManagerService

Dit component is de toegangspoort voor de ontwikkelaar tot de [MeshnetworkGateway](#). Op dit moment is het component alleen geschikt om verplaatsingverzoeken te versturen via de gateway naar de drones. Hiervoor biedt het twee interfaces, in de vorm van rosservices, aan waarbij er één is voor verplaatsingen naar een locatie is en de ander voor casusverzoeken.

## DroneManager

Dit subcomponent realiseert de rosservices en publiceert verplaatsingverzoeken naar de gateways.

## RosInternetMock

Deze internetmock laat de [DroneManager](#) zich voordoen als internetpunt zodat er geen daadwerkelijke TCP/IP implementatie hoeft worden gemaakt

## WirelessSimulation

Het WirelessSimulation component is verantwoordelijk voor het simuleren van de NRF24 en het draadloze signaal hiervan.

## WirelessSignalSimulator

Deze simulator bepaald of twee nodes met elkaar mogen communiceren. Dit wordt gedaan met informatie die het ontvangt via het drone informatie topic.

## Node

Dit component wordt alleen gebruikt door de [WirelessSignalSimulator](#) om te registreren welke Nodes bestaan, waar ze zijn en naar welk topic ze luisteren.

## VirtualNRF24

Dit component is de virtuele versie van de NRF24, het is in staat om payloads te verwerken van 32 byte. Deze kan de NRF24 naar een direct adres versturen of zenden naar alle NRF24 nodes binnen bereik.

## Extern aangeboden interfaces

**IIInternetConnection** Deze interface wordt aangeboden door [RosInternetMock](#) biedt daarmee de optie om een internet verbinding op te zetten.

```
void init (Communication::Internet::IGatewayCommands *IGC, uint8_t threads = 1)
void connect( )
void disconnect( )
```

**init** Deze functie zal het internet component initialiseren Als parameter wordt een verwijzing naar een aangeboden interface verwacht waar de internetconnectie gebruik van mag maken om commando's het netwerk in te sturen. Optioneel kan aangegeven wordt hoeveel threads de server mag gebruiken standaard is dit er een.

**preconditie:** Component die een internetverbinding wil kunnen maken kan de interface [IGatewayCommands](#) aanbieden. Het adres die het component publiceert is beschikbaar

**postconditie:** Het component die externe services kan aanbieden is geïnitialiseerd.



**Connect** Deze functie wordt aangeroepen om een verbinding op te zetten. *preconditie:* Het component is geïnitieerd. *postconditie:* Er is een externe verbinding buiten het meshnetwerk opgezet geschikt om commando's het netwerk in te sturen.

**Disconnect** Deze functie wordt aangeroepen om een internet verbinding te verbreken. *preconditie:* Er is geen preconditionie aan deze functie. Wanneer er nog geen verbinding bestaat zal de uitkomst hetzelfde blijven *postconditie:* Er bestaat geen internetverbinding meer.

**IWirelessCommunication** Deze interface wordt gebruikt om een draadloos communicatie middel aan te kunnen bieden.

```
void StartAntenna(IMeshNetwork* IMN)
void StopAntenna( )
bool SendMessageTo(const uint8_t* msg)
void BroadcastMessage(const uint8_t* msg)
void DebuggingMode(IMeshDebugInfo* IMD, const bool on)
const bool On( )
```

**StartAntenna** Deze functie wordt gebruikt om de antenne te starten. Het verwacht een pointer terug naar het MeshNetwork waar de ontvangen berichten aangegeven kunnen worden.

*preconditie:* De antenne is aangesloten op een component die berichten van minimaal 32 byte kan verwerken.

*postconditie:* De antenne is gestart en klaar om berichten te ontvangen.

**StopAntenna** Door het aanroepen van deze functie zal de aangesloten antenne stoppen.

*preconditie:* Er is geen preconditionie.

*postconditie:* De antenne is gestopt.

**SendMessageTo** Deze functie verwacht een verwijzing naar een 32 byte array die vervolgens verstuurd zal worden

*preconditie:* Er is een bericht opgesteld om verstuurd te worden.

*postconditie:* Het bericht is verstuurd naar de geadresseerde.

**BroadcastMessage** Deze functie zal een 32 byte bericht versturen op het algemene kanaal van de NRF24 waardoor dus elke node binnen bereik dit zal ontvangen

*preconditie:* Er is een bericht opgesteld die naar iedereen in de buurt verstuurd zal worden.

*postconditie:* Elke node binnen bereik van deze antenne heeft het bericht ontvangen.

**DebuggingMode** Deze functie verwacht een verwijzing naar debuginformatie over het meshnetwerk zodat dit samen met debuginfo van de antenne zelf gepubliceerd kan worden.

*preconditie:* De aanvrager van deze functie beschikt over de interface [IMeshDebugInfo](#).

*postconditie:* Er wordt debuginformatie gepubliceerd.

**IDroneEngine** Deze interface wordt gebruikt om een drone aan te sturen en zijn locatie op te kunnen vragen. Hij wordt aangeboden door [RosDroneEngineConnector](#).

```
virtual void turnOn( ) = 0;
virtual void turnOff( ) = 0;
void setGoal(const float latitude, const float longitude, const float height)
const Vector3< float > getLocation( )
```

**turnOn** De functie wordt gebruikt om drone aan te zetten.

*preconditie:* De drone staat uit.

*postconditie:* De drone staat aan en kan aangestuurd worden om zich te verplaatsen.

**turnOff** De functie wordt gebruikt om drone uit te zetten.

*preconditie:* De drone staat aan.

*postconditie:* De drone staat uit en kan niet aangestuurd worden om te verplaatsen.

**SetGoal** De functie wordt gebruikt om een doel in te stellen voor de drone om zich naartoe te verplaatsen.

*preconditie:* Er is geen preconditie voor deze functie.

*postconditie:* Er is een doel ingesteld in de drone waarnaar hij zich moet verplaatsen.

**getLocation** Deze functie wordt gebruikt om de huidige locatie van een drone op te vragen er wordt een latitude, longitude en een height verwacht in de vorm van een Vector3.

*preconditie:* De drone heeft kennis van zijn huidige locatie.

*postconditie:* De huidige locatie van de drone is berekend en teruggegeven.

### Extern aangeboden Ros interfaces

Via rosservices en topic biedt het component ook externe functionaliteit en functies aan deze worden hieronder behandeld.

**/Drone/drone\_id/goal** De [RosDroneEngineConnector](#) publiceert doelen voor de drone op dit topic. De doelen worden gepubliceerd met een Location message welke de volgende structuur heeft.

```
float32 latitude
float32 longitude
int16 height
```

**/DroneManager/requestDroneMovement** Deze service wordt aangeboden door de [DroneManager](#) het biedt de service aan om verzoeken te sturen naar de gateway om locatie-verplaatsingen uit te voeren. Het maakt gebruik van het servicebericht RequestDroneFlight. De structuur van dit bericht is als volgt.

```
uint8 ID
float32 latitude
float32 longitude
int16 height
-----
uint8 status
```

**ID** Is de parameter die gebruikt wordt om de ID van de te verplaatsen drone aan te geven.

**latitude, longitude, height** Zijn de parameters die gebruikt worden op de gewenste locatie op te geven.

**status** Deze response geeft aan of het de manager is gelukt om het verzoek te versturen naar de gateway.

**preconditie:** De drone manager is verbonden met een drone gateway.

**postconditie:** Er is een bericht gestuurd naar de gateway met een verzoek om een locatie te veranderen.

**/DroneManager/SetToCasus** De service die hier aangeboden wordt is een functie om drones te verplaatsen naar een casus positie. Om deze functie te gebruiken wordt een casus bericht verwacht deze heeft de volgende structuur.

```
uint32 caseID
```

Er is een casus geïmplementeerd uit het plan van aanpak. Deze casus gebruikt 17 drones. Daarom kan er met caseID 0 alle drones naar die casus positie gezet worden en met 1 t/m 17 worden ze individueel naar deze positie gebracht. Met casus 100 wordt er een grid gemaakt van 10 bij 10 drones wanneer er minder aanwezig zijn vullen ze plek 0 tot het aantal wat aanwezig is. Casus 101, 102, 201, 202 worden gebruik voor de situaties uit het onderzoek.

