Bachelor Scriptie

Ultrawide Band in mobiele applicaties en specifiek Flutter

Door: Leroy Hendrikus Lambertus Geurts

Leerlingnummer: 336179

Bedrijfsbegeleider: Harry ten Berge

Schoolbegeleider: Dick Heijink

Bedrijf: Baseflow

School: Hogeschool Saxion te Enschede

12 juni 2022

Samenvatting

In dit document vindt U het onderzoek naar het gebruik van ultrawide band technologie op het mobiele platform. Dit document bevat in specifiek een onderzoek naar de werking van ultra wideband, de beschikbaarheid van ultrawide band op het mobiele platform en de ondersteuning voor UWB voor development mogelijkheden. Hierbij worden de mogelijkheden onderzocht die ultrawide band op dit moment biedt op het mobiele platform en wordt een oplossing geboden om deze technologie naar het Flutter platform toe te halen aan de hand van een Flutter plugin. Hoe deze plugin tot stand is gekomen en hoe de uitwerking er hiervan uitziet is onder andere terug te lezen in dit verslag.

Inhoudsopgave

1 Introductie

Dit hoofdstuk bevat de algehele introductie van de afstudeeropdracht. In dit hoofdstuk komt onder andere de aanleiding, probleem definitie en doel aan bod. Vervolgens wordt de hoofdvraag gedefinieerd en worden de ondersteunende deelvragen gegeven. Tot slot wordt de scope van de opdracht besproken.

1.1 Aanleiding

Op 20 april 2021 introduceert Apple een nieuwe innovatie onder de naam AirTags. Een kleine accessoire waarmee je belangrijke spullen makkelijk weer terug vindt via de 'Zoek mijn'-app van Apple (Apple, 2021). De technologie waarmee deze zoekfunctie onder andere wordt mogelijk gemaakt bevindt zich deels in de nieuwe U1 chips die zijn te vinden in de iPhones 11, 12 en 13. Deze chip maakt gebruik van de zogenaamde ultra wideband technologie, hierna te vermelden als "UWB". UWB technologie bestaat al langer maar lijkt door onder andere Apple op dit moment gepopulariseerd te worden voor verschillende doeleinden. Niet veel later komt Samsung ook met hun eigen tracker, de Samsung Galaxy SmartTag+, ook gebaseerd op deze technologie. En heeft ook Google op 19 oktober dit jaar aangekondigd dat hun nieuwste flagship, de Google pixel 6 Pro, UWB technologie gaat ondersteunen met de uitrol van Android versie 12 (Google, 2021).

Baseflow werkt onder andere veel op het gebied van app development met Flutter en Flutter plug-ins. En gaan zich onder andere ook bezighouden met ontwikkelingen op het gebied van internet of things. Nu er verschillende API's worden vrijgegeven om te kunnen interacteren met deze UWB chips in de nieuwere toestellen is het voor Baseflow interessant om te onderzoeken wat de mogelijkheden van deze technologie zijn en hoe deze technologie naar het Flutter platform toe te halen is. Hierbij wil Baseflow de mogelijkheden onderzoeken voor een Flutter plug-in en onderzoeken wat Android en Apple op dit moment aan mogelijkheden bieden voor deze technologie. Uiteindelijk zou Baseflow een proof of concept willen zien die duidelijk weergeeft waar deze technologie tot toe in staat is.

1.2 Probleem definitie

Verschillende vraagstukken komen voort naar aanleiding van dit probleem. Baseflow wil eigenlijk meerdere zaken onderzoeken waarvan zij op dit moment nog geen kennis over beschikken. Te beginnen bij wat is de UWB technologie die zich nu lijkt te verspreiden in onder andere de nieuwere iPhones en straks de Google pixel 6. En hoe werkt deze technologie op technisch vlak? Vervolgens wordt er gekeken naar de huidige mogelijkheden op gebied van zowel Android als iOS. Welke attributen zijn er al? Welke API's en welke apparaten ondersteunen deze technologie? En wat zijn de verschillende mogelijkheden die kunnen worden gerealiseerd met deze technologie?

Hierna wil Baseflow weten wat de mogelijkheden zijn om deze technologie naar het Flutter platform toe te halen. Hoe zal de plug-in hiervoor eruit zien en in hoeverre is het mogelijk om een plug-in te creëren waarbij de API voor zowel Android en iOS generiek is.

Tot slot zou Baseflow graag een applicatie als proof of concept willen zien die de mogelijkheden en de functionaliteiten van deze technologie goed weergeeft om op deze manier een inzicht te krijgen in de potentie van deze technologie.

1.3 Doel

Het doel van deze opdracht is Baseflow antwoord te geven op hun vragen op gebied van UWB technologie. Hierbij wil Baseflow graag inzicht krijgen op de potentie van UWB technologie in mobiele apps op dit moment. Waarbij de mogelijkheden worden onderzocht voor een Flutter plug-in waarbij aangetoond wordt wat de functionaliteiten en mogelijkheden zijn van deze technologie aan de hand van een proof of concept.

1.4 Onderzoeksvragen

Nu de aanleiding, probleem definitie en doel bekend zijn kan de hoofdonderzoeksvraag worden vastgesteld. De hoofdonderzoeksvraag is als volgt gedefinieerd:

Hoe kan UWB technologie worden toegepast in mobiele apps, en specifiek in Flutter?

Om de hoofd-onderzoeksvraag te kunnen beantwoorden zijn de volgende subvragen gedefinieerd:

1. Wat is UWB en wat is de achterliggende technologie achter ultra wideband?
2. Wat zijn mogelijke toepassingen bij het inzetten van ultra wideband technologie op Android en Apple?
3. Op welke manier kan je een Flutter plug-in ontwikkelen voor UWB technologie en in hoeverre is het mogelijk een plug-in te schrijven waarbij de API voor zowel iOS als Android generiek is?
4. Welke applicatie zou als proof of concept de mogelijkheden en functionaliteiten van deze technologie goed kunnen weergeven en hoe ziet deze eruit?

1.5 Scope

Om een helder beeld te scheppen wat er wel en niet behandeld gaat worden gedurende dit onderzoek wordt hier de scope van dit onderzoek gegeven.

- Het onderzoek limiteert zich tot mogelijkheden in iOS, Android en Flutter. Hoewel UWB technologie al een stuk langer bestaat zal naast de inhoud en technische werking van UWB het onderzoek zich richten op de platformen Android, iOS en uiteindelijk Flutter. Dit betekent dat er geen onderzoek wordt gedaan naar de werking van UWB technologie op mogelijke andere platformen.
- Het proof of concept zal een vorm van plaatsbepaling bevatten gebaseerd op UWB technologie. Gezien UWB technologie uitermate geschikt lijkt te zijn voor onder andere plaatsbepaling is het dan ook vanzelfsprekend dat er een proof of concept zal worden opgeleverd dat gebruik maakt van deze technologie om dit te kunnen toepassen.
- Eventuele aanschaf of levering van benodigde producten zal gebeuren in overleg met Baseflow. Mocht de student bepaalde benodigdheden hebben met betrekking tot het aanschaffen of gebruik van dergelijke producten voor zijn of haar onderzoek dan zal dit ten allen tijde gebeuren in overleg.
- Gezien de ontwikkelingen van UWB technologie op het mobiele platform recent zijn kan het voorkomen dat er gedurende het onderzoek de student tegen barrières aan kan gaan lopen waarbij er bijvoorbeeld nieuwe ontwikkelingen nog niet direct zijn vrijgegeven. In dit geval zal de student zich gaan focussen op de mogelijkheden die er wel zijn. Gezien de opzet en scala aan mogelijkheden van en gedurende dit onderzoek zal dit naar verwachting geen problemen opleveren.

2 Onderzoek

In dit hoofdstuk wordt het onderzoek gedeelte van de opdracht vastgelegd. Hier worden de bouwstenen gelegd om met voldoende kennis een Flutter plugin, met bijbehorende applicatie, te kunnen ontwerpen en realiseren. Hiervoor wordt er onder andere onderzoek gedaan naar de werking van UWB, de beschikbare apparaten voor UWB, de verschillende frameworks die hierbij aan bod komen, het opzetten van een Flutter plugin en de verschillende mogelijke architecturen die hierbij komen kijken.

2.1 UWB

Om te achterhalen en begrijpen waarom UWB als nieuwe technologie op de mobiele markt wordt geïntroduceerd, en hiermee steeds meer in opkomst lijkt te zijn, is het goed om in ieder geval basis kennis op te doen wat deze technologie precies inhoudt, hoe deze technologie werkt en hoe deze zich onderscheidt ten opzichte van andere soortgelijke technologieën. Om te zien wat er wordt verstaan onder UWB wordt de volgende definitie gegeven:

> *"Ultrabreedband (ook bekend als UWB of Ultra Wideband) is een radio technologie die een zeer laag energieniveau kan gebruiken voor communicatie over korte afstand en hoge bandbreedte over een groot deel van het radiospectrum."[1]*

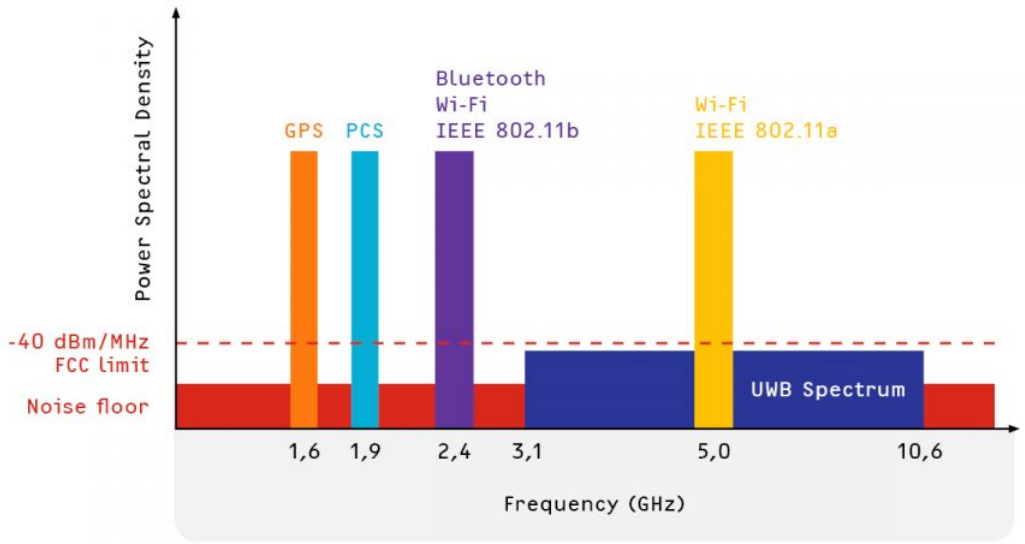
Zoals de bovenstaande definitie aangeeft valt UWB onder de zogenaamde radio technologieën. Andere bekendere radio technologieën zijn bijvoorbeeld WiFi en Bluetooth. Deze draadloze technologieën gebruiken radiosignalen om met andere apparaten die deze signalen kunnen ontvangen te communiceren. Toch kenmerkt UWB zich, naast deze 'bekendere' technieken, met een aantal opvallende eigenschappen die deze technologie een interessante keus maakt voor verschillende doeleinden. Dit is onder andere te zien in de toenemende populariteit onder het gebruik en beschikbaarheid van deze technologie onder mobiele apparaten. Meer hierover in kopje 2.1.2. Om wat dieper in te kunnen gaan op deze kenmerkende eigenschappen van UWB moet er worden gekeken naar de werking ervan.

2.1.1 Werking

Om in te kunnen gaan op de werking van UWB wordt er kort gekeken naar het ontstaan van UWB. Hier zijn namelijk destijds een aantal keuzes gemaakt omtrent het gebruik van UWB waar UWB op dit moment grotendeels zijn kenmerkende eigenschappen aan heeft te danken. UWB is in het leven geroepen in 2002 in Amerika. Hier heeft destijds de Amerikaanse Federal Communications Commission, of FCC in het kort, UWB toegestaan voor gebruik.[11] Echter wel onder een aantal voorwaarden. De FCC specificeerde dat het gebruik van UWB is toegestaan onder de frequenties 3.1 tot 10.6 GHz. Met als enige limiterende factor een limiet op de zogenaamde 'radiated power'. Of te wel, een limiet op de kracht van het radiosignaal dat mag worden uitgezonden. Een aantal jaar later komt ook Europa in 2005 met een protocol dat het gebruik van UWB toestaat. Hoewel dit protocol iets strengere voorwaarden bevat komt deze in grote lijnen overheen met de besluiten van de Amerikaanse FCC in 2002.[11] Hoewel UWB pas vele jaren later wordt gepopulariseerd, en UWB in het begin maar weinig gebruik heeft gezien, zijn met deze regels de basis gezet voor een nieuwe draadloze technologie.

Opvallend is de keuze om UWB toe te staan op de frequenties 3.1 tot 10.6 GHz. Om te visualiseren hoe dit er ter vergelijking uitziet ten opzichte van andere draadloze technologieën wordt er gekeken naar afbeelding 1. Dit betekent dat UWB, ten opzichte van andere draadloze technologieën, waaronder GPS, PCS, WiFi en Bluetooth, over een veel breder spectrum beschikt om een signaal op uit te kunnen zenden. Wat voor UWB resulteert in een bandbreedte van maar liefst 500MHz per kanaal, wat kan oplopen tot een bandbreedte van 7.5GHz. Om even in ter vergelijking te brengen hoeveel dit is, beschikt een Bluetooth kanaal over een bandbreedte van 1MHz en een WiFi kanaal respectievelijk over een bandbreedte van 22Mhz.[33] De grote van de bandbreedte is dan tevens ook waar UWB zijn naam 'Ultra Wideband' aan te danken heeft. Waarbij andere radio technologieën zoals Bluetooth, WiFi en GPS ook wel bekend staan als 'narrowband' technieken, gezien hier de bandbreedte een stuk lager ligt. Voor een visuele representatie hiervan, zie afbeelding 2.

Op afbeelding 1 is tevens ook de voorwaarde te zien die het FCC destijds heeft gesteld. Namelijk het limiet op de zogenaamde radiated power. Of, zoals in de figuur aangegeven, onder de naam 'Power Spectral Density'. Deze is namelijk zoals te zien gezet op 40dBm/Mhz. Dit heeft ermee te maken dat de UWB technologie andere 'narrowband' technieken niet in de weg gaat zitten. De



Figuur 1: Ultra Wideband frequentie

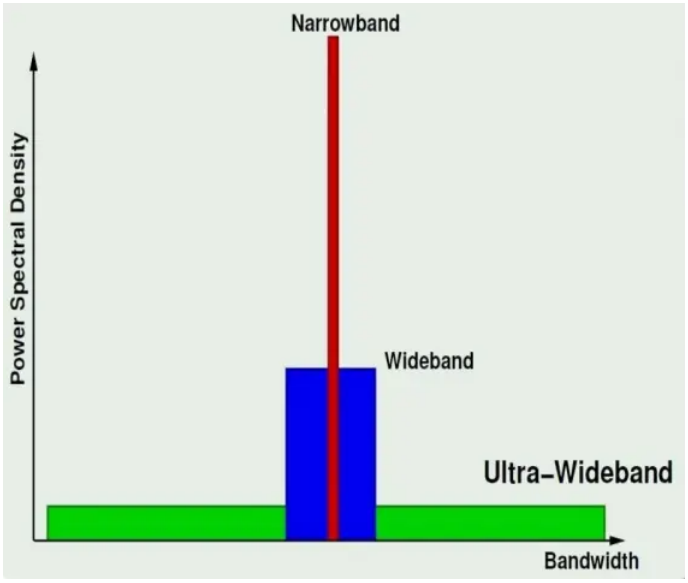
lage density zorgt er namelijk voor dat de apparaten die onder de narrowband technieken vallen, de UWB signalen, als ze deze al zien, niet meer zien dan alleen een beetje ruis. Dit komt omdat het limiet in dit geval dicht ligt bij de 'noice floor', waarbij de noice floor in dit geval verwijst naar het altijd aanwezige 'ruis' dat apparaten uitzenden. Wat tevens ook de reden is waarom UWB überhaupt mag uitzenden op zo 'n breed frequentie spectrum. Om aan te geven hoe stil UWB eigenlijk is wordt er verwezen naar de volgende quote:

> *"UWB operates at a much lower power than traditional radio uses. The power use is so low that it is arguably indistinguishable from ever-present "noise"."[30]*

Ook de techniek die UWB gebruikt om zijn radiosignalen uit te zenden verschilt dermate met zijn mede radio technologieën. Met de hoge bandbreedte (500MHz) waarover UWB beschikt kan UWb gebruik maken van pulseradiogolven. Een techniek waarbij UWB in een zeer korte tijd een gigantische hoeveelheid getimede pulses uitzend.[11] Hier wordt er gesproken over een hoeveelheid dat kan oplopen tot 2 gigapulses per seconde. Deze hoge frequentie aan pulsen maakt een apparaat dat UWB gebruikt zo goed als direct vindbaar voor andere UWB apparaten. Tevens betekend dit ook dat er door deze techniek, die alleen mogelijk wordt gemaakt door de hoge frequentie, en hiermee de hoge bandbreedte, UWB uitermate geschikt is voor bijvoorbeeld realtime tracking en verschillende andere doeleinden waarbij precisie een zeer grote rol speelt. Denk hier bijvoorbeeld aan:

- Secure Acces Control. Hier kan er bijvoorbeeld aan gedacht worden aan het gebruik van slimme (handsfree) toegang tot je auto via je telefoon.
- Secure Wireless Payments. Aan de hand van timestamps en het toepassen van willekeur in de frequentie van pulsen kan er encryptie worden toegepast op de berichten die verstuurd worden. Dit kan UWB ook een goed alternatief maken voor het verrichten van draadloze betalingen.
- Location-based Services. Denk hierbij bijvoorbeeld aan het tracken van verschillende objecten of het uitmeten van afstanden tussen kleine objecten. Maar ook bijvoorbeeld aan het tracken van spelers of atleten in sport of fitness.
- Healthcare. UWB is precies genoeg dat het ook zijn gebruik heeft gezien in de medische sector aan de hand van bijvoorbeeld medische radars of draadloze health sensoren.

Nu er een basiskennis ligt over en van UWB wordt er kort gekeken naar de beschikbaarheid van UWB in subhoofdstuk 2.1.2. Voor een totaaloverzicht van de kenmerken van UWB en de bijbehorende vergelijkingen ten opzichte van vergelijkbare andere draadloze technieken wordt er verwezen naar figuur 3.



Figuur 2: Ultra Wideband bandbreedte

TABLE I
COMPARISON OF THE BLUETOOTH, UWB, ZIGBEE, AND WI-FI PROTOCOLS

Standard	Bluetooth	UWB	ZigBee	Wi-Fi
IEEE spec.	802.15.1	802.15.3a *	802.15.4	802.11a/b/g
Frequency band	2.4 GHz	3.1-10.6 GHz	868/915 MHz; 2.4 GHz	2.4 GHz; 5 GHz
Max signal rate	1 Mb/s	110 Mb/s	250 Kb/s	54 Mb/s
Nominal range	10 m	10 m	10 - 100 m	100 m
Nominal TX power	0 - 10 dBm	-41.3 dBm/MHz	(-25) - 0 dBm	15 - 20 dBm
Number of RF channels	79	(1-15)	1/10; 16	14 (2.4 GHz)
Channel bandwidth	1 MHz	500 MHz - 7.5 GHz	0.3/0.6 MHz; 2 MHz	22 MHz
Modulation type	GFSK	BPSK, QPSK	BPSK (+ ASK), O-QPSK	BPSK, QPSK COFDM, CCK, M-QAM
Spreading	FHSS	DS-UWB, MB-OFDM	DSSS	DSSS, CCK, OFDM
Coexistence mechanism	Adaptive freq. hopping	Adaptive freq. hopping	Dynamic freq. selection	Dynamic freq. selection, transmit power control (802.11h)
Basic cell	Piconet	Piconet	Star	BSS
Extension of the basic cell	Scatternet	Peer-to-peer	Cluster tree, Mesh	ESS
Max number of cell nodes	8	8	> 65000	2007
Encryption	E0 stream cipher	AES block cipher (CTR, counter mode)	AES block cipher (CTR, counter mode)	RC4 stream cipher (WEP), AES block cipher
Authentication	Shared secret	CBC-MAC (CCM)	CBC-MAC (ext. of CCM)	WPA2 (802.11i)
Data protection	16-bit CRC	32-bit CRC	16-bit CRC	32-bit CRC

Figuur 3: Ultra Wideband vergelijking overzicht

2.1.2 Beschikbaarheid

Nu er wat kennis op tafel ligt over de werking van UWB en hoe deze aan zijn eigenschappen is gekomen, wordt er gekeken naar de huidige beschikbaarheid en gebruik van UWB. Deze is namelijk in de recente jaren een flink stuk toegenomen. Wel limiteert dit onderzoek zich tot de huidige beschikbaarheid van UWB op het mobiele platform. In dit geval te beginnen bij Apple.

Op gebied van UWB lijkt namelijk Apple het platform te zijn geweest dat het gebruik van UWB heeft geïntroduceerd naar het mobiele platform. Om precies te zijn heeft Apple UWB stilletjes naar het mobiele platform gehaald aan de hand van de introductie van de zogenaamde U1 chip in de iPhone 11 eind 2019.[15] De 'U' in U1 verwijst in dit geval naar Ultra wideband. En bevat dan ook de UWB technologie zoals beschreven is in subhoofdstuk 2.1.1. Inmiddels zijn we 2 generaties aan iPhones verder en heeft iPhone de chip wederom toegevoegd aan de nieuwere iPhones. Met ondersteuning in iOS 14 voor UWB beschikt Apple op dit moment over de volgende apparaten die gebruik kunnen maken van UWB aan de hand van Apples U1 chip [21]:

- iPhone 11 series
- iPhone 12 series
- iPhone 13 series
- Airtags
- Apple Watch series 6
- Apple Watch series 7
- HomePod mini

Niet alleen lijkt Apple er nu voor gekozen te hebben om de U1 chip standaard toe te voegen aan de hun nieuwste flagships, maar breid Apple tevens ook het gebruik van UWB uit door de onder andere de introductie van de Airtag, het toevoegen van UWB aan de Apple Watch series 6 en 7 en aan de HomePod mini. De U1 chip in deze apparaten stelt de gebruiker in staat om gebruik te maken van Apple's precision location tracking voor bijvoorbeeld het terug vinden van apparaten, maar ook voor het verbeteren van de Airdrop functie. Een functie waarmee Apple gebruikers bestanden met elkaar kunnen uitwisselen door middel van de telefoons op elkaar te richten.[22]

Dat het gebruik onder Apple apparaten met UWB technologie is toegenomen de afgelopen jaren viel ook op bij onder andere Samsung en Google. Niet veel later dan Apple introduceert namelijk ook Samsung UWB aan hun nieuwste Samsung Galaxy smartphones en Google bij hun Google Pixel 6.[30][31] Op dit moment bieden de volgende Android telefoons ondersteuning voor UWB:

- Samsung Galaxy Note 20
- Samsung Galaxy S20
- Samsung Galaxy S21
- Samsung Galaxy s22
- Samsung Galaxy Z Fold 2
- Samsung Galaxy Z Fold 3
- Samsung Galaxy SmartTag+
- Google Pixel 6

Nu er iets bekend is over welke apparaten op het mobiele platform ondersteuning bieden voor UWB kan er worden gekeken naar de ondersteuning die de verschillende platformen bieden voor development purposes. In dit geval te beginnen bij iOS.

2.2 iOS

Voor developers heeft Apple de API van de U1 chip vrij gegeven. Dit betekent dat er in dit geval ook third party apps ontwikkeld kunnen worden die gebruik kunnen maken van de nieuwe UWB technologie in de U1 chips. Hiervoor heeft Apple het Nearby Interaction Framework ontwikkeld. Tevens biedt Apple ook de mogelijkheid om applicaties, die gebruik maken van de UWB technologie, te testen in een iPhone emulator.

2.2.1 Nearby Interaction Framework

Het Nearby Interaction Framework stelt de gebruikers in staat om een verbinding op te zetten tussen twee of meer gebruikers die beschikken over een apparaat met een U1 chip.[24] Deze verbinding kan dan gebruikt worden om de locatie, ten opzichte van elkaar, uit te wisselen. Om aan zo 'n verbinding deel te kunnen nemen, kunnen apparaten, die binnen fysiek bereik zijn van elkaar, via een applicatie een token uitwisselen om vervolgens hun locatie te uitwisselen door middel van elkaars richting en afstand in meters door te geven.

Om een verbinding te kunnen opzetten die gebruik maakt van het Nearby Interaction Framework zijn, in ruwe lijnen, de volgende stappen vereist:

1. Het aanmaken van een Nearby Interaction Session object.

   Voordat een Nearby Interaction Session object kan worden aangemaakt en de volgende stappen kunnen worden ondernomen moet de class waarin deze sessie gaat worden aangemaakt eerst erven van NSObject en NISessionDelegate. Hierna kan er een Nearby Interaction Session object worden aangemaakt. Dit wordt als volgt gedaan:

   ```
   var session = NISession()
   ```

   Het Nearby Interaction Session object regelt alle communicatie tussen twee peers. Dat betekend dat wanneer een applicatie wenst meerdere verbindingen tegelijkertijd aan te gaan er voor elke verbinding tussen twee peers een aparte sessie moet worden aangemaakt.[17]

2. Het instellen van een Nearby Interaction Session delegate.

   Nu er een Nearby Interaction Session object is aangemaakt kan de class gekoppeld worden aan de Nearby Interaction Session delegate. Dit kan door de class toe te kennen aan de Nearby Interaction Session delegate. Dit gebeurt als volgt:

   ```
   session.delegate = self
   ```

   Nu de class gekoppeld is aan de delegate beschikt de class over een aantal delegate functies. Deze zullen, zodra er een sessie is gestart, de class voorzien van locatie gegevens van gekoppelde apparaten. De delegate functies zien er als volgt uit:

   ```
   func session(_ session: NISession, didUpdate nearbyObjects:
   [NINearbyObject]) {}

   func session(_ session: NISession, didRemove nearbyObjects:
   [NINearbyObject], reason: NINearbyObject.RemovalReason) {}

   func sessionWasSuspended(_ session: NISession) {}

   func sessionSuspensionEnded(_ session: NISession) {}

   func session(_ session: NISession, didInvalidateWith error: Error) {}
   ```

   De delegate functies zoals hierboven vermeld doen nu nog niks. Tevens wordt er nog niks gedaan met de data die binnen komt zodra er een sessie is gestart. Deze vind pas zijn invulling bij het voltooien van alle stappen in deze lijst.

3. Het aanmaken van een Nearby Interaction Discovery token.

   De delegate is inmiddels ingesteld. Nu moet er een discovery token worden aangemaakt zodat het mobiele apparaat, aan de hand van het verzenden van deze token, zich kenbaar

kan maken richting andere apparaten die beschikken over UWB. De token is voor elke sessie uniek en wordt gebruikt om het apparaat te kunnen identificeren die de sessie start.[14] Het aanmaken van zo 'n discovery token is als volgt:

```
var discoveryToken = session.discoveryToken
```

4. Het versturen van het Nearby Interaction Discovery token.

   Nu er een token is aangemaakt moet de token worden gedeeld met een apparaat dat in de buurt is. Hiervoor kan in principe elke 'netwerktechnologie' voor worden gebruikt waar beide apparaten met elkaar over instemmen. In dit geval raad Apple, voor het het uitwisselen van de tokens, het Multipeer Connectivity Framework aan.[16] Gezien dit framework een substantieel onderdeel is voor het opzetten van een Nearby Interaction sessie wordt ook deze, net zoals het Nearby Interaction framework, vastgelegd in dit verslag. Hoe dit framework precies werkt, en hoe deze gebruikt kan worden om de discovery token over te zenden naar een ander apparaat, is vastgelegd in hoofdstuk 2.2.2.

5. Het ontvangen van een Nearby Interaction Discovery token.

   Ook voor het ontvangen van een Nearby Interaction Discovery token wordt, zoals hierboven vermeld, verwezen naar hoofdstuk 2.2.2.

6. Het instellen van de Nearby Interaction Nearby Peer Configuration.

   Er vanuit gaande dat het gelukt is om met behulp van het Multipeer Connectivity Framework de discovery token over te dragen aan een peer, en de 'host' de token van zijn peer heeft ontvangen kan het Nearby Interaction Nearby Peer configuratie worden ingesteld. Dit wordt als volgt gedaan:

   ```
   let configuration = NINearbyPeerConfiguration(peerToken: token)
   ```

   De token, zoals in de code hierboven vermeld, is de ontvangen peer token.

7. En tot slot het starten van een sessie met de configuratie gegevens zoals vermeld in stap 6.

   Nu de configuratie ingesteld is, is het alleen nog een kwestie van het starten van de sessie aan de hand van de configuratie gegevens. Het starten van een sessie gebeurt als volgt:

   ```
   session.run(configuration)
   ```

Nu er een Nearby Interaction sessie is gestart gaat de Nearby Interaction Session Delegate zijn werk doen. Er is een datastream tussen twee peers opgezet waarbij er inmiddels locatie data binnenkomt. Deze data komt in dit geval binnen bij één van de delegate functies zoals vermeld in stap 2. De functie waarbij er locatie data binnen komt betreft de volgende:

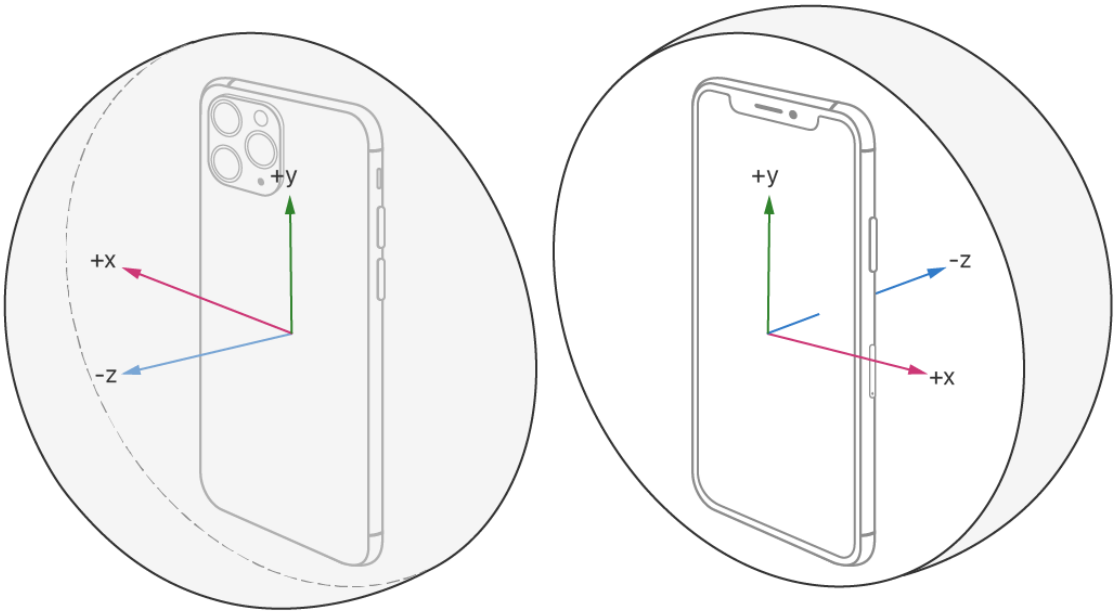
```
func session(_ session: NISession, didUpdate nearbyObjects: [NINearbyObject])
{}
```

De bovenstaande functie wordt door de delegate elke keer aangeroepen zodra er nieuwe locatie gegevens binnen komen. Locatie gegevens komen in dit geval binnen onder het nearbyObjects object. Dit object bevat zowel de afstand tussen beide peers als de richting tussen beide apparaten. Een voorbeeld van hoe de informatie uit een nearbyObjects eruit kan zien is als volgt:

```
[<NINearbyObject: Token: 52F4F35FE5C24C79B5563E1C32B9B088, Distance: 0.25m,
Direction: (0.475,-0.026,-0.880)>]
```

Hierboven, zoals aangegeven, is een voorbeeld te zien van de informatie die een nearbyObjects object kan bevatten. Te beginnen met de token van wie de data afkomstig is, gevolgd door de afstand tussen het ontvangen apparaat en de versturende partij. Tot slot zijn er drie waarden te

zien die de directie aangeven. Deze zijn, in volgorde, de 'x', 'y' en 'z' as. Waarbij respectievelijk de x de horizontale waarde bevat, de y de verticale en de z as de diepte tot het apparaat aangeeft.[18] Voor een visuele weergave hiervan zie figuur 4.