**preconditie:** De drone manager is verbonden met een drone gateway

**postconditie:** Er zijn één of meerdere verplaatsingverzoeken verstuurd naar de gateway om de drones naar hun casus positie te zetten.

**/Nodes/NodeID/powerSwitch** Deze service wordt aangeboden door het component [VirtualNRF24](#) en representeert een aan en uit knop voor een antenne. De service maakt gebruik van een PowerSwitch bericht die de volgende structuur kent:

```
bool power
```

De boolean die in het bericht zit bepaald of de antenne aan of uit wordt gezet.

**preconditie:** Er is geen post conditie een NRF24 kan altijd aan of uit gezet worden.

**postconditie:** De antenne staat in de opgegeven aan of uit staat.

**/Nodes/NodeID/debug** Het volgende topic wordt gepubliceerd om de ontwikkelaar van informatie te voorzien over de node en aangesloten antenne. [VirtualNRF24](#) is de aanbieder van dit topic. Het bericht wat gepubliceerd wordt op dit topic is een NodeDebugInfo bericht die de volgende structuur heeft:

```
uint8  nodeID
bool   ConnectedWithGateway
uint16 familySize
uint32 totalMessages
uint8  prefferedGateWay
bool   on
uint8  hops
uint8  prefLoc
```

**nodeID** Het ID van die de antenne gebruikt.

**ConnectedWithGateway** boolean die aangeeft of de node een verbinding heeft met een gateway.

**familySize** Het aantal nodes die de huidige node kent. Niet directe nodes kunnen dubbel voorkomen.

**totalMessages** Het totaal aantal berichten die een node verstuurt heeft.

**prefferedGateWay** Het ID van de gateway waar de node voorkeur voor heeft.

**on** Boolean die aangeeft of de antenne aan staat.

**hops** Het aantal hops die een node nodig heeft om de gateway te bereiken.

**prefLoc** Het ID van een andere node waar deze node voorkeur voor heeft om in nood naartoe te verplaatsen.

#### Intern aangeboden Ros interfaces

**/gateway** Het gateway topic is de verbinding tussen de [DroneManager](#) en de [RosInternetMock](#). Omdat er op dit moment alleen ondersteuning is voor verzoeken tot het verplaatsen van een drone wordt er gebruik gemaakt van het bericht RequestGateway-Droneflight. Dit bericht heeft de volgende samenstelling:

```
uint8 ID
float32 latitude
float32 longitude
int16 height
```

**ID** Het ID wordt gebruikt om aan te geven welke drone zich moet verplaatsen.

**latitude, longitude, height** Zijn de parameters die gebruikt worden op de gewenste locatie op te geven.

**/Node/NodeID** Elke [VirtualNRF24](#) maakt dit topic aan waarbij NodeID slaat op het ID van de node. Dit topic wordt gebruikt om NRF24 berichten te ontvangen van andere virtuele nodes via de [WirelessSignalSimulator](#). De berichten die over dit topic verzonden worden zijn NRF24 berichten die de volgende structuur hebben:

```
uint8 [32] payload
```

**payload** Een NRF24 is in staat berichten te versturen van maximaal 32 byte daarom telt dit ook voor het virtuele NRF24 bericht.

**/SignalSimulator/othersinRange** Deze service wordt aangeboden door de [WirelessSignalSimulator](#). Het biedt de functie voor een node om op te vragen wie er allemaal binnen bereik is. Het opvragen gaat via het servicebericht AreaScan die er als volgt uit ziet:

```
uint8 id
-----
uint8 [] near
```

**id** Deze parameter geeft aan voor welke node gekeken moet worden wie er allemaal binnen bereik is.

**near** Als response wordt er een vector terug geven met alle nodes die in buurt zijn.

**/SignalSimulator/message** Om een bericht te kunnen versturen biedt de [WirelessSignalSimulator](#) een service aan. Deze service verstuurd een bericht naar een andere Node. De service doet dit met opgelegde regels of het mogelijk is een bericht te kunnen versturen. Het servicebericht wat gebruikt wordt is een [WirelessMessage](#) bericht met de volgende samenstelling:

```
uint8 from
uint8 to
NRF24 message
-----
bool succes
```

**from** Het ID van de node die het bericht wil versturen.

**to** Het ID van de node waaraan het bericht verstuurd wordt.

**NRF24** Het bericht die verstuurd moet worden.

**/WirelessSignalSimulator** De [WirelessSignalSimulator](#) is het component die luistert naar dit topic. De berichten die hierop gepubliceerd worden zijn essentieel voor de simulator. Dit omdat uit de berichten gehaald kan worden welke virtuele antennes bestaan en op welke positie zij zich bevinden. Deze informatie wordt gepubliceerd met het berichttype DroneInfo die er als volgt uit ziet:

```
uint8 nodeID
float32 [3] position
string sub
bool on
```

**NodeID** Het ID van de node waar de informatie betrekking op heeft.

**position** De positie waar de node zich bevindt gevuld met de volgende array [longitude][latitude][height].

**sub** Dit is de naam van het topic waar de [VirtualNRF24](#) op luistert.

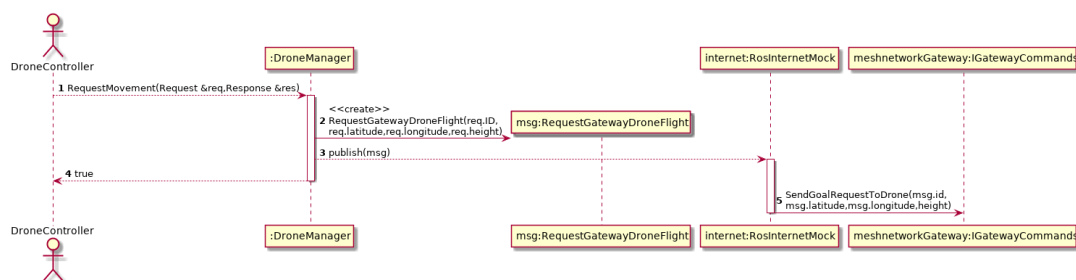
**on** Deze boolean geeft aan of de antenne aan of uit staat.

### 3.4.2 Sequence Diagrams

Binnen het Ros component vindt veel communicatie tussen componenten plaats door het gebruik van de Ros transport laag. Hoe deze communicatie verloopt wordt beter toegelicht met de volgende componenten.

#### Verzoek tot verplaatsing van een drone

Een dronemanager biedt de service aan een verzoek te versturen om een drone te verplaatsen. De dronemanager doet dit door te communiceren met het [RosInternetMock](#) component. Deze werking staat hieronder uitgelegd.