Figuur 4: Nearby Interaction - visuele weergave directie

2.2.2 Multipeer Connectivity Framework

Het Multipeer Connectivity Framework is een framework dat gebruikers in staat stelt om tussen twee peers een verbinding op te zetten om vervolgens via berichten of een datastream data met elkaar uit te wisselen. [16]

Om een Nearby Interaction sessie token te kunnen uitwisselen via het Multipeer Connectivity Framework moet er eerst een Multipeer Connectivity sessie worden gestart tussen twee peers. Voor het starten van een Multipeer Connectivity sessie, om vervolgens de Nearby Interaction token uit te kunnen wisselen, zijn de volgende stappen vereist:

1. Aanmaken van een Multipeer Connectivity ID.

   Voordat de Multipeer Connectivity ID kan worden aangemaakt, moet de class in dit geval 'extenden' van onder andere de MCNearbyServiceBrowserDelegate, MCNearbyServiceAdvertiserDelegate, MCSessionDelegate en de MCBrowserViewControllerDelegate. Deze zullen elk worden toegelicht en worden gebruikt in de volgende stappen. Nu deze zijn toegevoegd kan de Multipeer Connectivity ID worden aangemaakt. Het aanmaken van een Multipeer Connectivity ID is als volgt:

   ```
   let peerID = MCPeerID(displayName: UIDevice.current.name)
   ```

   In het voorbeeld hierboven wordt de naam van het toestel meegegeven. Maar in principe kan hier elke naam aan worden toegekend die de developer wil. De naam die hier wordt toegekend wordt later in de UI weergegeven in stap 5d.

2. Initialiseren van een Multipeer Connectivity Session.

   Nu er een peer id is aangemaakt kan er een instantie worden aangemaakt van een Multipeer Connectivity sessie. Dit gebeurt als volgt:

   ```
   var mcSession = MCSession(peer: peerID, securityIdentity: nil,
   encryptionPreference: .required)
   ```

3. Het toekennen van de class aan de Multipeer Connectivity Session delegate.

   Ook binnen het Multipeer Connectivity framework zijn er een aantal delegates die gekoppeld moeten worden aan de class waarin je het framework wil gebruiken. De sessie delegate zorgt er voor het ontvangen van data die verstuurd wordt via de sessie zoals die is aangemaakt in stap 2. Het koppelen van de Multipeer Connectivity Session delegate is als volgt:

   ```
   browser.delegate = self
   ```

   Deze delegate regelt alle communicatie tussen twee peers wanneer er een Multipeer Connectivity sessie is gestart. De session delegate functies zien er als volgt uit:

   ```
   func session(_ session: MCSession, peer peerID: MCPeerID, didChange
   state: MCSessionState) {}

   func session(_ session: MCSession, didReceive data: Data, fromPeer
   peerID: MCPeerID) {}

   func session(_ session: MCSession, didReceive stream: InputStream,
   withName streamName: String, fromPeer peerID: MCPeerID) {}

   func session(_ session: MCSession, didStartReceivingResourceWithName
   resourceName: String, fromPeer peerID: MCPeerID, with progress: Progress)
   {}

   func session(_ session: MCSession, didFinishReceivingResourceWithName
   resourceName: String, fromPeer peerID: MCPeerID, at localURL: URL?,
   withError error: Error?) {}
   ```

   Elke keer wanneer er een nieuwe sessie tot stand komt of wanneer er data binnenkomt via bericht of datastream wordt één van deze delegate functies aangeroepen. Vervolgens kan aan de hand hiervan verschillende nieuwe functies worden aangeroepen die de data dan weer verder afhandelt.

4. Het adverteren van het aangemaakte peer id over een net-werkprotocol.

   Multipeer Connectivity gebruikt onder water Bluetooth of WiFi om het peer id dat is aangemaakt te uitzenden naar andere apparaten om op die manier uiteindelijk een peer to peer sessie te kunnen opzetten. Het adverteren van het aangemaakte peer id kan binnen het Multipeer Connectivity Framework op een aantal verschillende manieren. In dit geval wordt er gebruik gemaakt van de Multipeer Connectivity Advertiser Assistant. Om de Multipeer Connectivity Advertiser Assistant te kunnen gebruiken zijn een aantal stappen vereist. Deze zijn als volgt:

   (a) Het initialiseren van de Multipeer Connectivity Advertiser Assistant.

   Als eerste stap moet de Multipeer Connectivity Advertiser Assistant worden geïnitialiseerd. Dit wordt gedaan met behulp van het peer id dat eerder in stap 1 is aangemaakt. Dit gebeurt als volgt:

   ```
   var mcAdvertiserAssistant = MCNearbyServiceAdvertiser (peer: peerID,
   discoveryInfo: nil, serviceType: "mpc-connect")
   ```

   Zoals in het voorbeeld hierboven te zien is wordt er ook een service type meegegeven. Wanneer een ander apparaat gaat zoeken naar peers die aan het adverteren zijn is het ook hier een vereiste om een service type mee te geven. Dit service type moet in dit geval met elkaar overeen komen om elkaar te kunnen vinden. Tevens zorgt dit ervoor dat men niet standaard alle advertisers ziet op het moment dat je zoekt naar een peer.

   (b) Het toekennen van de class aan de Multipeer Connectivity Advertiser Assistant delegate.

   Ook de Multipeer Connectivity Advertiser Assistant maakt gebruik van een delegate. Deze zal dan ook gekoppeld moeten worden aan de class waarin je deze gebruikt. Dit is als volgt:

```
mcAdvertiserAssistant.delegate = self
```

Deze delegate zorgt ervoor dat er connectie verzoeken kunnen worden ontvangen van peers die een verbinding willen aangaan met de advertiser. Deze delegate functie ziet er als volgt uit:

```
func advertiser(_ advertiser: MCNearbyServiceAdvertiser,
didReceiveInvitationFromPeer peerID: MCPeerID, withContext context:
Data?, invitationHandler: @escaping (Bool, MCSession?) -> Void) {
// accept invite
invitationHandler(true, mcSession)
}
```

Hierboven is een voorbeeld te zien van de de Multipeer Connectivity Advertiser Assistant delegate functie. In dit voorbeeld wordt er een connectie verzoek, zodra deze binnen komt, geaccepteerd.

(c) Het starten van de Multipeer Connectivity Advertiser Assistant.
Tot slot, moet de Multipeer Connectivity Advertiser Assistant nog worden gestart. Dit kan als volgt:

```
mcAdvertiserAssistant.startAdvertisingPeer()
```

Nu de advertiser is gestart kan er door een ander apparaat dat gebruikt maakt van het Multipeer Connectivity Framework zoeken naar de desbetreffende advertiser om uiteindelijk een verbinding op te starten. Hoe dit gebeurt is te lezen in stap 5.

5. Het 'browsen' of zoeken naar advertisers en het versturen van een connectie verzoek.

Het zoeken naar advertisers en en het versturen van een connectie verzoek naar een advertiser kan met behulp van de Multipeer Connectivity Browser View Controller. Hiervoor zijn de volgende stappen vereist:

(a) Het initialiseren van de Multipeer Connectivity Browser View Controller.
Het initialiseren van de Multipeer Connectivity Browser View Controller gebeurt als volgt:

```
var mcAdvertiserAssistant = MCNearbyServiceAdvertiser(peer: peerID,
discoveryInfo: nil, serviceType: "mpc-connect")
```

Ook hier wordt er, zoals aangegeven in stap mpc-advertiser initialisatie, het peer id van het apparaat meegeven, evenals het service type waarop er straks gezocht gaat worden.

(b) Het koppelen van de class aan de Multipeer Connectivity Browser View Controller delegate.
Ook de Multipeer Connectivity Browser View Controller gebruikt een delegate. Deze moet worden gekoppeld aan de desbetreffende class zodat deze later de benodigde functies aan kan roepen. Dit gebeurt als volgt:

```
mcAdvertiserAssistant.delegate = self
```

De Multipeer Connectivity Browser View Controller delegate functies worden onder andere aangeroepen wanneer de view controller wordt geannuleerd of wanneer er een verzoek komt om deze weer te sluiten. De Browser View Controller delegate functies zien er als volgt uit:

```
func browserViewControllerDidFinish(_ browserViewController:
MCBrowserViewController) {}

func browserViewControllerWasCancelled(_ browserViewController:
MCBrowserViewController) {}
```

Belangrijk is om de Multipeer Connectivity Browser View Controller weer te sluiten wanneer deze klaar is of wanneer deze wordt geannuleerd. Dit kan worden gedaan door de volgende regel aan te roepen wanneer de browserViewControllerDidFinish of browserViewControllerWasCancelled functie wordt aangeroepen door de delegate:

```
browserViewController.dismiss(animated: true, completion: nil)
```

(c) Het presenten van de browser aan de viewcontroller.
Nu de delegate is gekoppeld kan de Multipeer Connectivity Browser worden gepresenteerd aan de view controller. Hiermee krijgt de gebruiker een interface te zien met apparaten die, mits die er zijn, uitzenden op hetzelfde service type zoals uitgelegd in stap 4a en 5a. Het presenteren van de browser aan de view controller kan met behulp van een UIWindowScene.[20] Een voorbeeld hoe dit gebruikt kan worden om de browser aan de view controller te presenteren is als volgt:

```
let scenes = UIApplication.shared.connectedScenes
let windowScene = scenes.first as? UIWindowScene
let window = windowScene?.windows.first

window?.rootViewController?.present(browser, animated: true,
completion: nil)
```

(d) Via de UI kan vervolgens een connectie verzoek worden gestuurd aan de personen die aan het adverteren zijn op hetzelfde service-type.

6. Versturen van data via de Multipeer Connectivity Session.

Op het moment dat de Multipeer Connectivity Browser View Controller een connectie verzoek stuurt naar de advertiser gaat de advertiser delegate, zoals beschreven in stap 4b, in werking. Hier kan de connectie vervolgens worden geaccepteerd. Zodra deze wordt geaccepteerd gaat de session delegate, zoals beschreven is in stap 3, in werking. Deze kan dan vervolgens gebruikt worden om te kijken of er een 'connected state' is. Hoe er gecheckt kan worden wat de status is van een verbinding, en of er dan dus een connected state is, kan als volgt:

```
func session(_ session: MCSession, peer peerID: MCPeerID, didChange
state: MCSessionState) {
    switch state {
    case.connecting:
        print("\(peerID) state: connecting")
    case.connected:
        print("\(peerID) state: connected")
        //TODO: Send Nearby Interaction Discovery token
    case.notConnected:
        print("\(peerID) state: not connected")
    @unknown default:
        print("\(peerID) state: unkown")
    }
}
```

In dit geval wordt er een switch state gebruikt om te kijken wat de status is van een verbinding. Wanneer er een connected state is betekend dat er een actieve verbinding is tussen beide peers aan de hand van een Multipeer Connectivity sessie. Vervolgens kan er via de sessie een bericht worden verstuurd. Dit werkt als volgt:

```
mcSession.send(data, toPeers: mcSession.connectedPeers, with: .reliable)
```

De data in het voorbeeld hierboven verwijst in dit geval naar de data die de gebruiker wil versturen naar zijn peer.

7. Het ontvangen van data via een Multipeer Connectivity Session.

Wanneer er data wordt verstuurd in een actieve sessie, zoals beschreven is in stap 6, treed de session delegate, zoals beschreven in stap 3, voor het ontvangen van data in werking. Dit betreft de volgende session delegate functie:

```
func session(_ session: MCSession, didReceive data: Data, fromPeer
peerID: MCPeerID) {
print("Data received")
//TODO: start Nearby Interaction Session with received peer token
```

Dit zou voldoende moeten zijn om de Nearby Interaction discovery token, zoals beschreven is in stap 3 in het hoofdstuk 2.2.1, te verzenden en ontvangen. Vervolgens kan de token worden gebruikt om de configuratie in te stellen om de Nearby Interaction sessie te starten.

2.3 Android

Helaas heeft Android op dit moment nog niet de API vrij gegeven voor third party developers om gebruik te kunnen maken van de UWB chips die in de Android telefoons zitten. De verwachting was dat deze met de uitrol van Android 12 de API zou worden vrij gegeven. De API is echter, zoals Android heeft gespecificeerd, op dit moment vermeld als interne API. Dat betekent dat de API alleen voor gebruik is binnen de eigen organisatie, en hiermee dus (nog) niet beschikbaar is voor third party developers. Met de komst van Android 13 zijn er tot op heden nog geen (concrete) bevestigingen voor de uitrol van een publieke UWB API voor Android developers. Bij het gebrek aan een publieke API wordt er onderzoek gedaan naar de interne API van Android. Deze is namelijk in te zien in het 'Android Open Source Project'. Android open source is, zoals de naam al aangeeft, een open source project, die de open source code bevat waaruit Android is opgebouwd. Door te zoeken in het open source project van Android naar UWB is het wellicht mogelijk, hoewel documentatie ontbreekt, om iets te leren over het gebruik van UWB in Android.

2.4 Android Open Source project

In het open source project is het volgende over UWB gevonden: [3]. Hier wordt de huidige interne werking van UWB beschreven die gebruikt wordt binnen Android.

2.4.1 locatiebepaling

Te zien is hier dat ook Android zich focust op het uitwisselen van locatie gegevens. Dit is onder andere terug te zien aan de definitie van de UwbManager [8]:

> *This class provides a way to perform Ultra Wideband (UWB) operations such as querying the device's capabilities and determining the distance and angle between the local device and a remote device.*

Ook hier is er aan af te leiden, hoewel er ook aan de code afgekeken kan worden, dat de locatie gegevens die hiermee verzonden worden zich berusten op een afstand en richting zoals die bij Apple ook mee wordt verzonden. Als er wat dieper wordt ingedoken in de de DistanceMeasurement.java [5] en de AngleMeasurement.java [4] is ook hier te zien dat er Time of Flight (ToF) en Angle of Arrival (AoA) wordt gebruikt om de afstand en richting te bepalen.

2.4.2 Opzetten van een sessie

Android wisselt locatie gegevens uit door het opzetten van een 'ranging session'. Het starten van een sessie in Android gebeurt hier aan de hand van een openSession methode. Deze is terug te vinden in de RangingManager class. [6] Alvorens er een sessie gestart kan worden moet UWB op een Android apparaat worden aan gezet. Dit gebeurd aan de hand van de volgende methode zoals beschreven is in de UWBManager: [8]

```
/**
 * Disables or enables UWB for a user
 *
```

```
     * @param enabled value representing intent to disable or enable UWB.
    If true any subsequent
     * calls to IUwbAdapter#openRanging will be allowed. If false, all
    active ranging sessions will
     * be closed and subsequent calls to IUwbAdapter#openRanging will be
    disallowed.
     *
     * @hide
     */
    public void setUwbEnabled(boolean enabled) {
        mAdapterStateListener.setEnabled(enabled);
    }
```

Het opzetten van een ranging session gebeurd aan de hand van het uitwisselen van uwb adress zoals te zien is aan de hand van [7]. Een uwb adress bestaat in dit geval uit een bytearray. Hoe deze adressen precies worden uitgewisseld is lastig te achterhalen zonder bijbehorende documentatie. Wel zou aan de hand van bijvoorbeeld een bluetooth of wifi verbinding de adressen net zoals bij iOS de tokens worden uitgewisseld, uitgewisseld kunnen worden. Wel moet er nog steeds aan de hand hiervan er rekening mee moet worden gehouden dat de architectuur van de geboden oplossing op dusdanige wijze moet worden ontworpen dat, zover als dit mogelijk is, de Android kant eventueel later alsnog kan worden toegevoegd. Tevens zal de Android kant, in ieder geval voor opzetten van een Flutter plugin, wel worden meegenomen in dit onderzoek.

2.5 Android public API

Hoewel er geen publieke API vrijgegeven is zijn is Android wel bezig met een public variant van de UWB API. Dit is onder andere terug te zien aan deze commit op het Android open source project: [12]. Hoewel deze initiële commit nog geen code bevat wordt het er in de gradle file hiervan het vogende aangegeven:

```
29
30 androidx {
31     name = "androidx.core.uwb:uwb"
32     type = LibraryType.PUBLISHED_LIBRARY
33     mavenGroup = LibraryGroups.CORE_UWB
34     inceptionYear = "2022"
35     description = "Public API surface for apps to use UWB (ultra-wideband) on supported devices."
36 }
37
```

Figuur 5: Android initial commit public API UWB

Met de aanwijzing voor de komst van een public API moeten de bevindingen van de interne API wel met een korrel zout worden genomen gezien deze nog open staan voor verandering. Wel is goed om te zien dat er een sessie benodigt is het uitwisselen van locatie gegevens en dat de locatie gegevens in ieder geval een afstand en richting bevatten.

2.6 Flutter plugins en packages

Om de UWB techniek naar het Flutter platform toe te halen is er een manier nodig om vanuit Flutter te kunnen communiceren met het native platform dat gebruik maakt van de UWB technologie. Een manier om dit te doen is het ontwerpen en realiseren van een Flutter plugin. Maar voordat deze Flutter plugin wordt ontworpen en gerealiseerd is eerst basiskennis benodigd over Flutter plugins en packages. Te beginnen over een introductie over de verschillende packages die Flutter aanbiedt.

2.6.1 Introductie packages

Flutter beschikt over een tweetal soorten packages. Namelijk de 'Dart package' en de 'Plugin package'.[28] Om te begrijpen wat hier een 'package' precies inhoudt wordt er gekeken naar de volgende quote:

> *"Packages can be explained as shared packages contributed by other developers to the Flutter and Dart ecosystems. This allows developers to quickly build an app without having to develop everything from scratch."[32]*

Een package is in dit geval dus niks anders dan een stuk code dat toegevoegd is aan het Flutter ecosysteem voor gebruik voor andere developers. Een 'Dart package' houdt in dit geval een package in die puur en alleen geschreven is in de Flutter taal Dart. Een Flutter plugin package, is een package dat in dit geval de connectie legt tussen de native platformen en het Flutter platform. Voor een algemeen begrip over wat hier precies onder wordt verstaan wordt er gerefereerd naar de volgende quote:

> *"A plugin package is a special kind of package that makes platform functionality available to the app. Plugin packages can be written for Android (using Kotlin or Java), iOS (using Swift or Objective-C), web, macOS, Windows, Linux, or any combination thereof. For example, a plugin might provide Flutter apps with the ability to use a device's camera."[26]*

Met behulp van Flutter plugin package, of Flutter plugin in het kort, zou in theorie elke 'native' technologie naar het Flutter platform toegehaald kunnen worden gezien deze werkt als een soort van 'brug' tussen de Dart taal in Flutter en native talen zoals Kotlin of Swift voor Android en iOS. Zoals te lezen is, is een Flutter plugin eigenlijk ook gewoon een package. Het verschil zit het hem hier in dat een Flutter plugin wel meerdere packages kan bevatten. Denk hierbij aan bijvoorbeeld een Flutter plugin die iets schrijft voor de platformen Windows en Linux. In dit geval zal de Flutter plugin een Windows package bevatten, die een directory omvat met alle geschreven code voor het Windows platform, en een Linux package, die daarin tegen een directory omvat met alle geschreven code voor het Linux platform. Dit laatste staat, in basis, ook wel bekend als een 'Federated plugin', waarbij elk platform in een Flutter plugin beschikt over zijn eigen package.[28]. Voor een gedetailleerde beschrijving van een federated plugin wordt verwezen naar subhoofdstuk 2.6.2.

Gezien er een connectie moet worden gelegd met de platformen Android en iOS wordt er verder niet ingegaan op het maken van een 'Dart package'. De focus wordt in dit geval gelegd op het creëren van een Flutter plugin. Mocht U als lezer toch geïnteresseerd zijn in het maken van Dart package wordt er verwezen naar de volgende link [28].

2.6.2 Federated plugins

De huidige standaard voor het creëren van een Flutter plugin is door middel van een 'Federated plugin'.[28] Een federated plugin is een plugin waarbij er voor elke platform implementatie een aparte package wordt geschreven. Deze bevatten elk ieder zijn eigen code die gebaseerd op het communiceren voor één specifiek platform. Een federated plugin bestaat in ieder geval uit de volgende packages:

- App-facing package.

  De app-facing package is de package waar de gebruikers van de plugin op vertrouwen. Deze bevat een beschrijving van de API die gebruikt kan worden door een Flutter app.

- Platform package(s).

  De platform package bevat, zoals voorheen beschreven, de platform specifieke implementatie. De calls, gedefinieerd in de API van de app-facing package, gaan in dit geval naar de platform specifieke packages. Deze handelen vervolgens de API calls af.

- Platform interface package.

  Als laatste is er de platform interface package. Deze bind in dit geval de app-facing package en de platform specifieke packages samen. De platform interface package declareert in dit geval een interface, met API calls, die de platform specifieke packages moeten implementeren. Dit zorgt ervoor dat de platform specifieke packages een standaard aantal generieke API calls moet kunnen afhandelen die gelijk zijn voor alle platformen.

Federated plugins geven developers ook de mogelijkheid om later een nieuwe platform specifieke package toe te voegen aan de plugin. Het mooie hieraan is, is dat dit niet perse hoeft te gebeuren door de originele auteur van de plugin. Stel dat een developer een platform specifieke implementatie bouwt voor een bepaalde plugin dan kan de originele auteur van de plugin deze implementatie toevoegen door deze toe te voegen als dependency in de pubspec file.[28] Wanneer deze implementatie door de auteur wordt toegevoegd dan wordt dit ook wel een endorsed federated plugin genoemd. De auteur 'endorsed' als het ware de implementatie van zijn mede developer. Dit

houdt ook in dat wanneer een ontwikkelaar gebruik maakt van de plugin dat ook de endorsed implementaties worden meegenomen en beschikbaar worden gesteld ter gebruik in de app.

Mocht het zo zijn dat een implementatie, om welke reden dan ook, niet wordt 'geëndorsed' door de originele auteur dan bestaat er nog de mogelijkheid om deze nog manueel toe te voegen als developer in de pubspec file van de app.

2.6.3 Het opzetten van een package

Voor het opzetten van een Flutter package wordt de command line gebruikt. In dit geval kan er een plugin package worden gemaakt door '–template=plugin' aan te roepen met 'flutter create', Vervolgens wordt het platform meegegeven waarvoor de package wordt gemaakt. Dit gebeurt aan de hand van '–platforms=' gevolgd door het gewenste platform. Platformen die op dit moment worden ondersteund door flutter voor het maken van een flutter plugin package zijn Android, iOS, web, Linux, macOS en Windows.[28] Wanneer er geen platform wordt gedefinieerd ondersteund de package dus geen enkel platform. Ter voorbeeld voor het creëren van een Android package kan het volgende commando in de terminal worden gebruik:

```
$ flutter create --template=plugin --platforms=android -a kotlin hello
```

In het bovenstaande voorbeeld wordt er een package aangemaakt voor het Android platform. Tevens is de taal die gebruikt gaat worden gespecificeerd. Namelijk, Kotlin. De package wordt in dit geval aangemaakt onder de folder met de naam 'hello'. Om te begrijpen wat er nu precies is aangemaakt wordt er kort gekeken naar de inhoudt van de 'hello' mappen. Hierin komt onder andere het volgende terug:

- lib/hello.dart

  De lib/hello.dart file bevat in dit geval de API voor de plugin.

- android/src/main/java/com/example/hello/HelloPlugin.kt

  Deze file bevat de platform specifieke implementatie van de API voor de plugin. In dit geval is het een Kotlin file. Het adres voor een ander platform ziet er uiteraard anders uit.

- example/lib/main.dart

  Deze file bevat een voorbeeld Flutter applicatie die weergeeft hoe je de plugin kan gebruiken.

Voordat een package in gebruik kan worden genomen is het handig om te weten waar de platform specifieke code zich precies bevind voor in dit geval Android en iOS. Tevens zijn er een aantal vereiste stappen benodigd voor het opzetten van een package omgeving. Deze zijn per platform als volgt:

- Android.

  Voor het openen en bewerken van native code adviseert Google om Android studio te gebruiken.[28] Vervolgens zijn de volgende stappen vereist.

  1. Open Android studio
  2. Select File, open from menu, en selecteer de hello/example/android/build.gradle file.
  3. In de Gradle Sync dialoog, selecteer 'OK'.
  4. In de Android Gradle Plugin Update dialoog, selecteer 'Don't remind me again for this project'.

  Vervolgens kan de platform specifieke code worden geopend en bewerkt in hello/java/com.example.hello/HelloPlugin. De code kan nu worden gestart door op het start icoontje te klikken.

- iOS.

  Voor het openen en bewerken van iOS code adviseert Google Xcode.[28] Vervolgens moeten eerst de volgende stappen worden ondernomen:

  1. Voordat de native files worden geopend om te bewerken adviseert Google eerst de package een keer te runnen. Dit kan aan de hand van het command:

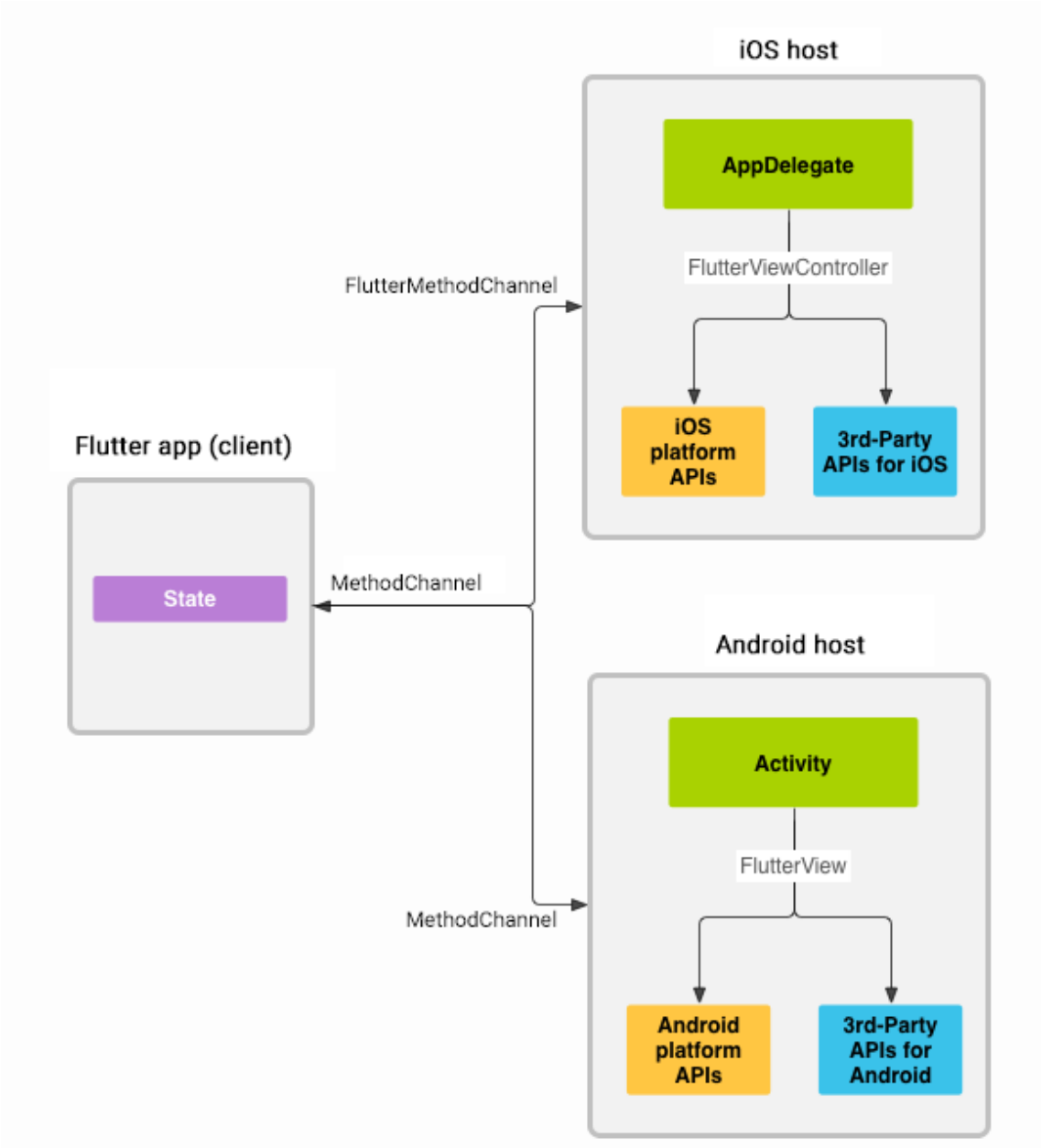
```
cd hello/example; flutter build ios --no-codesign
```

2. Vervolgens kan Xcode worden geopend.
3. Select File, open, en selecteer de hello/example/ios/Runner.xcworkspace file.

Nu is de platform specifieke code te openen via Pods/hello/../../example/ios/.symlinks/plugins/hello/ios/Classes in de Project Navigator. De code kan nu worden gestart door op het start icoontje te klikken.

2.7 Methodchannels

Nu er basis kennis is over het opzetten van een Flutter plugin met de platform specifieke packages moet er nog een manier zijn om de API te koppelen aan de native code. Dit gebeurt in dit geval aan de hand van methodchannels.[29] Een methodchannel is in dit geval de brug die Flutter in staat stelt om berichten naar een native platform zoals iOS of Android te sturen. Zodat deze vervolgens gebruik kan maken van de native platform API's of third-party API's van dit specifieke platform. Voor een visuele weergave van de werking van een methodchannel zie afbeelding 6.



Figuur 6: Methodchannel

Goed is om te weten dat in het algemeen Flutter via een methodchannel een bericht stuurt naar een platform specific package, en deze package via een response een bericht terug stuurt. Voor het opzetten van een methodchannel wordt er in dit geval gekeken naar de implementatie voor de platformen iOS en Android. Om berichten te kunnen versturen vanuit Dart naar Android of iOS is het één en ander aan set-up vereist. De set-up die benodigd is voor het versturen van berichten van Flutter naar het Android en iOS platform wordt beschreven in de subhoofdstukken 2.7.1, 2.7.2 en 2.7.3. In dit geval wordt er begonnen bij Dart.

2.7.1 Dart

Voor het versturen van berichten in Dart moeten de volgende stappen worden ondernomen:

1. Initialiseren van de methodchannel instantie.

   Het initialiseren van de methodchannel instantie in Dart is als volgt:

   ```
   static const _channel =
   MethodChannel('com.baseflow.uwb/example_channel');
   ```

   In dit geval is de String die word meegegeven aan de methodchannel de identifier. Dat wil zeggen dat aan de platform specific packages aan de native kant de methodchannel moet worden geïnitialiseerd met de zelfde String identifier om elkaar terug te kunnen vinden.

2. Verzenden van berichten.

   Om een bericht te kunnen versturen kan er via de aangemaakte methodchannel instantie een bericht worden verstuurd op de volgende manier:

   ```
   Future<String?> get platformVersion async {
       final String? version = await
   _channel.invokeMethod('getPlatformVersion');
       return version;
   }
   ```

   In dit voorbeeld bericht wordt er een verzoek verstuurd om de platformversie van het apparaat te achterhalen. Dit verzoek verwacht een String terug. Deze staat genoteerd als 'Future', dat wil zeggen dat er een promise wordt gedaan dat er ergens ooit een String wordt terug gegeven. De String die in dit geval aan de invokemethod wordt meegegeven ('getPlaformVersion') is in dit geval de identifier waarop straks aan de aan de native kant het bericht kan worden afgevangen. Tot slot gebeurt deze call async, zodat de main thread niet hoeft te wachten tot het een antwoord terug heeft van de platform specific package.

2.7.2 Android

Om te kunnen reageren op de berichten die aan de Dart kant worden verzonden, zoals aangegeven in stap 2 van subhoofdstuk 2.7.1 moet aan de Android kant het volgende gebeuren:

1. Initialiseren van de methodchannel.

   Voordat de methodchannel kan worden geïnitialiseerd moet eerst de desbetreffende class die deze implementatie wil gebruiken erven van FlutterPlugin en de MethodCallHandler. De FlutterPlugin extentie zorgt er in dit geval voor dat de methodcall handler weer kan worden stop gezet wanneer de 'verbinding' met de Dart kant wordt verbroken. Nu de class erft van FlutterPlugin en de MethodCallHandler kan de methodchannel worden geïnitialiseerd. Het initialiseren van de methodchannel gebeurt als volgt:

   ```
   private var channel =
   MethodChannel(flutterPluginBinding.binaryMessenger,
   "com.baseflow.uwb/example_channel")
   ```

   Let er hierbij wel op dat de String identifier hetzelfde is als bij de methodchannel zoals die aangegeven is bij stap 1 in subhoofdstuk 2.7.1.

2. Het setten van de MethodCallHandler door de class er aan toe te kennen.

```
channel.setMethodCallHandler(this)
```

3. Het ontvangen van berichten.

   Nu de MethodCallHandler ingesteld is en gekoppeld is aan de class wordt er elke keer als er een bericht binnenkomt, met dezelfde String identifiër, een onMethodCall functie aangeroepen. Deze ziet er als volgt uit:

```
override fun onMethodCall(@NonNull call: MethodCall, @NonNull
result: Result) {
    when (call.method) {
        "getPlatformVersion" -> {
          result.success("Android
${android.os.Build.VERSION.RELEASE}")
        }
        else -> {
          result.notImplemented()
        }
    }
}
```

   In het voorbeeld hierboven wordt er gekeken of de method call als identifiër 'getPlatformVersion' bevat. Wanneer dit het geval is wordt er een result.succes terug gestuurd met een de os versie van de Android telefoon. Wanneer er een call binnenkomt die niet voldoet aan de identifiërs wordt er een result.notImplemented terug gestuurd.