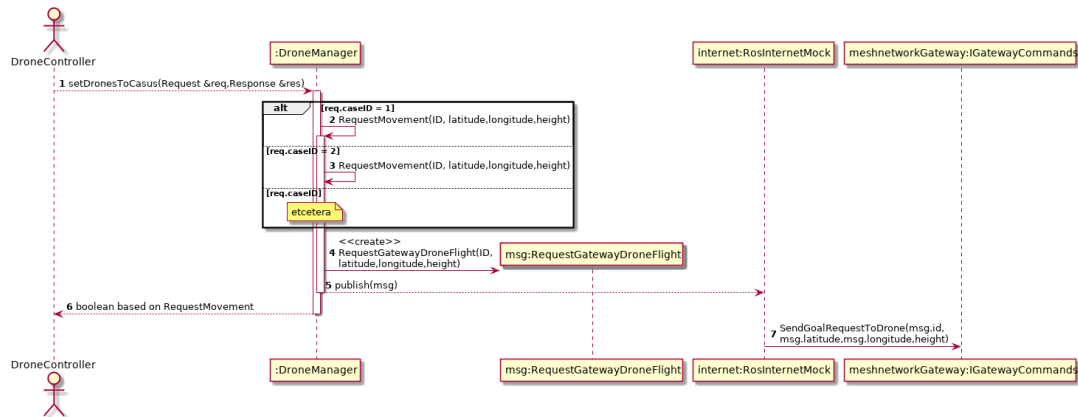
Figuur 3.22: Sequence diagram waarbij een DroneManager verzocht wordt een drone te verplaatsen.

Een actor kan een verzoek doen aan de drone manager om een drone te verplaatsen. De drone manager verwacht hier een DroneFlightRequest bericht voor waar in staat wie er moet verplaatsen en waarheen. Op basis van dit bericht maakt de DroneManager een

RequestGatewayDroneFlight bericht aan en publiceert dit op het /gateway topic. Elke RosInternetMock zal dit bericht ontvangen en versturen naar de [IGatewayCommands](#) waar deze op aangesloten zit. Per [MeshnetworkGateway](#) bestaat er een RosInternetMock.

### Verzoek tot casus situatie

Om situaties makkelijk te kunnen testen heeft de drone manager een service die het aanbiedt de drones in een bepaalde casus verdeling te zetten.

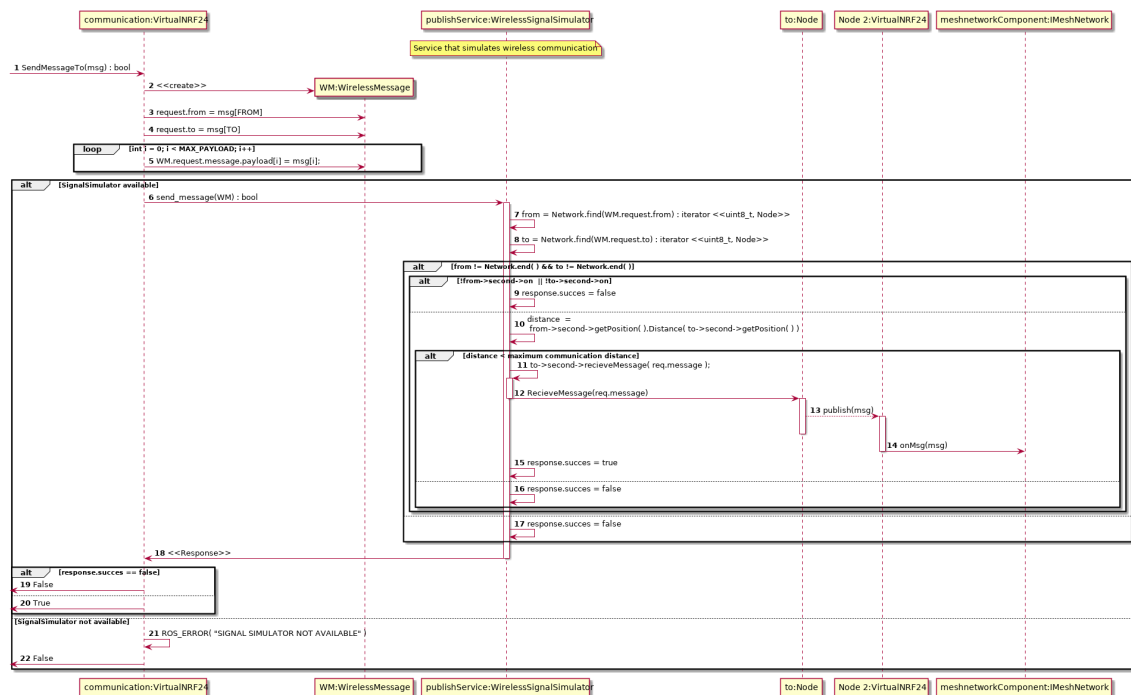


Figuur 3.23: Sequence diagram waarbij een DroneManager verzocht wordt een casus situatie te starten.

De functie werkt grotendeels gelijk aan die van een verzoek tot een verplaatsing van een drone. Er wordt gebruik gemaakt van een switch case die reageert op het CaseID die gestuurd wordt. Vervolgens kunnen daardoor een of meerdere RequestMovements aangeroepen worden om de drones te verzoeken te verplaatsen naar de casus positie.

### Versturen bericht van VirtualNRF24 tot VirtualNRF24

Elk [MeshnetworkComponent](#) is voorzien van een [VirtualNRF24](#). Deze kunnen met elkaar communiceren, maar om een draadloze situatie na te bootsen gebeurt dit via de [Wireless-SignalSimulator](#). Deze simulator bepaalt of dit mag met de onderlinge afstand en de aan of uit stand van de antenne.



Figuur 3.24: Sequence diagram voor het versturen van een bericht van NRF24 tot NRF24.

Op het moment dat een **VirtualNRF24** een verzoek tot het versturen van een bericht ontvangt, maakt het een **WirelessMessage** aan. Hij stelt de zender en ontvanger van het bericht in en kopieert het bericht naar de payload van **WirelessMessage**. Vervolgens wordt de service van de **WirelessSignalSimulator** voor het versturen van berichten aangeroepen om de **WirelessMessage** te versturen. Als deze service niet beschikbaar is wordt er een **ROS\_ERROR** geprint zichtbaar in stap 21.

Als de service wel beschikbaar is gaat deze als eerste opgegeven kijken of hij de zender en ontvanger kent. Vervolgens kijkt de **WirelessSimulator** of beide antennes aan staan. Wanneer een van de twee niet bekend is of er staat er één uit zal er een false als response terug gegeven worden.