2.7.3 iOS

1. Aanmaken van een methodchannel instantie in iOS.

   Alvorens een methodchannel instantie kan worden aangemaakt moet eerst die class erven van FlutterPlugin. Vervolgens kan in iOS kan een methodchannel instantie op de volgende manier worden aangemaakt:

```
static var channel = FlutterMethodChannel(name:
    "com.baseflow.uwb/example_channel", binaryMessenger:
    registrar.messenger())
```

   Ook hier geld dat er hierbij op gelet moet worden dat de String identifier hetzelfde is als bij de methodchannel zoals die aangegeven is bij stap 1 in subhoofdstuk 2.7.1.

2. Het instellen van de Method Call Handler in iOS.

   Het instellen van de Method Call Handler in iOS is als volgt:

```
registrar.addMethodCallDelegate(instance, channel: channel)
```

   Waarbij de instance in dit geval verwijst naar een instantie van de huidige class. De registrar methode komt mee met een methode die automatisch wordt aangemaakt bij het erven van de FlutterPlugin in de class.

3. Het ontvangen van berichten.

   Nu de addMethodCallDelegate is toegevoegd aan de class kunnen er berichten worden ontvangen. Wanneer er berichten binnenkomen met een string identifiër die overeen komt met de String identifiër van de methodcall in Dart wordt de volgende functie aangeroepen:

```
public func handle(_ call: FlutterMethodCall, result: @escaping
    FlutterResult)
    {
        switch call.method {
        case "getPlatformVersion":
            result("iOS " + UIDevice.current.systemVersion)
        default:
            result("Methodcall doesn't exist")
        }
    }
```

In het voorbeeld hierboven worden de berichten afgevangen door een switch case. Hoewel er hier maar één voorbeeld in staat zullen dit er in praktijk altijd een stuk meer zijn. In dit geval wordt er bij een identifiër van 'getPlatformVersion' een bericht terug gestuurd met de OS versie van het huidige apparaat. Wanneer er een bericht binnenkomt dat niet hiermee overeenkomt wordt er een string teruggestuurd met "Methodcall doesn't exist".

Deze basiskennis over methodchannels zou voldoende moeten zijn om te kunnen beginnen met het opzetten van plugin die onder andere via een methodchannel berichten kan versturen naar de native platform packages. Tevens bestaat er ook zoiets als 'reverse methodchannels' waarbij er berichten verstuurd kunnen worden vanuit de native platform package richting Dart. Hoewel een 'normale' methodchannel gebruikt kan worden om gegevens op te halen van een native platform package aan de hand van een response bericht kan het soms handig zijn om een reverse methodchannel te gebruiken. Denk hierbij bijvoorbeeld aan een listener aan de native kant, waarbij er zodra er data binnenkomt, deze data graag doorgestuurd wil hebben naar de Dart kant. Het opzetten hiervan is omgekeerd. De berichten worden nu verstuurd aan de native kant en afgevangen in Dart. Een voorbeeld van de Dart kant om berichten af te vangen ziet er dan als volgt uit:

```
  static const MethodChannel _locationChannel =
    MethodChannel('com.baseflow.uwb/ni_session_location');

   _locationChannel.setMethodCallHandler(_handleMethodCall);

  Future<void> _handleMethodCall(MethodCall call) async {
  switch (call.method) {
    case 'updateLocation':
      print(call.arguments)
    default:
      throw MissingPluginException();
  }
}
```

3 Proof of Concept iOS

Dit hoofdstuk focust zich op het ontwikkelen en realiseren van een proof of concept voor het gebruik van UWB in iOS. Hierbij wordt er gebruik gemaakt van de verschillende frameworks zoals ze beschreven zijn in hoofdstuk 2.2. Ook wordt de architectuur gegeven aan de hand van een class diagram en wordt er in gegaan op verschillende belangrijke en/of complexe code en wordt deze voorzien van het benodigde commentaar.

3.1 Opzet

Voor het opdoen van kennis met betrekking tot development in iOS en de bijbehorende frameworks op gebied van UWB wordt er zoals hiervoor benoemd een proof of concept ontwikkeld. Voor dit proof of concept wordt de programmeertaal Swift gebruikt en wordt er voor de UI SwiftUI gebruikt. Hierbij is er voor Swift gekozen omdat tussen objective C en Swift, Swift als meest toegankelijke taal wordt gezien voor developers die nog geen ervaring hebben in iOS development. Tevens wordt Swift inmiddels ook ondersteund als native taal voor het ontwikkelen van Flutter plugins. SwiftUI wordt gebruikt gezien dit in lijn is met de huidige standaard. Voor het ontwikkelen van de benodigde kennis over de desbetreffende frameworks worden de volgende voorwaarden gesteld aan het proof of concept:

1. De applicatie moet in staat zijn een Nearby Interaction sessie op te kunnen zetten tussen twee apparaten die beschikken over een U1 chip.
2. De applicatie moet in staat zijn om een Multipeer Connectivity sessie op te kunnen zetten om berichten te kunnen uitwisselen.
3. De applicatie moet de gebruiker kunnen informeren met welke apparaten/gebruikers hij of zij een verbinding kan aangaan.
4. De applicatie moet de locatie gegevens weer kunnen geven aan de gebruiker wanneer een verbinding tot stand is gekomen.

Na het aanmaken van een nieuw Swift project in Xcode voor iOS ziet de mappenstructuur er als volgt uit:

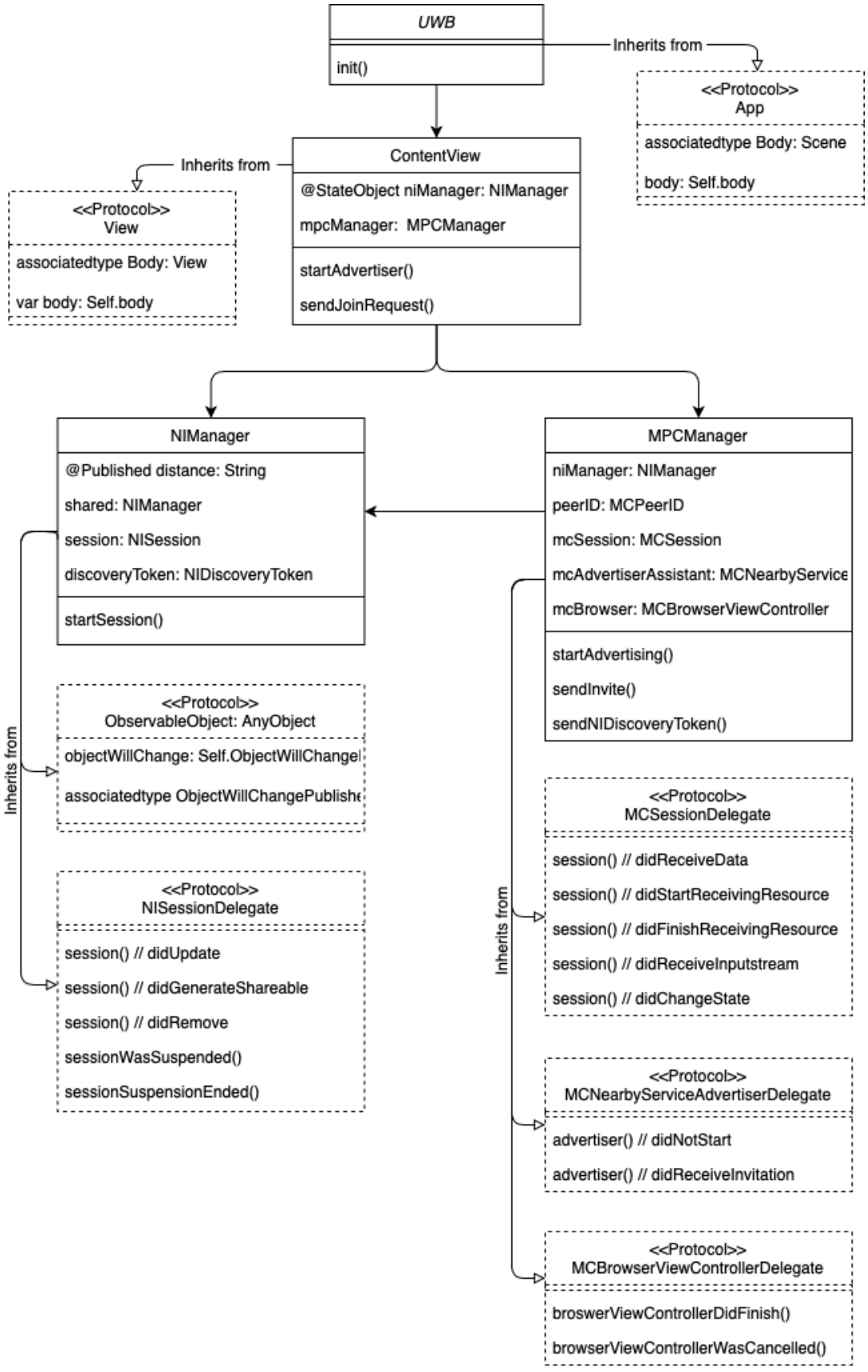
```
UWB
└── UWB
    ├── UWB.swift
    ├── ContentView.swift
    └── Assets.xcassets
```

De UWB.swift roept in dit geval de Contentview.swift aan en presenteerd deze aan de gebruiker. Waarbij de Contentview de UI bevat. Assets.xcassets houdt zich normaliter bezig met alle assets van app. Deze kan echter voor dit proof of concept worden genegeerd gezien deze geen assets zal bevatten. Aan deze mappenstructuur worden de volgende twee Swift files aan toegevoegd. Namelijk NIManager.swift en MPCManager.swift. De NIManager zal zich gaan buigen over het opzetten van de Nearby Interaction Framework, waarbij de MPCManager benodigd is voor het opzetten van peer to peer verbinding om de nodige tokens uit te kunnen wisselen.[23] Meer hierover in hoofdstuk 2.2.2. Dit geeft de volgende mappenstructuur:

```
UWB
└── UWB
    ├── UWB.swift
    ├── ContentView.swift
    ├── NIManager.swift
    ├── MPCManager.swift
    └── Assets.xcassets
```

Nu de er een initiële mappenstructuur is opgezet kan de benodigde code worden toegevoegd. Voor een inzage op de belangrijke en complexe methodes met bijbehorende toelichting zie 3.3. Voor een totaal overzicht van het proof of concept met alle classes, bijbehorende methodes en extensies zie figuur 7.

3.2 Class Diagram



Figuur 7: Class diagram - Proof of Concept

3.3 Toelichting op code

In dit hoofdstuk wordt er gekeken naar alle belangrijke en complexe methoden en functies van het proof of concept. Deze zullen in dit hoofdstuk worden toegelicht. Om het overzicht te bewaren wordt deze toegelicht per class. Te beginnen bij UWB:

3.3.1 UWB

De UWB class is de ingang van de app. Deze roept in dit geval de ContentView aan om deze te presenteren aan de gebruiker. Echter voordat de app wordt gepresenteerd aan de gebruiker wordt er een check gedaan om te kijken of het apparaat wel geschikt is voor UWB. Dit gebeurt aan de hand van de volgende code in de 'init' methode:

```
init() {
      guard NISession.isSupported else {
          print("This device doesn't support Nearby Interaction.")
          return
      }
  }
```

Wanneer het apparaat in dit geval 'NISession' support is het geschikt voor UWB en kan hiermee de ContentView aan de gebruiker worden gepresenteerd.

3.3.2 ContentView

De ContentView bevat twee variabelen. Dit zijn de volgende:

```
@StateObject var niManager: NIManager = NIManager.shared
var mpcManager: MPCManager = MPCManager()
```

In dit geval bevat ContentView een instantie van MPCManager en een gedeelde instantie van NIManager. Deze gedeelde instantie wordt later ook in de MPCManager class gebruikt. De niManager instantie staat genoteerd als een stateobject. Dat betekend dat de niManager instantie een variabele kan bevatten waar vanuit deze class een listener op is gezet. Zodra deze variabele een nieuwe waarde krijgt dan wordt de view opnieuw geladen, mits deze variable gebruikt wordt in de view. De variabele die hiervoor wordt gebruikt is 'distance'. Meer hierover in 3.3.4.

De view is als volgt opgebouwd:

```
var body: some View {
    VStack(spacing: 50){
        Button {
            mpcManager.startAdvertising()
        } label: {
            Text("HOST")
        }
        Button {
            mpcManager.sendInvite()
        } label: {
            Text("JOIN")
        }
        Text("Distance: ")
        Text("\(niManager.distance)m").font(.system(size: 40))
    }
    .padding()
}
```

De view bevat twee buttons. De 'HOST' button en de 'JOIN' button. Deze roepen respectievelijk de advertiser en invite methode aan van de mpcManager. De afstand wordt in dit geval weergegeven door de niManager.distance. Zoals hiervoor benoemd staat hier een listener op en wordt deze waarde gewijzigd bij een wijziging ervan in de niManager instantie. Tot slot bevat de body nog wat spacing aan de hand van een VStack.

3.3.3 MPCManager

De MPCManager erft van de volgende protocollen:

```
class MPCManager: NSObject, MCNearbyServiceAdvertiserDelegate,
MCSessionDelegate, MCBrowserViewControllerDelegate
```

De MPCManager bevat de volgende variabelen en kent ze op de volgende manier toe:

```
var peerID: MCPeerID
var niManager: NIManager
var mcSession: MCSession
var mcAdvertiserAssistant: MCNearbyServiceAdvertiser?
var serviceType: String?

override init(){
    peerID = MCPeerID(displayName: UIDevice.current.name)
    mcSession = MCSession(peer: peerID, securityIdentity: nil,
encryptionPreference: .required)
    niManager = NIManager.shared
    serviceType = "mpc-connect"

    super.init()
    mcSession.delegate = self
}
```

Belangrijke variabelen die hier worden aangemaakt zijn onder andere het peerID en de serviceType. Het peerID wordt straks omgezet naar de 'displayName', deze wordt weergegeven als gebruikers zoeken met de BrowserViewControllerDelegate. En het servicetype is de string identifiër waarop gezocht wordt. Tot slot wordt de mcAdvertiserAssistant later geïnitialiseerd op het moment dat de advertiser methode wordt gestart. De advertiser methode ziet er als volgt uit:

```
func startAdvertising() {
    mcAdvertiserAssistant = MCNearbyServiceAdvertiser(peer: peerID,
discoveryInfo: nil, serviceType: serviceType)
    mcAdvertiserAssistant?.delegate = self
    mcAdvertiserAssistant?.startAdvertisingPeer()
}
```

Hier wordt de advertiser assistant geïnitialiseerd met onder andere het peerID en de voorheen vermelde serviceType. Vervolgens wordt de delegate toegekend aan de class en tot slot wordt de advertiser gestart. Na het aanroepen van deze methode kan er naar dit specifieke apparaat worden gezocht met een browser delegate. Voor het starten van de MCBrowserViewControllerDelegate wordt de volgende methode gebruikt:

```
func sendInvite() {
    let browser = MCBrowserViewController(serviceType: serviceType,
session: mcSession)
    browser.delegate = self

    //present the browser to the viewcontroller
    let scenes = UIApplication.shared.connectedScenes
    let windowScene = scenes.first as? UIWindowScene
    let window = windowScene?.windows.first

    window?.rootViewController?.present(browser, animated: true,
completion: nil)
}
```

In deze methode wordt er een browser instantie aangemaakt met de voorheen aangegeven peerID en serviceType. De delegate wordt vervolgens toegewezen aan de MPCManager class. Tot slot word de browser aan de viewcontroller gepresenteerd. Voor het annuleren en sluiten van de browser worden de volgende delegate functies gebruikt:

```
func browserViewControllerDidFinish(_ browserViewController:
MCBrowserViewController) {
    //dismiss browser when done
    browserViewController.dismiss(animated: true, completion: nil)
}

func browserViewControllerWasCancelled(_ browserViewController:
MCBrowserViewController) {
    //dismiss browser when cancelled
    browserViewController.dismiss(animated: true, completion: nil)
}
```

Wanneer er een advertiser is gekozen aan de hand van de browserViewController om een verbinding mee te starten, wordt er vervolgens een connectieverzoek verstuurd. Deze komt binnen bij de volgende delegate en kan hop de volgende manier worden goed gekeurd:

```
func advertiser(_ advertiser: MCNearbyServiceAdvertiser,
didReceiveInvitationFromPeer peerID: MCPeerID, withContext context:
Data?, invitationHandler: @escaping (Bool, MCSession?) -> Void) {
    // accept invite
    invitationHandler(true, mcSession)
}
```

Nu het verzoek is goedgekeurd kan er aan de hand van de volgende delegate functie de state worden gecheckt:

```
func session(_ session: MCSession, peer peerID: MCPeerID, didChange
state: MCSessionState) {
    switch state {
    case.connecting:
        print("\(peerID) state: connecting")
    case.connected:
        print("\(peerID) state: connected")
        if niManager.session == nil {
            niManager.start()
            sendNIDiscoveryToken()
        }
    case.notConnected:
        print("\(peerID) state: not connected")
    @unknown default:
        print("\(peerID) state: unkown")
    }
}
```

Ook wordt er aan de hand van de bovenstaande delegate functie een aantal variabelen geset in de niManager instantie en er een functie aangeroepen die de Nearby Interaction discovery token verstuurd. Deze functie ziet er als volgt uit:

```
func sendNIDiscoveryToken(){
    print("Trying to send discoverytoken")
    if mcSession.connectedPeers.count > 0 {

        guard let dataToken = niManager.discoveryToken,
```

```
                let data = try?
NSKeyedArchiver.archivedData(withRootObject: dataToken,
requiringSecureCoding: true) else {
                    fatalError("can't convert token to data")
                }
            do {
                try mcSession.send(data, toPeers:
mcSession.connectedPeers, with: .reliable)
                print("Token send")
            } catch {
                fatalError("Could not send discovery token")
            }
        } else {
            print("You are not connected to other devices")
        }
    }
```

Om de token te kunnen versturen aan de hand van een Multipeer Connectivity sessie moet de token eerst worden omgezet naar een data object. Dit gebeurt aan de hand van een NSKeyedArchiver.[19] Vervolgens wordt de token verzonden naar de verbonden gebruiker. De delegate functie waarop dit bericht binnen komt is als volgt:

```
func session(_ session: MCSession, didReceive data: Data, fromPeer
peerID: MCPeerID) {
    niManager.startSession(data: data)
}
```

Nu de token binnen is kan er via de niManager instantie de sessie worden gestart door de startSession methode ervan aan te roepen. Aan deze methode wordt vervolgens het ontvangen data object mee gegeven.

3.3.4 NIManager

De class NIManager class erft van de volgende protocollen:

```
class NIManager: NSObject, NISessionDelegate, ObservableObject
```

De NIManager class bevat de volgende variabelen en constructor:

```
static let shared = NIManager()
@Published var distance: String?
var session: NISession?
var discoveryToken: NIDiscoveryToken?

private override init() {}
```

Zoals voorheen benoemd bevat de NIManager een 'distance' variabele van waaruit de ContenView class een listener is opgezet. De '@Published' geeft in dit geval aan dat er vanuit een StateObject instantie van deze class naar deze specifieke variabele wordt gekeken voor eventuele wijzigingen. Om dit toe te kunnen passen moet deze class erven van het ObservableObject protocol, zoals te zien is in de class diagram van figuur 7. De MPCManager start, na een token te hebben uitgewisseld, de Nearby Interaction sessie. Om te kunnen garanderen dat zowel de ContentView class, als de MPCManager class beschikt over dezelfde instantie van de NIManager, is er een static 'shared' instantie aangemaakt. Deze instantie kan door de static label worden benaderd buiten deze class door andere classes. Om te voorkomen dat er een nieuwe instantie aan wordt gemaakt beschikt de NIManager over een 'lege' private contructor. De start sessie methode is als volgt:

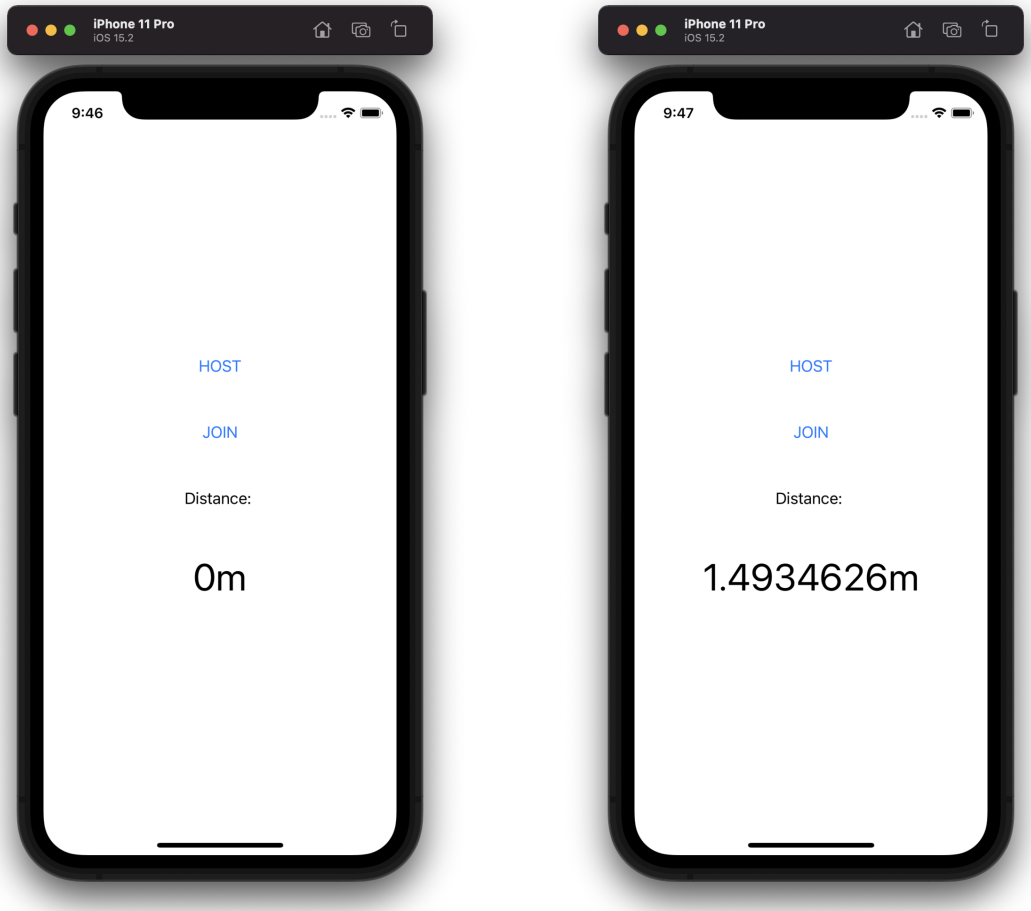
```
 func startSession(data: Data){
    guard let token = try? NSKeyedUnarchiver.unarchivedObject(ofClass:
NIDiscoveryToken.self, from: data) else {
        fatalError("Unexpectedly failed to encode discovery token.")
    }
    let configuration = NINearbyPeerConfiguration(peerToken: token)
    session?.run(configuration)
}
```

Aan deze methode wordt de discovery token van de peer meegegeven als een data object. Het data object wordt met behulp van een NSKeyedUnarchiver gecodeerd naar een discovery token.[19] Vervolgens wordt configuratie ingesteld en gestart. Na het starten van een Nearby Interaction sessie tussen twee peers worden de delegate functies aangeroepen en worden de locatie gegevens ten opzichte van elkaar gedeeld. Deze bestaan uit een afstand en richting. Voor het proof of concept is ervoor gekozen om alleen de afstand tussen beide apparaten weer te geven. De delegate functie die de afstand af vangt en opslaat in de 'distance' variabele is als volgt:

```
func session(_ session: NISession, didUpdate nearbyObjects:
[NINearbyObject]) {
    distance = String(nearbyObjects.first?.distance ?? 0)
}
```

Doordat er een listener functie op de distance variabele is gezet met behulp van een ObservableObject protocol wordt de distance in de ContentView automatisch gewijzigd elke keer als de delegate wordt aangeroepen met nieuwe data. Hoe dit er in praktijk allemaal uitziet is te zien aan de hand van de user interface in hoofdstuk 3.4.

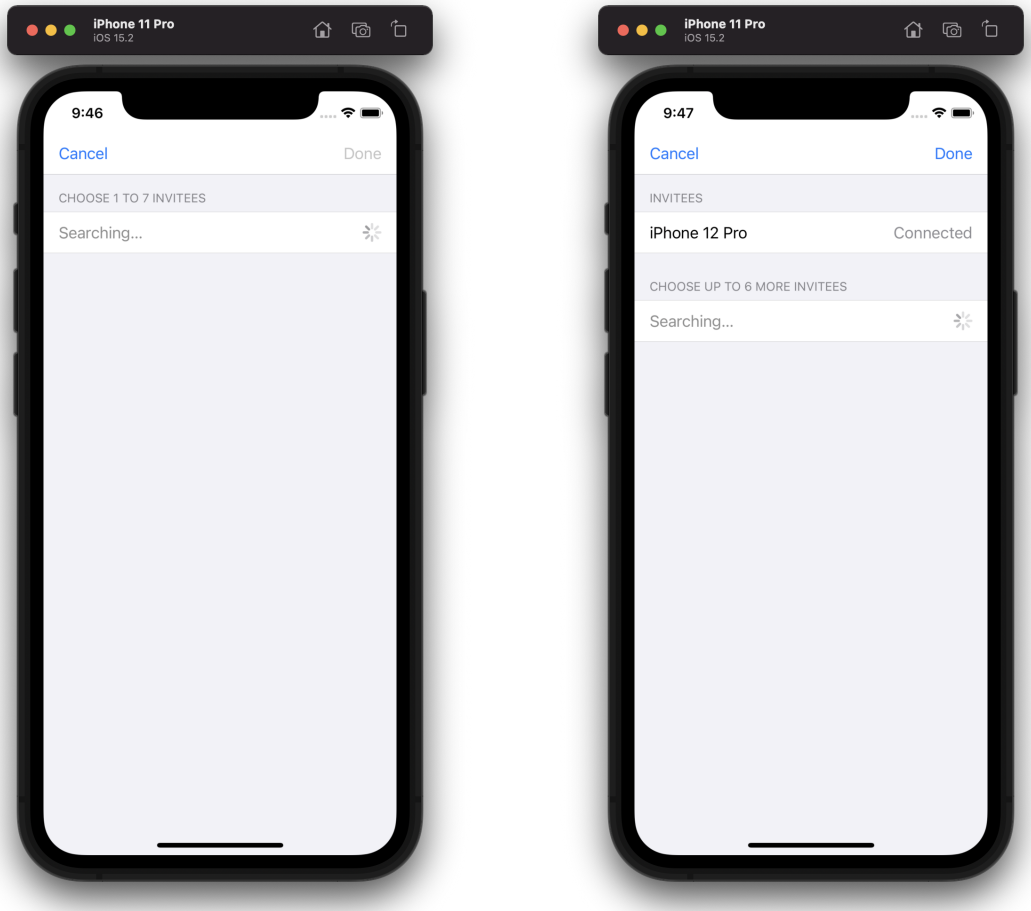
3.4 User Interface



(a) Initial state

(b) Connected state

Figuur 8: Proof of Concept iOS - Home screen



(a) Searching for hosts

(b) Connected to host

Figuur 9: Proof of Concept iOS - Connection screen

4 Ontwerpen

In dit hoofdstuk wordt het ontwerp en werk proces vastgelegd. Hier wordt er in gegaan op het ontwerpen van de API, de bijbehorende architectuur en de voorbeeld applicatie die aantoont hoe de plugin kan worden gebruikt, waarbij de voorbeeld applicatie tevens een indruk geeft van de mogelijkheden van UWB technologie op gebied van app ontwikkeling. Hiernaast wordt het werkproces benoemd voor het ontwerpen en realiseren van de voorbeeld applicatie.

4.1 Flutter plug-in

Nu de bouwstenen zijn gelegd kan de Flutter plugin worden ontworpen. Het ontwerpen van de plugin komt in dit hoofdstuk aan bod. Hier wordt er onder andere in gegaan op de architectuur en ontwerp keuzes van de plugin en de algemene opzet.

4.1.1 Ontwerpen van de API

Nu er een werkend proof of concept ligt, en er een 'general understanding' is van de werking van onder andere het Nearby Interaction Framework, het Multipeer Connectivity Framework en hoe een Flutter plugin in elkaar steekt kan er een API worden ontworpen voor de Flutter plugin. Hoewel Android, zoals vermeld is in 2.3 nog geen public API heeft vrij gegeven om te kunnen communiceren met hun UWB chips, wordt er wel goed nagedacht over een API waarbij er rekening wordt gehouden dat, zover als dit mogelijk is en zodra Android de API vrijgeeft, de Android kant nog kan worden toegevoegd.

Ten eerste wordt er gekeken naar het proof of concept dat gebouwd is voor iOS. Hier staat in ieder geval vast dat alvorens een Nearby Interaction Session kan worden opgestart er een token moet worden uitgewisseld zodat het Nearby Interaction protocol weet met welk apparaat hij of zij een verbinding moet aangaan. Om een token te kunnen uitwisselen moet in dit geval zoals beschreven is in hoofdstuk 2.2.2 een apparaat zich eerst kenbaar maken. Aannemelijk is dat er ook aan de Android kant een apparaat zich eerst op een manier kenbaar zou moeten maken richting andere UWB apparaten alvorens een verbinding kan worden opgezet. Dit gebeurd naar waarschijnlijkheid op Android aan de hand van het uitwisselen van een UWB adres zoals aangegeven is in hoofdstuk 2.3. Een mogelijke API call zou hier dus het kenbaar maken van het apparaat kunnen zijn.

Wanneer een 'host' apparaat zichzelf kenbaar heeft gemaakt voor andere UWB apparaten kan een ander UWB apparaat een verzoek sturen met als uiteindelijk doel het opzetten van een verbinding om data met elkaar uit te wisselen. Zoals te zien is in het proof of concept in hoofdstuk 3 zou bijvoorbeeld deze API call gebruikt kunnen worden om de verbinding te initialiseren. Dit brengt ons bij een mogelijke tweede API call. Namelijk, het versturen van een response bericht naar de 'host', met als doel een verzoek voor het opzetten van een verbinding tussen beide apparaten. Voor Android zou deze verbinding dus kunnen worden gebruikt om bijvoorbeeld de UWB adressen uit te wisselen.

Van uitgaande dat er met de eerste twee API calls voldoende zijn om een verbinding te kunnen initialiseren tussen twee UWB apparaten zullen de gegevens aan de native kant van zowel iOS en Android, zodra deze in de toekomst kan worden toegevoegd, moeten worden opgehaald. Gezien deze plugin zich focust op locatie bepalingen tussen twee of meer apparaten zal dit in dit geval gaan om locatie gegevens. Dit wordt dan ook tevens de derde API call. Namelijk, het ophalen van de locatie gegevens.