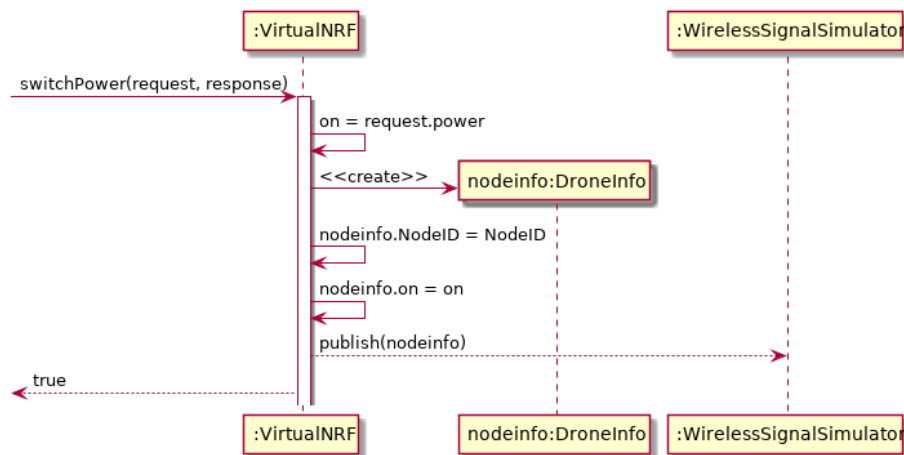
Als dit niet het geval is zal er berekend worden wat de afstand is tussen de zender en de ontvanger. Op het moment dat deze kleiner is dan de maximaal toegestane communicatie afstand wordt het bericht verstuurd naar het topic waar de ontvangende **VirtualNRF24** op luistert, en wordt het succes van de response op true gezet.

De zendende NRF24 geeft de response van de **WirelessSignalSimulator** terug om de functie te sluiten.

### Powerswitch service van de Virtuele NRF24

Elke **VirtualNRF24** biedt een service aan om de NRF24 aan of uit te zetten. Op het moment dat deze service wordt aangeroepen vindt het volgende plaats.



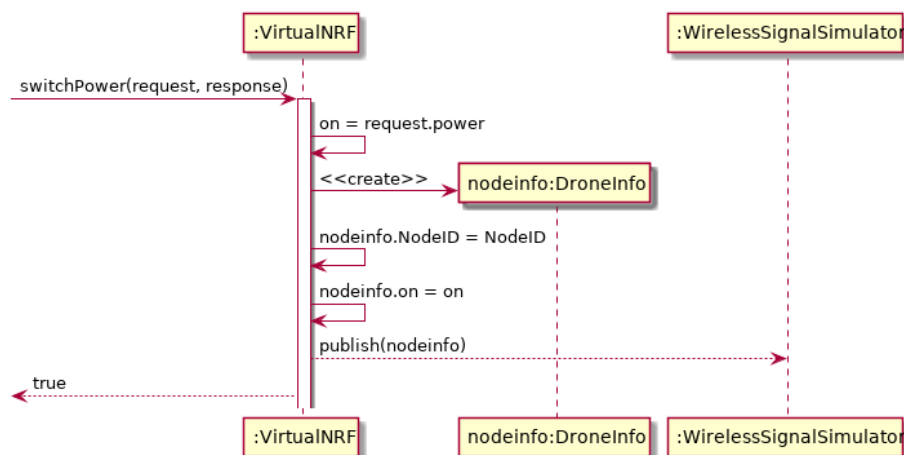


Figuur 3.25: Sequence diagram power switch Virtuele NRF24.

Zodra de functie aangeroepen wordt zet de antenne zijn staat gelijk aan die van de request. Vervolgens maakt hij een DroneInfo bericht aan waar ingezet wordt wat de nieuwe staat en welk ID. Dit bericht wordt tenslotte gepubliceerd naar de [WirelessSignalSimulator](#).

### Sturen doel naar drone

De [RosDroneEngineConnector](#) biedt in zijn aangeboden interface twee functies aan. Een functie is het versturen van een doel naar de [DroneEngine](#).

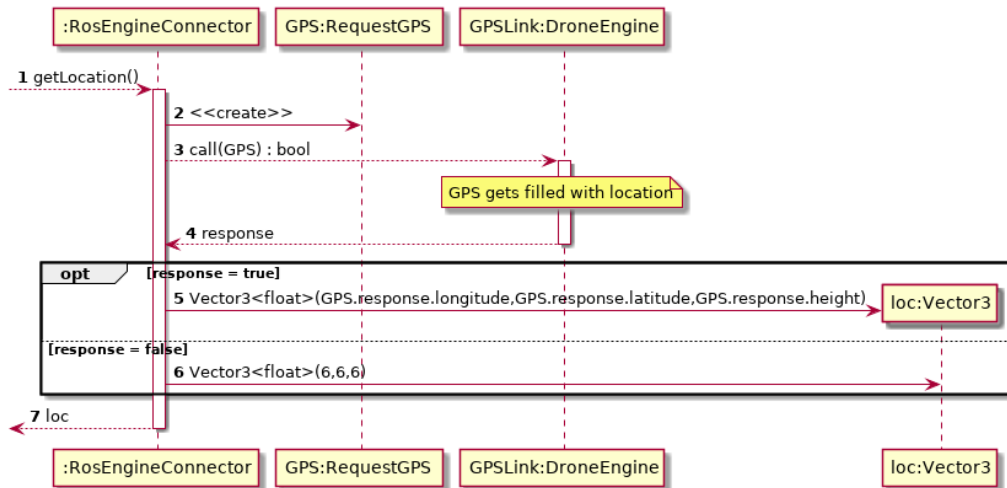


Figuur 3.26: Sequence diagram sturen doel naar de DroneEngine.

Om een doel te versturen maakt de RosEngineConnector een Location bericht aan waarin de locatie staat waar de drone zich naartoe moet verplaatsen. Deze publiceert hij vervolgens op het topic van de DroneEngine.

### Ophalen locatie van de drone

De [RosDroneEngineConnector](#) biedt in zijn aangeboden interface de functie aan om een locatie van een drone op te vragen. Dit werkt als volgt

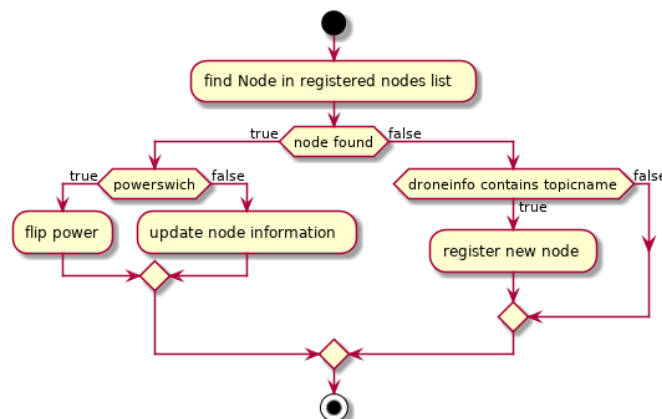


Figuur 3.27: Sequence diagram power switch Virtuele NRF24.

Als er een verzoek om een locatie gemaakt wordt, maakt de RosEngineConnector een RequestGPS bericht aan. Vervolgens gebruikt hij dit bericht in een servicecall naar de [DroneEngine](#) die als response de locatie terug zal geven. Met deze waarden wordt een Vector3 aangemaakt welke vervolgens teruggegeven wordt. Er is nog geen goede afhandeling wanneer de DroneEngine niet beschikbaar is. Nu wordt er locatie (6,6,6) teruggegeven.

### 3.4.3 Activity Diagrammen

**Verwerken drone informatie WirelessSimulator** Er is een generiek topic van de wireless simulator waar zowel de [DroneEngine](#) als de [VirtualNRF24](#) informatie op posten. Om dit te scheiden neemt de WirelessSimulator de volgende stappen bij het ontvangen van een bericht.