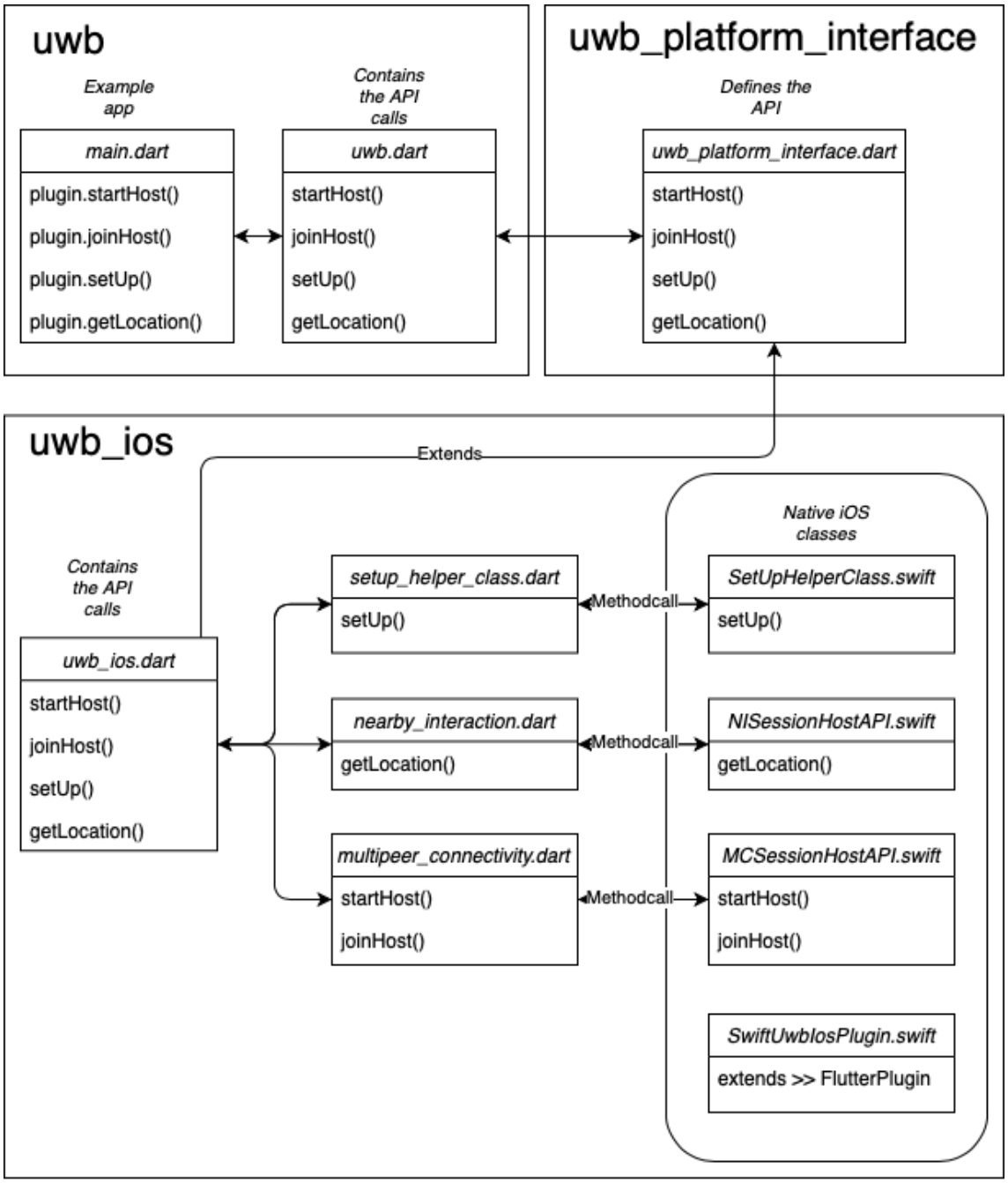
Verder is het netjes om de gebruiker te kunnen informeren of het apparaat wel geschikt is voor het opzetten voor een UWB verbinding. Ook dit moet met een call naar de native kant. Tot slot, kan er door middel van het onderzoek naar Flutter plugins in hoofdstuk 2.6 worden aangenomen dat er aan de native soms het één en ander aan set-up vereist is voor onder andere het opzetten van de benodigde methodchannels, reverse methodchannels en/of eventchannels. De call voor het achterhalen van de compatibiliteit van het apparaat en voor eventuele set-up purposes kan worden gedaan via een vierde API call. Namelijk, setUp. Ook kan deze setUp methode worden gebruikt om bijvoorbeeld UWB op een Android apparaat aan te zetten.

Dat brengt ons bij de volgende vier API calls:

- **startHost**
- **joinHost**
- **getLocation**
- **setUp**

Tijdens het ontwerpen van de API van deze plugin is er geprobeerd om met een minimum hoeveelheid API calls de plugin zo makkelijk, overzichtelijk en vriendelijk mogelijk in gebruik te maken. Tevens is er, zover als dit mogelijk is, rekening gehouden met het eventuele later toevoegen van het Android platform door het in gebruik nemen van API calls die zo eenvoudig en generiek mogelijk zijn.

4.1.2 Architectuur



Figuur 10: Class diagram - plugin API calls

De architectuur met betrekking tot het afhandelen van de API calls, zoals ze omschreven zijn in 4.1.1, wordt weergegeven in afbeelding 10. Let er wel op dat deze 'class diagram' alleen de methodes laat zien ter afhandeling van de API calls van de Flutter plugin. Op deze manier kan er een visuele indicatie worden gegeven hoe de plugin er in zijn geheel uitziet over de verschillende packages.

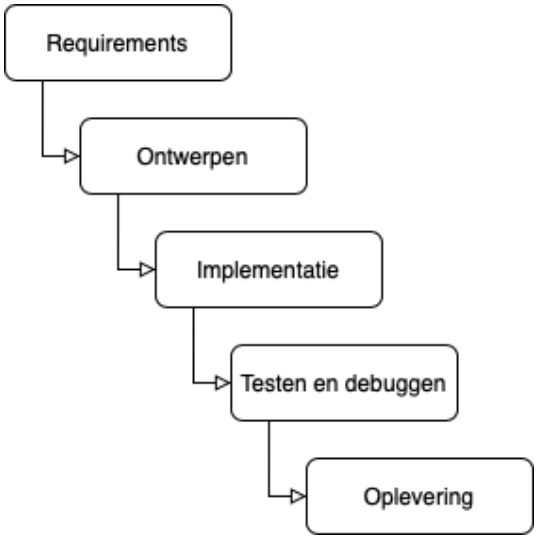
4.2 Voorbeeld applicatie - Breakout

Bij de plugin zal zoals voorheen beschreven een voorbeeld applicatie worden opgeleverd. Deze voorbeeld applicatie geeft een indruk waartoe de plugin tot in staat is, waarbij er op een overzichtelijke manier wordt aangetoond hoe de plugin kan worden gebruikt. Tevens dient deze voorbeeld applicatie ook als voorbeeld om aan te tonen waartoe UWB technologie op gebied van app ontwikkeling tot in staat is.

Als voorbeeld applicatie wordt er beroep gedaan op het klassieke 2D spel 'Breakout'. Voor deze applicatie zijn er twee telefoons benodigd. Waarbij één van de twee telefoons wordt gebruikt als meetpunt. Deze 'meet' telefoon wordt dan voor de speler op tafel neergelegd. De andere telefoon kan na een verbinding te hebben opgezet met de meet telefoon de Breakout game spelen door zijn of haar telefoon naar links of rechts te bewegen om het platform in het spel te bewegen. De telefoon die wordt gebruikt als meetpunt geeft ondertussen informatie over de telefoon waarop het spel wordt gespeeld, denk hierbij aan het weergeven van de afstand en richting.

4.2.1 Werkmethoden

Voor het ontwikkelen van de voorbeeld applicatie wordt er een waterval methode gebruikt. Er is gekozen voor het gebruik van de waterval methode gezien het geringe tijdsbestek waarin de voorbeeld applicatie moet worden voltooid, hierbij is het des te belangrijker om zicht te krijgen wanneer een bepaald product moet worden opgeleverd wanneer de tijd kort is. De waterval methode geeft dit inzicht. De waterval methode die voor deze applicatie wordt toegepast ziet er als volgt uit:



Figuur 11: Waterval methode

Per onderdeel zijn er een aantal producten en mijpalen. Deze zijn als volgt:

1. Requirements (looptijd: 1 week). In de requirements fase worden de requirements opgesteld. Deze worden vervolgens in MoSCoW notatie genoteerd. Tot slot worden de requirements gerefined tot user stories zodat deze in de implementatie fase kunnen worden geïmplementeerd.

2. Ontwerpen (looptijd: 1 week). In de ontwerpen fase van de waterval worden er aan de hand van de requirements mockups gemaakt. De mockups worden in dit geval gemaakt met behulp van Figma. De mockups zullen tevens worden voorzien van ondersteunde documentatie.

3. Implementatie (looptijd: 5 weken). In de implementatie fase word de applicatie gerealiseerd. Voor het ontwikkel proces wordt er Kanban gebruikt. Op deze manier wordt er aan de hand van tickets het implementatie proces bijgehouden en wordt er overzicht gecreëerd. Het Kanban bord zal bestaan uit een lijst met 'REQUIREMENTS', die alle requirements bevat, een 'BACKLOG' lijst waarin de requirements zijn gerefined tot tickets, een 'TO-DO' lijst, een 'IN PROGRESS' lijst, een 'QA' lijst (quality assurance), en een 'DONE' lijst. Voor een overzicht van het huidige Kanban bord wordt verwezen naar figuur 19 in de bijlage.

4. Testen en debuggen (looptijd: 1 week). In de testen en debuggen fase wordt er gecontroleerd of de applicatie, en de bijbehorende requirements die in de implementatie fase zijn

geïmplementeerd, voldoen aan alle voorwaarden en bijbehorende standaarden. Dit wordt gedaan aan de hand van onder andere Flutter format en Flutter analyze testen. Zie 5.4.

5. Oplevering (looptijd: 2 weken). Tijdens de oplevering fase wordt de code base van de voorbeeld applicatie, evenals de plugin, opgeleverd. De code base zal worden ondersteund met bijbehorende documentatie in de vorm van dit verslag.

4.2.2 Requirements

De volgende requirements zijn tot stand gekomen:

Nr.	Requirement	MoSCoW
R1	De applicatie moet worden gebouwd in Flutter.	Must
R2	De applicatie moet conform zijn aan de Flutter formattering standaarden.	Must
R3	De applicatie moet conform zijn aan de Flutter stijl richtlijnen.	Must
R4	De applicatie moet gebruik maken van de geschreven UWB plugin.	Must
R5	De applicatie moet aantonen wat de mogelijkheden zijn van de plugin.	Must
R6	De applicatie moet op een overzichtelijke manier aantonen hoe de plugin kan worden gebruikt.	Must
R6	De applicatie moet moet als open source applicatie beschikbaar worden gesteld.	Should

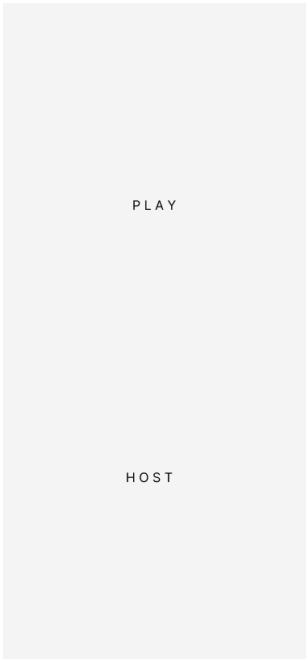
Tabel 1: Niet functionele requirements

Als gebruiker wil ik dat ...

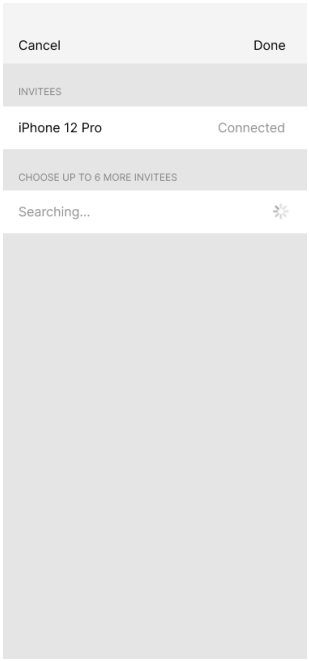
Nr.	Requirement	MoSCoW
UR1	mijn mobiel zichzelf kenbaar kan laten maken voor het opzetten van nearby interaction sessie, zodat deze kan worden gebruikt als meetpunt.	Must
UR1	ik op mijn mobiel, dat is ingesteld als meetpunt, de locatie gegevens zie van de telefoon die met mij verbonden is na het opzetten van een nearby interaction sessie, zodat ik weet waar deze telefoon zich bevindt.	Must
UR2	mijn mobiel kan verbinden met een mobiel dat zichzelf kenbaar heeft gemaakt als meetpunt, zodat ik locatie gegevens kan uitwisselen om het spel te kunnen spelen.	Must
UR3	ik het spel kan starten na het opzetten van een nearby interaction sessie, zodat het spel kan worden gespeeld.	Must
UR4	ik het platform in het spel kan bewegen door mijn telefoon naar links en rechts te bewegen, zodat ik de bal kan proberen terug te kaatsen.	Must
UR5	er een bal aanwezig is, zodat deze terug gekaatst kan worden om stenen te proberen weg te halen.	Must
UR6	er een stenen veld aanwezig is, zodat deze kunnen worden weggehaald door de bal er tegen aan te kaatsen om punten te scoren.	Must
UR7	ik kan zien wanneer ik het spel gewonnen of verloren heb, zodat ik weet dat het spel voorbij is.	Must
UR8	wanneer het spel voorbij is ik een score te zien krijg, zodat ik weet hoe goed ik het heb gedaan.	Should
UR9	er in het eind scherm een replay button is, zodat ik het spel makkelijk opnieuw kan spelen.	Should
UR10	er in het eind scherm een exit button zit, zodat ik het spel kan afsluiten.	Should

Tabel 2: Functionele requirements

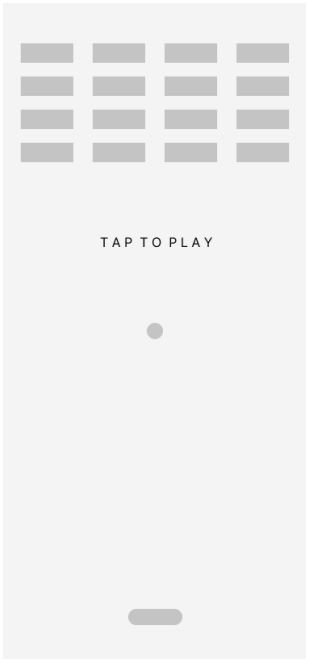
4.2.3 Mock-ups



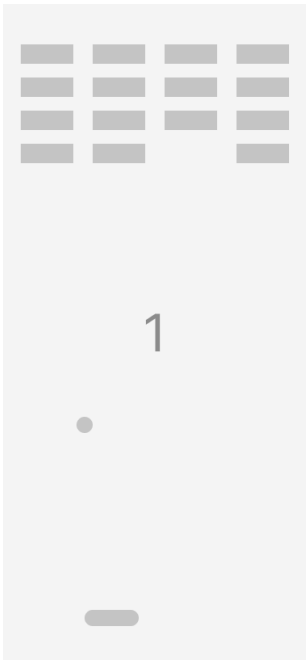
(a) Homescreen



(b) Connections



(c) Tap to play



(d) Gameplay

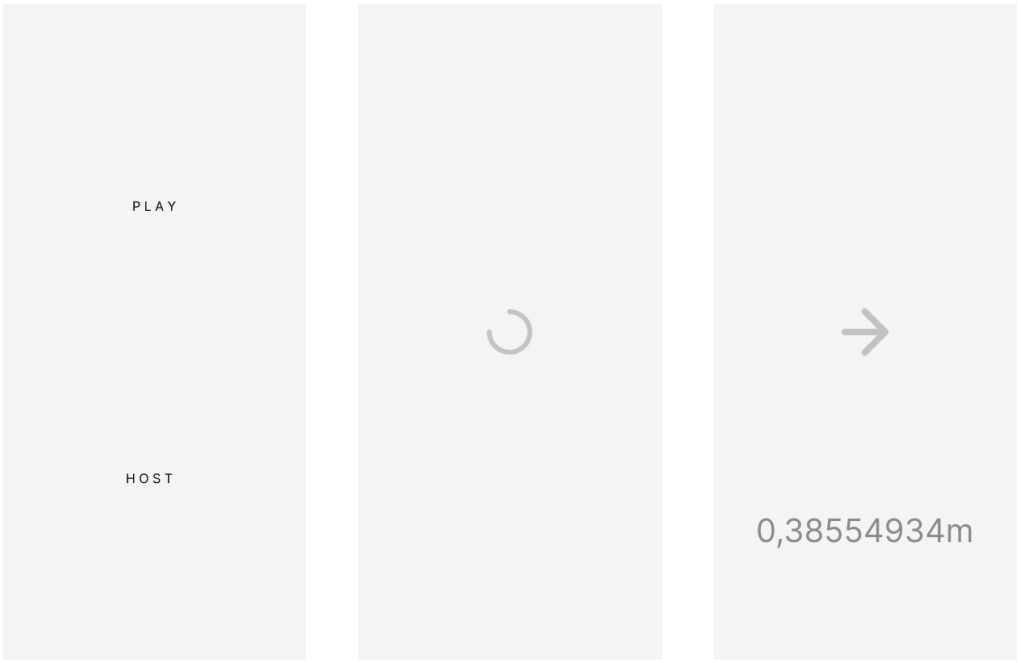
(e) Game over

(f) Victory

Figuur 12: Mockups - Speler

De voorbeeld applicatie beschikt over een homescreen, zoals te zien is in figuur 12, waarin een ‘play’ en een ‘host’ knop te vinden is. De ‘host’ knop start het proces voor het opzetten van de meet telefoon. De mockups voor het opzetten van de meet telefoon zijn te vinden in figuur 13. Wanneer de ‘play’ knop wordt ingedrukt wordt de gebruiker herleid naar een het ‘Connections’ scherm. Hier kan de speler een verbinding leggen met een telefoon dat wordt ingesteld als meet telefoon door de ‘host’ knop in te drukken in de homescreen. Wanneer een verbinding tot stand komt klikt de gebruiker op ‘Done’. Vervolgens wordt het ‘Tap to play’ scherm getoond. Door te klikken op het scherm het spel gestart. Nu kan de speler het platform onderin het scherm besturen door zijn telefoon naar links en rechts te bewegen. Wanneer de speler de bal mist wordt het ‘Game over’ scherm getoond. Wanneer een speler alle stenen bovenin het scherm wegspeelt wordt het

'Victory' scherm getoond. Tot slot kan de speler het spel afsluiten of het spel herstarten door op de 'exit' of 'replay' knop te drukken. Wanneer de 'replay' knop word ingedrukt wordt het spel gereset en krijgt de speler het beginscherm van het spel te zien zoals te zien is in het 'Tap to play' scherm.