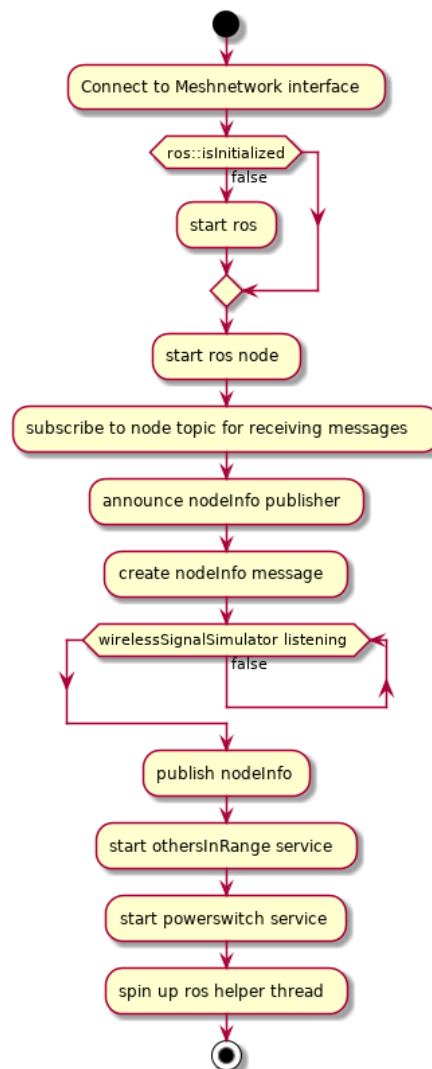


Figuur 3.28: Activity diagram initialiseren van de DroneEngine.

Wanneer er een bericht ontvangen wordt met drone informatie kijkt de WirelessSimulator eerst of de node al bekend is bij hem. Als deze nog niet bekend is kijkt de simulator of het bericht een string bevat met op welk topic de node luistert. Als niet in het bericht staat wordt de functie verlaten, wanneer dit wel zo wordt de node opgeslagen.

Wanneer het bericht van een bekende node af komt, wordt er gekeken of er een power switch is. Als dit zo is past de simulator ook de switch in zijn register voor die node aan zodat die weet in welke staat die staat. Bij een bericht waar de stand van de power niet wisselt, wordt de node zijn informatie bijgewerkt met de meest recente informatie.

**Start antenne VirtualNRF24** Omdat er tijdens het opstarten van een virtuele antenne veel stappen worden ondernomen wordt het volgende activity diagram gebruikt om dit proces extra toe te lichten.



Figuur 3.29: Activity diagram initialiseren van de virtuele antenne.

Als een antenne start legt die als eerste verbinding met zijn vereiste interface [IMeshNetwork](#). Vervolgens controleert de antenne of Ros al draait als dit nog niet zo is start hij

deze op. Hierna start de antenne een rosnode op die hij gebruikt voor Ros transport. Zodra deze node opgestart is start de antenne met luisteren naar zijn eigen topic. Als dit topic aangemaakt is maakt hij een Ros publisher aan voor de nodeInfo. Vervolgens wacht de antenne tot de wirelessSignalSimulator luistert naar het topic. Wanneer deze luistert, publiceert de antenne de informatie. Vervolgens worden de services opgestart om nodes binnen bereik te vinden en de powerswitch. Tenslotte wordt er een thread opgestart voor het bufferen van Ros berichten.

#### 3.4.4 Ontwerpkeuzes gemaakt voor het Ros component

In het volgende stuk worden de keuzes gemaakt voor het component toegelicht.

##### Verbindingsmogelijkheid op basis van onderlinge afstand

Er is de keuze gemaakt om te bepalen of twee antennes mogen communiceren op basis van de onderlinge afstand die ze hebben. Deze afstand wordt bepaald door het toepassen van Pythagoras wat een relatief simpele berekening is. Hoewel er in de werkelijkheid veel meer factoren komen kijken is deze implementatie het minst complex. De complexiteit om rekening te houden van met deze factoren kan wel erg toenemen. Een voorbeeld is het uitrekenen van de lengte van de route tussen twee punten, om zo rekening te houden met objecten. Dit zou een groot effect hebben bij een simulatie van honderd drones.

##### Wireless simulator in Ros in plaats van Gazebo

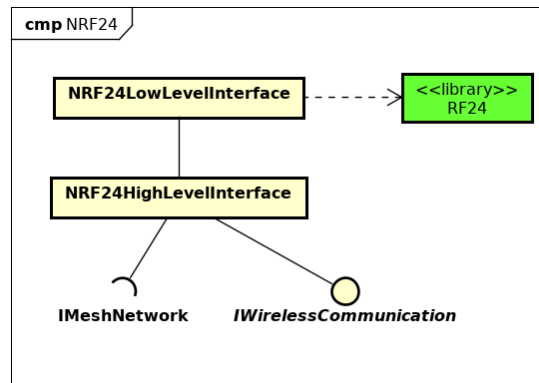
Eerst was de WirelessSignalSimulator geplaatst in het Gazebo component. Toen viel op dat dit een hoge coupling gaf. Het was ook logisch om de simulatie van communicatie bij elkaar te bundelen in één package. Daarom is besloten om deze te plaatsen in Ros in plaats van Gazebo.

##### RosDroneEngineConnector

Er wordt een adapter in Ros geplaatst om een drone in Gazebo aan te sturen. Het zou software technisch gezien logischer zijn om de drone aan te sturen met een Gazebo adapter die een gelijk soort transport systeem heeft als Ros. Er is gekozen om dit toch aan Ros over te laten om zo alle communicatie via hetzelfde transport te laten verlopen. Dit zorgt ervoor dat het Ros programma RQT inzicht geeft aan de ontwikkelaar over alle communicatie op één plek.

## 3.5 Design Sub-System NRF24

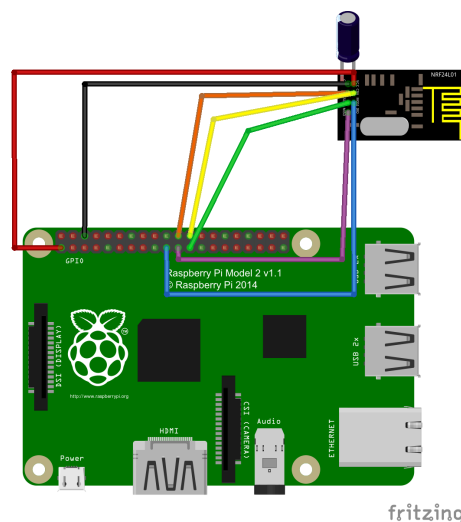
### 3.5.1 Component Diagram



Figuur 3.30: Component diagram Ros.

Het component diagram bestaat uit twee interfaces. Een high level interface en een low level interface ook wel de driver genoemd. De high level interface biedt de interface **IWirelessCommunication** aan. Omdat deze al volledig behandeld is in [3.4.1 IWirelessCommunication](#) wordt deze niet nogmaals behandeld. Er wordt gebruik gemaakt van de library RF24 (nRF24, 2019) voor directe aansturing naar het hardwarecomponent.

### 3.5.2 Aansluiting NRF24



Figuur 3.31: Aansluiting nRF24 op een Raspberry Pi.

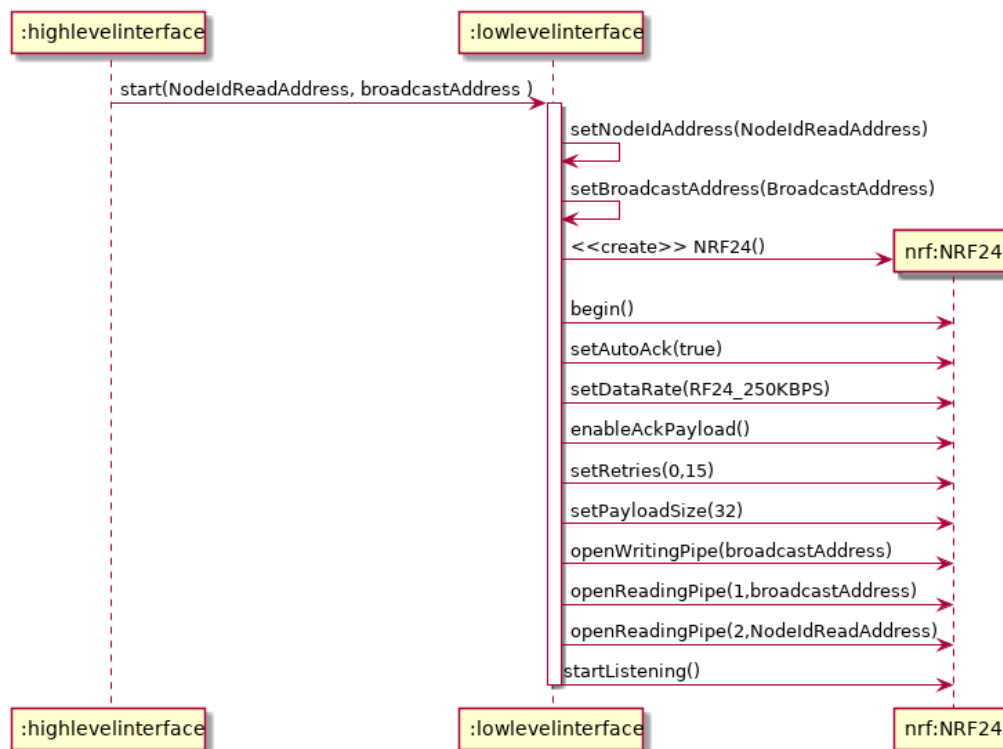
Voor een stabielere voltage wordt een condensator van  $4.7\ \mu\text{F}$  aanbevolen direct aangesloten op de ground en voltage poort van de NRF24L01+.

Raspberry Pi	NRF24L01+	Kleur
6 / GND	GND	Zwart
1 / 3.3V DC	VCC	Rood
22 / GPIO25	CE	Oranje
24 / GPIO 8	CSN/CS	Geel
23 / GPIO11 / SPI_CLK	SCK	Groen
19 / GPIO10 / SPI_MOSI	MOSI	Blauw
21 / GPIO9 / SPI_MISO	MISO	Paars

Tabel 3.1: Aansluitingen NRF24L01+ op een Raspberry Pi model 2B+.

### 3.5.3 Sequence Diagrams

**Starten van de NRF24 driver** Op het moment dat de driver van de NRF24 gestart wordt moet deze de NRF module correct instellen. Het instellen gebeurt door het volgende uit te voeren.

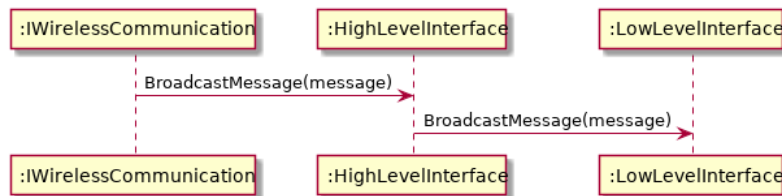


Figuur 3.32: Sequence diagram instellen NRF24.

Zodra de NRF24 gestart wordt moet hij over twee adressen geïnformeerd worden. Het unieke adres voor deze NRF24 en het algemene broadcast adres waar alle NRF24's op werken. Omdat de NRF24 driver een rij van adressen heeft die hij allemaal in de gaten moet houden wordt een aparte functie gebruikt voor het instellen van de adressen. Vervolgens kan de driver een instantie maken van de NRF24. De library van de NRF24 vereist dat als eerste functie altijd `begin()` aangeroepen wordt. De acknowledgement wordt aangezet waarmee er geverifieerd kan worden dat een bericht is aangekomen. De snelheid van de

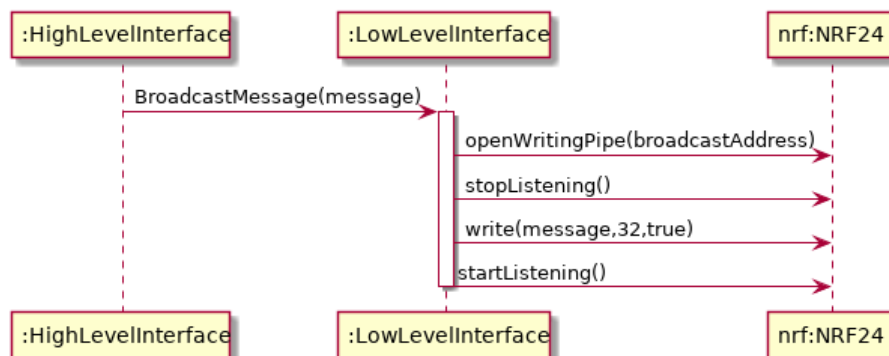
band wordt ingesteld op 250 kbitps en de payload grootte op het maximaal toegestane van 32 bytes. Vervolgens wordt het broadcast adres ingesteld als schrijfadres en worden de eerste twee leesadressen ingesteld op het broadcast adres en het eigen adres van de node. Tenslotte wordt de NRF verzocht om te starten met luisteren waarna de functie eindigt.

**Broadcast message op de high level interface** Het broadcasten van een message is een functie die de high level interface aanbiedt. Hier komt zoals te zien in [Figuur 3.33](#) weinig bij kijken, omdat het bericht direct doorgegeven kan worden aan de driver.



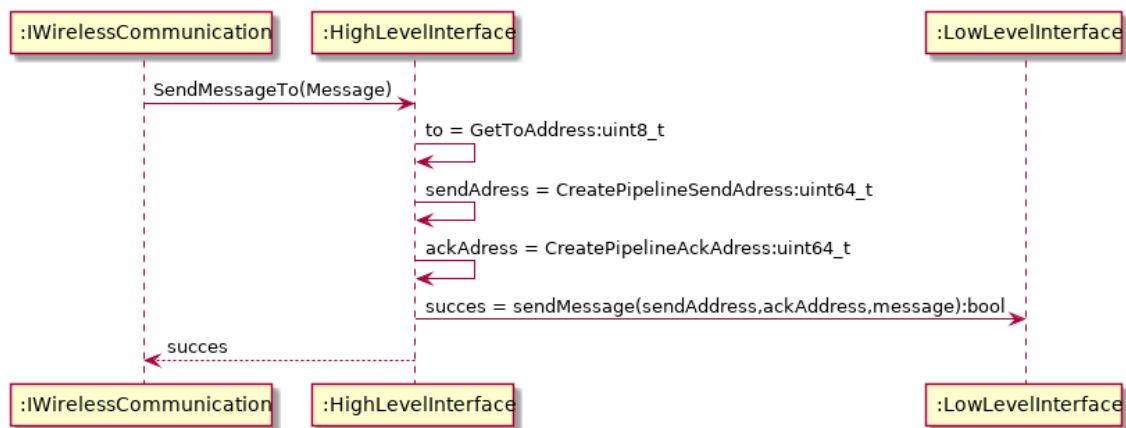
Figuur 3.33: Sequence diagram broadcast message op de high level interface.

**Broadcast message op de low level interface** Zodra de low level interface aangeroepen wordt om een bericht te broadcasten opent hij de pipe naar het algemene broadcast adres, stop hij met luisteren, verstuurt het bericht en begint daarna weer met luisteren.



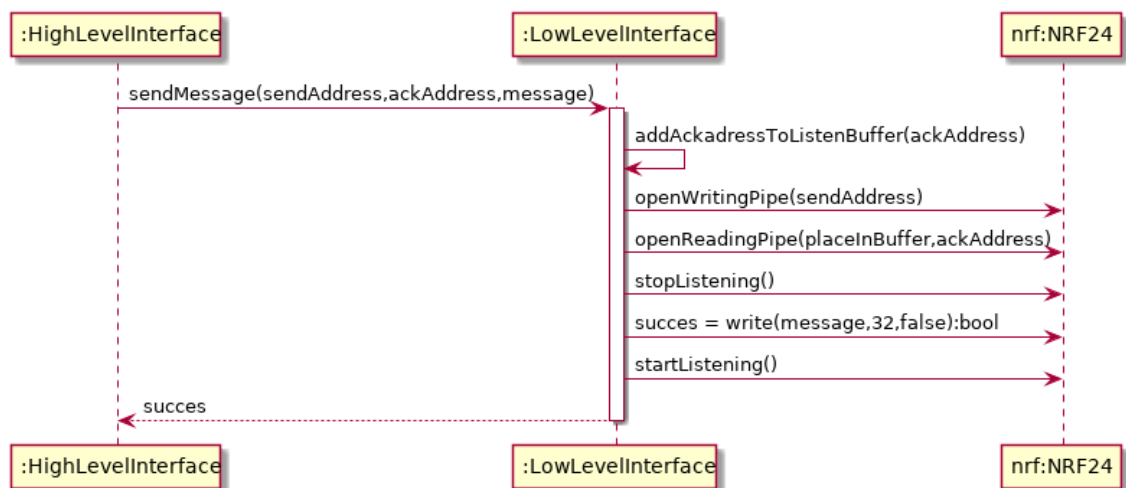
Figuur 3.34: Sequence diagram broadcast message op de low level interface.

**Bericht versturen high level interface** Als de high level interface verzocht wordt tot het versturen van een bericht naar een specifiek adres creëert de high level interface het adres op basis van het id voor het sturen en ontvangen acknowledgement. Nadat dit aangemaakt is wordt het bericht samen met de twee adressen doorgegeven aan de driver. De functie geeft een boolean terug met het aangegeven succes van de driver.



Figuur 3.35: Sequence diagram bericht versturen high level interface.

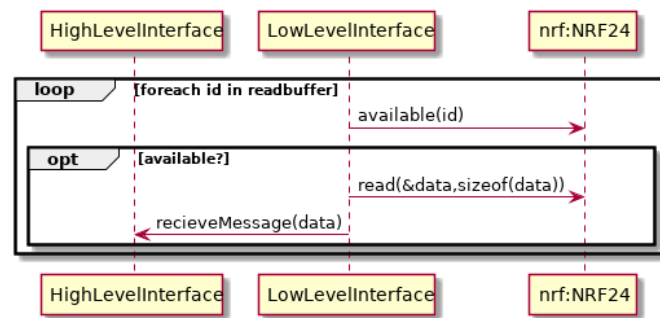
**Bericht versturen low level interface** Wanneer de low level interface verzocht wordt om een bericht te versturen voegt hij eerst het acknowledgement adres toe aan zijn lijst met adressen waar hij naar luistert. Vervolgens wordt het adres waar naartoe geschreven moet worden ingesteld op de schrijf pipe en het adres waar de acknowledgement op ontvangen wordt ook toegewezen op een pipe. Tenslotte wordt het bericht verstuurd terwijl er tijdelijk gestopt wordt met luisteren. De functie retourneert of het zenden gelukt is op basis van de acknowledgement.



Figuur 3.36: Sequence diagram bericht versturen low level interface.

**Het ontvangen van een bericht** De driver van de NRF24 zal constant de adressen aangemeld in het buffer controleren of er een bericht beschikbaar is. Zodra dit het geval is geeft de driver dit bericht door aan de high level interface.

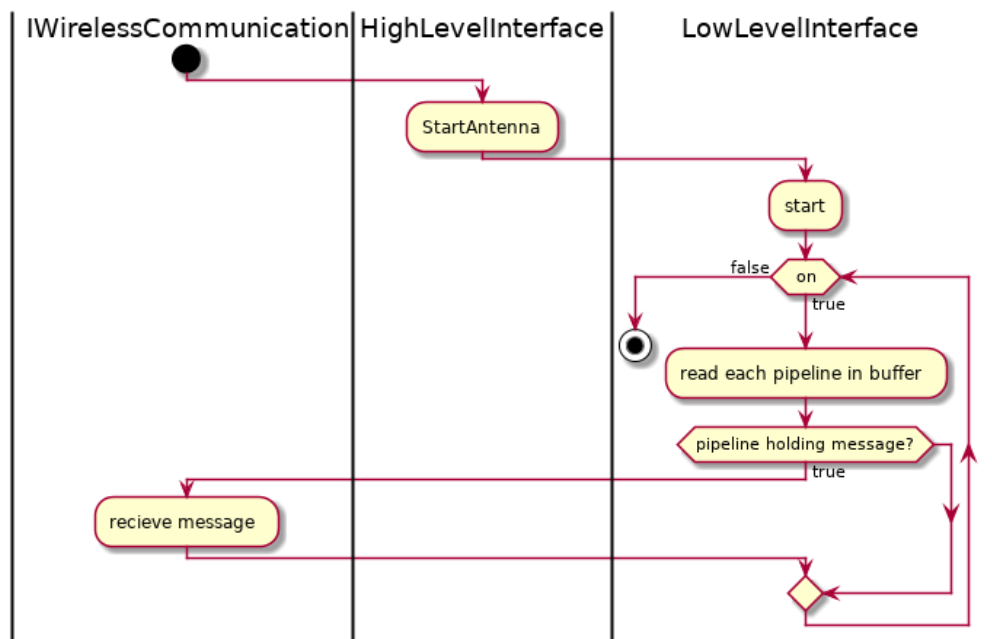




Figuur 3.37: Sequence diagram ontvangen bericht.

### 3.5.4 Activity Diagrams

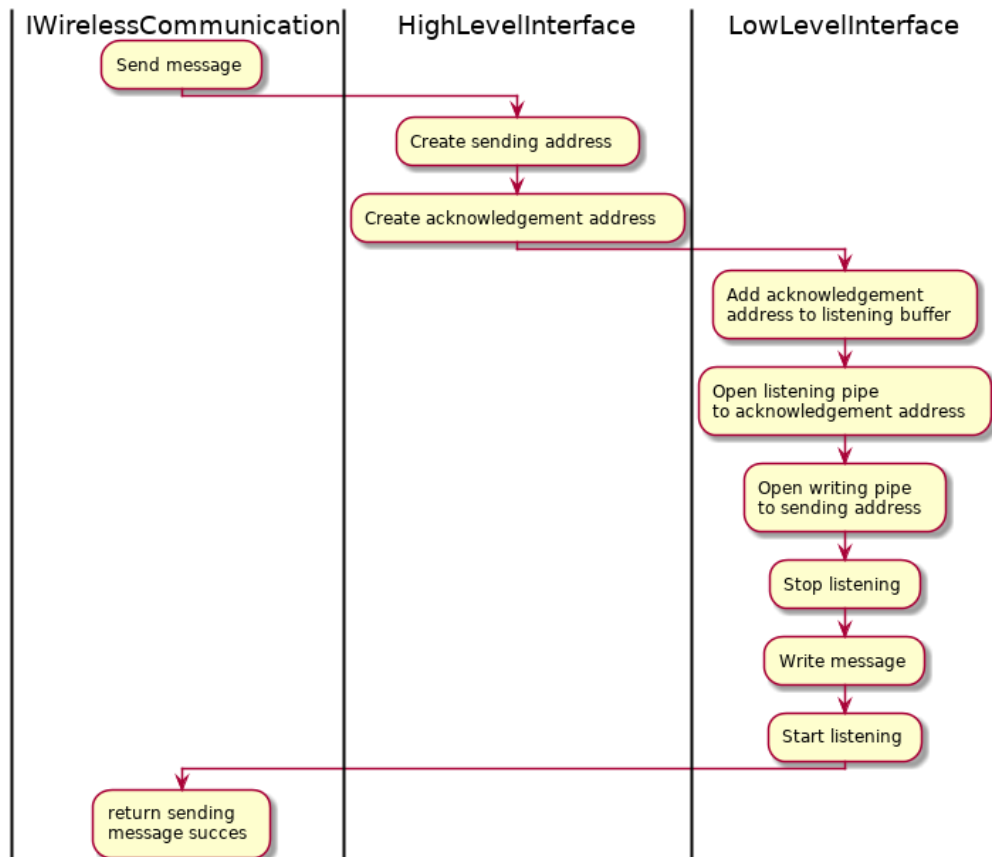
**Starten antenne** Op het moment dat de antenne aangezet wordt via de interface die de NRF24 aanbiedt, zal er het volgende ondernomen worden zichtbaar in [Figuur 3.38](#)



Figuur 3.38: Activity diagram van het starten van een antenne.

Op het moment dat de antenne gestart wordt, roept deze de driver aan om de antenne hardwarematig op te starten, en te initialiseren. Vervolgens zal de driver zolang die aan staat continue elke pipeline controleren of er een bericht beschikbaar is. Als die dit zo is geeft de low level interface die via de high level interface terug op de vereiste interface [IMeshNetwork](#)

**Versturen bericht** Voor het versturen van een bericht loopt het NRF24 component meerdere stappen door. Deze worden onder [Figuur 3.39](#) toegelicht.



Figuur 3.39: Activity diagram over het versturen van een bericht.

Zodra de **IWirelessCommunication** verzocht wordt tot het versturen van een bericht komt het bericht eerst aan in de high level interface. De haalt uit het bericht wie de geadresseerde is en maakt hiervoor een pipeline adres aan en een adres voor het ontvangen van een bevestiging. Waarna dit gedaan is geeft de high level interface dit door aan de driver. Die zal op zijn beurt de adressen juist registreren en vervolgens het bericht versturen. Op het sturen van een bericht zal meteen een bevestiging ontvangen worden of dit goed is gegaan. Het resultaat wordt teruggegeven aan de **IWirelessCommunication**.

### 3.5.5 Ontwerpkeuzes gemaakt voor het NRF24 component

In het volgende stuk worden de keuzes gemaakt voor het component toegelicht.

#### Pipe verdeling

Er is gekozen om een vaste pipe verdeling te houden die uit een statisch stuk en een circulaire buffer bestaat. De antenne kan op één pipe tegelijk schrijven. Daarom wordt er per schrijf actie van pipe gewisseld. Er zijn twee adressen waar altijd interesse naar is om naar te luisteren dit zijn het broadcast adres en het eigen adres van de node. Deze staan daarom vast. Dan blijven er drie plekken over om adressen in te zetten. Deze worden circulair gebruikt waardoor altijd het adres die het oudste is verwijderd wordt.

Deze adressen worden alleen gebruik voor acknowledgement het is daarom aannemelijk dat het oudste adres ook niet meer gebruikt gaat worden.

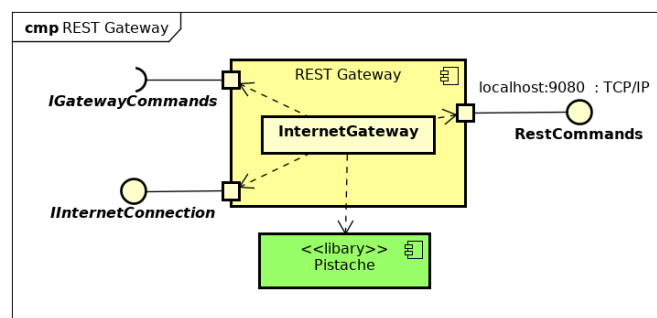
### High en Low level interface

Hoewel er ook nog een aparte library is voor de NRF24l01+ wordt er toch nog een high en low interface toegepast. Dit is gedaan om specifieke technische aanroepen uit de high level interface weg te houden.

## 3.6 REST gateway

De volgende sectie zal de werking beschrijven van de REST gateway. Deze gateway maakt het mogelijk om via een REST service commando's het netwerk in te sturen. De implementatie betreft een prototype daarom is er een beperkt aantal functies aanwezig en leunt de applicatie grotendeel op de applicatie Pistache (Stefani, z. j.).

### 3.6.1 Component diagram



Figuur 3.40: Component diagram REST gateway.

### InternetGateway

Deze class is het enige zelfgemaakte gedeelte uit dit component. Het draagt de verantwoording van het opstarten en het draaiende houden van de REST service die aangeboden wordt via de interface RestCommands. Om deze service aan te kunnen bieden maakt het gebruik van de library Pistache (Stefani, z. j.).

### 3.6.2 Interfaces

Het component biedt twee interface aan. De interface [IInternetConnection](#) wordt al behandeld in het pakket van Ros en wordt daarom hier niet nogmaals behandeld.

## RestCommands

Via een REST protocol de gateway een interface aan voor aansturing. De interface luistert hiervoor op het ipadres van het apparaat en gebruikt de poort 9080.

De interface heeft twee aanwezige routes om een functie aan te bieden.

```
POST: /DroneMovement/:id/:latitude/:longitude/:height
GET:  /ready
```

**POST: /DroneMovement/:id/:latitude/:longitude/:height** Via deze route kunnen verzoeken verstuurd worden om een drone te verplaatsen. Door in de post aan te geven welke drone naar welke punt toe moet zal de gateway dit bericht vertalen naar een bericht gebruikt in het meshnetwerk en dit vervolgens versturen.

**preconditie:** De drone is aanwezig in het meshnetwerk.

**postconditie:** Het verzoek naar de drone om zich te verplaatsen is verstuurd.

**GET: /ready** Deze functie biedt de optie aan om te controleren of de REST service al is opgestart.

**preconditie:** -

**postconditie:** Er is een response over de staat van de REST service.

## Literatuur

- Van Heesch, U. (2016, 23 september). *Software design description template*.
- nRF24. (2019). *nrf24/rf24: Optimized fork of nrf24l01 for arduino & raspberry pi/linux devices*. Op 10 juni 2019 verkregen van <https://github.com/nRF24/RF24>
- Stefani, M. (z. j.). *Pistache*. Op 10 juni 2019 verkregen van <http://pistache.io/>