(a) Homescreen (b) Waiting for player (c) Connected to player

Figuur 13: Mockups - Meet telefoon

De meet telefoon wordt ingesteld door op de homescreen de 'Host' knop in te drukken. Wanneer de host knop wordt ingedrukt wordt er een laad icoon getoond totdat een speler een verbinding maakt met de meet telefoon zoals getoond wordt in het 'Connections' scherm in figuur 12. Wanneer een verbinding tot stand komt wordt er op deze telefoon de afstand en richting van de locatie tussen de spelende telefoon en de meet telefoon weergegeven. Op deze manier wordt er samen met het spel de mogelijkheden van de plugin benut en weergegeven in een leuke en compacte app.

5 Realisatie

Nu de architectuur keuzes zijn gemaakt en het ontwerp op tafel ligt kan de software worden geschreven. Dit hoofdstuk gaat dieper in op verschillende technische en complexe aspecten van de plugin en de bijbehorende voorbeeld applicatie en zal aan de hand van voorbeelden hier een toelichting op geven.

5.1 Opzet

Nu er een ontwerp klaar ligt kan de plugin worden gerealiseerd. Voor het opzetten van de plugin worden de volgende command regels aangeroepen:

```
$ flutter create --template=package uwb
$ flutter create --template=package uwb_platform_interface
$ flutter create --template=plugin --platforms=ios -i swift uwb_ios
```

Hiermee is het aanmaken van de drie verschillende packages voltooid. Tot slot moeten er in de yaml files van zowel de uwb package als de uwb_ios package de dependencies en de default package worden geset. Voor de uwb package wordt het volgende toegevoegd:

```
dependencies:
  flutter:
    sdk: flutter
  uwb_ios:
    path: ../uwb_ios
  uwb_platform_interface:
    path: ../uwb_platform_interface

flutter:
  plugin:
    platforms:
      ios:
        default_package: uwb_ios
```

En voor de uwb_ios package het volgende:

```
dependencies:
  flutter:
    sdk: flutter
    path: ../uwb_platform_interface

flutter:
  plugin:
    implements: uwb
    platforms:
      ios:
        pluginClass: UwbIosPlugin
        dartPluginClass: UwbIos
```

Nu hiermee de basis ingesteld is moet alleen nog de ios package eerst een keer worden gerunned alvorens deze geopend wordt. Dit kan door naar de ios/example folder te navigeren en de volgende command te runnen:

```
$ flutter build ios --no-codesign
```

Op dit moment is alles correct ingesteld en kan er worden gewerkt aan de plugin.

5.2 Toelichting op code

Om een toelichting te geven op de belangrijke en/of complexe stukken code wordt er in dit geval specifiek gekeken naar de platform interface package die de API definieert en de platform package voor iOS. Gezien de app facing package voornamelijk een voorbeeld applicatie bevat die uitgebreid besproken wordt in hoofdstuk 5.3.

5.2.1 Platform Interface Package

De platform interface package definieert de API van de plugin. De API in de platform interface package is als volgt gedefinieerd:

```
/// Must be called once for initial setUp
Future<bool?> setUp() =>
    throw UnimplementedError('setUp() has not been implemented.');

/// Starts the advertiser
Future<bool?> startHost(String peerID, String serviceType) =>
    throw UnimplementedError('startHost() has not been implemented.');

/// Sends an invite to peer
Future<bool?> joinHost(String peerID, String serviceType) =>
    throw UnimplementedError('joinHost() has not been implemented.');

/// Callback method to get the location updates
Future<bool?> getLocation(
        {required Function(Map<dynamic, dynamic>) onLocation}) =>
    throw UnimplementedError('getLocation() has not been implemented.');
```

Op het moment dat er een platform specific package wordt toegevoegd moet deze package erven van deze package en de bovenstaande functies implementeren. Om te kunnen garanderen dat deze functies worden geïmplementeerd is de volgende code geschreven:

```
/// Constructs a UwbPlatform.
UwbPlatform() : super(token: _token);

static final Object _token = Object();

/// The instance of [UwbPlatform] to use.
/// Must be set before accessing.
static UwbPlatform get instance => _instance;

/// Platform-specific plugins should set this with their own
  platform-specific
/// class that extends [UwbPlatform] when they register themselves.
static set instance(UwbPlatform instance) {
  PlatformInterface.verify(instance, _token);
  _instance = instance;
}

/// Should only be accessed after setter is called.
static late UwbPlatform _instance;
```

Wanneer er een platform specific package wordt toegevoegd er een Dart file erft van deze class wordt er verwacht dat de instantie van de class wordt gebruikt om de UWB platform instantie in te stellen.

5.2.2 platform specific package - iOS

De platform specific package implementeert zoals hierboven vermeld de UWB.class van de platform interface package. Deze Dart file passed vervolgens de API calls door naar de desbetreffende class die aan de hand van een methodchannel weer het bericht doorstuurt naar de native iOS kant. Deze class ziet er als volgt uit:

```
/// Extends UwbPlatform. Directs method calls to the corresponding
wrapper and helper classes
class UwbIos extends UwbPlatform {
  /// Sets the UbwPlatform instance
  static void registerWith() {
    UwbPlatform.instance = UwbIos();
  }

  /// Initialises MCSessionWrapper
  MCSessionWrapper mCSession = MCSessionWrapper();

  /// Initialises NISessionWrapper
  NISessionWrapper nISession = NISessionWrapper();

  /// Initialises the setupHelperClass
  SetupHelperClass setupHelperClass = SetupHelperClass();

  @override
  Future<bool?> setUp() {
    /// Sets the method callHandler for the Nearby Interaction Session
    /// This needs to be set after the corresponding
FlutterMethodChannel has been set in the native swift file
    nISession.setUp();

    /// Checks device iOS-version & UWB compatibility
    /// returns (true) if compatible
    return setupHelperClass.setUp();
  }

  @override
  Future<bool?> startHost(String peerID, String serviceType) {
    return mCSession.startHost(peerID, serviceType);
  }

  @override
  Future<bool?> joinHost(String peerID, String serviceType) {
    return mCSession.joinHost(peerID, serviceType);
  }

  @override
  Future<bool?> getLocation(
      {required Function(Map<dynamic, dynamic>) onLocation}) {
    return nISession.getLocation(onLocation: onLocation);
  }
}
```

Hier kan er onder andere terug worden gezien dat er aan de hand van de registerWith methode de UWBplatform instantie wordt ingesteld. Ook worden alle vier de API calls hier geïmplementeerd. Om de setup call zorgt er in dit geval ook voor dat de reverse methodchannel in de NISession-Wrapper class wordt ingesteld nadat de methodchannel aan de native kant is geïnitialiseerd. Tot slot stuurt deze een true terug nadat het bericht terug komt van de native kant om te kijken of het apparaat Nearby Interaction ondersteund.

De MCSessionWrapper class bevat in dit geval een methodchannel om de advertiser te starten en een invite request te kunnen afhandelen. Dit ziet er als volgt uit:

```
/// Contains all the Multipeer Connectivity Framework calls
class MCSessionWrapper {
  static const MethodChannel _channel =
      MethodChannel('com.baseflow.uwb/mc_session');

  /// Starts the advertiser
  Future<bool?> startHost(String peerID, String serviceType) async {
    final Map<String, String> parameters = {
      'peerID': peerID,
      'serviceType': serviceType
    };
    final bool? hostingProcess =
        await _channel.invokeMethod('MCSession.startHost', parameters);
    return hostingProcess;
  }

  /// Sends an invite to peer
  Future<bool?> joinHost(String peerID, String serviceType) async {
    final Map<String, String> parameters = {
      'peerID': peerID,
      'serviceType': serviceType
    };
    final bool? joiningProcess =
        await _channel.invokeMethod('MCSession.joinHost', parameters);
    return joiningProcess;
  }
}
```

En de NISessionWrapper class bevat een reverse methodchannel om berichten te kunnen ontvangen vanuit de native iOS kant. Dit ziet er als volgt uit:

```
/// Contains all the Nearby Interaction Framework calls
class NISessionWrapper {
  late final Function(Map<dynamic, dynamic>) _onLocation;
  static const MethodChannel _locationChannel =
      MethodChannel('com.baseflow.uwb/ni_session_location');

  /// Sets the method callHandler for the Nearby Interaction Session
  /// This needs to be set after the corresponding
FlutterMethodChannel has been set in the native swift file
  void setUp() {
    _locationChannel.setMethodCallHandler(_handleMethodCall);
  }
  /// Callback method to get the location updates
  Future<bool?> getLocation(
      {required Function(Map<dynamic, dynamic>) onLocation}) async {
    _onLocation = onLocation;
    return true;
  }
  Future<void> _handleMethodCall(MethodCall call) async {
    switch (call.method) {
      case 'updateLocation':
        final Map<dynamic, dynamic> location =
            call.arguments as Map<dynamic, dynamic>;
        _onLocation(location);
        break;
      default:
        throw MissingPluginException();
    }
  }
}
```

Aan de native iOS kant is er een class die het Flutterplugin protocoll bevat. Deze zorgt ervoor dat er de benodigde methodchannels aan de native kant kunnen worden opgezet. Gezien er meerdere methodchannels zijn aan de native kant roept deze class de setUp methodes aan van de SetUpHelperClass, de NISessionHostApi en de MCSessionHostAPI en geeft hierbij de registrar.messenger variabele door die benodigd is voor het opzetten van de methodchannels. Deze class ziet er als volgt uit:

```
public class SwiftUwbIosPlugin: NSObject, FlutterPlugin {
  public static func register(with registrar: FlutterPluginRegistrar) {

      SetupHelperClass.setUp(binaryMessenger: registrar.messenger())

      if #available(iOS 14.0, *) {
          NISessionHostApi.setUp(binaryMessenger:
registrar.messenger())
          MCSessionHostApi.setUp(binaryMessenger:
registrar.messenger())
      }
  }
}
```

Gezien de setupclass nieuw is ten opzichte van de native classes zoals ze zijn weergegeven in het proof of concept in hoofdstuk 3 wordt deze hier nog kort toegelicht. Deze bevat een methodchannel om het bericht vanuit de Dart kant te kunnen ontvangen en een methode die de check uitvoert om te kijken of het apparaat compatibel is met Nearby Interaction. Deze class ziet er als volgt uit:

```
public class SetupHelperClass {
    public static func setUp (binaryMessenger: FlutterBinaryMessenger)
{
        let session = SetupHelperClass()

        let channel = FlutterMethodChannel(name:
"com.baseflow.uwb/setup_helper_class", binaryMessenger:
binaryMessenger)

        channel.setMethodCallHandler {(call: FlutterMethodCall,
result: FlutterResult) -> Void in
            switch call.method {
            case "SetupHelperClass.setUp":
                result(session.checkDevicecompatibility())
            default:
                result(FlutterMethodNotImplemented)
            }
        }
    }

    public func checkDevicecompatibility() -> Bool {
        if #available(iOS 14.0, *) {
            var shared: NISessionHostApi
            shared = NISessionHostApi.shared!
            return shared.checkDeviceCompatibility()
        } else {
          return false
        }
    }
}
```

De andere twee classes implementeren de methodchannel op de zelfde manier. Al dan wel niet met hen eigen specifieke cases. Gezien de werking van deze classes, op de methodchannels na, grotendeels al zijn besproken in hoofdstuk 3 wordt er hier geen verdere uitleg gegeven maar wordt er in dit geval verwezen naar hoofdstuk 3 voor een inzage in de implementatie van deze specifieke frameworks.

5.3 Voorbeeld applicatie: Breakout

Nu de UWB plugin is gerealiseerd kan de UWB plugin worden gebruikt in een Flutter applicatie. Om een voorbeeld applicatie op te bouwen wordt er aan de example.lib folder in de uwb map een main.dart file toegevoegd. Hierin komt de voorbeeld applicatie te staan die de functionaliteit van de plugin weergeeft. Deze applicatie wordt geschreven in Dart waarbij de applicatie zal worden gebaseerd op de mockups en requirements zoals ze beschreven staan in hoofdstuk 4.2.2 en 4.2.3.

In dit hoofdstuk is ervoor gekozen om alleen in te gaan op de twee belangrijkste schermen van de Breakout applicatie, namelijk het game scherm en het host scherm. De overige code is in dit geval te vinden in de bijlage in hoofdstuk 8.1.

5.3.1 Host scherm

Het host scherm geeft in dit geval informatie over de telefoon waarop het Breakout spel wordt gespeeld. Hiervoor moet de telefoon zich eerst kenbaar voor het opzetten van een Nearby Interaction sessie. Om dit te kunnen doen moet eerst de plugin.setUp methode worden aangeroepen om te zien of de telefoon geschikt is voor UWB en wordt er indien nodig een error bericht getoond. Hierna kan het host proces worden gestart met de plugin.startHost methode. Nadat een andere telefoon verbinding heeft gemaakt met de host telefoon kunnen er locatie gegevens worden opgehaald en worden getoond aan de gebruiker.

5.3.1.1 Opbouw UI

De UI van het host scherm wordt als volgt opgebouwd:

```
        @override
          Widget build(BuildContext context) {
            return Scaffold(
              backgroundColor: Colors.grey[900],
              body: Center(
                child: Stack(
                  children: <Widget> [
                    LoadingIndicator(x: 0, y: 0, waitingForPeer:
_waitingForPeer),
                    ErrorMessage(x: 0, y: -0.5, error: _error),
                    Arrow(x: 0, y: 0, angle: _angle),
                    Distance(x: 0, y: 0.5, distance: _distance),
                  ],
                ),
              ),
            );
          }
```

De widgets die in de stack worden getoond zijn te vinden in de bijlage in hoofdstuk 8.1.1. De x en y waarden die worden meegegeven bepalen de positie van het betreffende element.

5.3.1.2 Belangrijke methoden

Voor het opzetten van het host scherm zijn de volgende vier methoden opgesteld:

1. startHosting

```
Future<void> startHosting() async {
    await _plugin.startHost(deviceName, serviceType);
    _plugin.getLocation(onLocation: onLocation);
    _waitingForPeer = false;
}
```

In de bovenstaande methode wordt de plugin.startHost methode van de plugin aangeroepen. Hier wordt een naam van het apparaat meegegeven evenals het service type. Op het moment dat er een verbinding ontstaat met een peer dan komt deze uit zijn await en kan de plugin.getLocation methode worden aangeroepen om de locatie gegevens op te halen. Deze geeft vervolgens zijn locatie gegevens door aan de onLocation methode aan de hand van een callback.

2. initPlatformState

```
Future<void> initPlatformState() async {
    ///Needed for initial set-up and checks device compatibility
    onSetup(await _plugin.setUp());
}
```

Alvorens de host methode moet de plugin.setUp methode worden aangeroepen. Deze geeft een true terug bij een succesvolle setUp. Dit betekend dat het apparaat geschikt is voor het gebruikt van plugin en een nearby interaction sessie kan starten. Op het moment dat er een false terug komt, dan wordt er aan de hand van een onSetup methode een error bericht getoond.

3. onSetup

```
void onSetup(bool? status) {
    if (status != null && !status) {
      setState(() {
        _error = """Device is incompatible with this app.
        Please check device OS version and UWB compatibility.
        For Apple users iOS version should be 14.0 or higher.""";
      });
    }
}
```

4. onLocation

```
void onLocation(Map location) {
    var _direction = location['direction'];
    var _directionArray = _direction.split(",");
    var _x = double.parse(_directionArray[0]);
    var _y = double.parse(_directionArray[1]);
    var _z = double.parse(_directionArray[2]);

    setState(() {
      _distance = double.parse(location['distance']);
      if (_x == 0.0 && _y == 0.0) {
        _angle = null;
      } else {
        _angle = math.atan2(_x, _y);
      }
    });
}
```

De bovenstaande methode wordt als callback aangeroepen bij het ontvangen van locatiedata. De methode laat in dit geval zien hoe de locatie gegevens kunnen worden omgezet naar een richting en afstand. Tevens is het mogelijk om los de x, y en z as te ontvangen.

5.3.2 Game scherm

In dit scherm wordt het spel gespeeld, wordt er verbinding gelegd met de host telefoon, en wordt er gebruikt gemaakt van de UWB plugin om elementen in het spel te kunnen bewegen.

5.3.2.1 Opbouw UI

```
@override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: startGame,
      child: Scaffold(
        backgroundColor: Colors.grey[900],
        body: Center(
          child: Stack(
```

```
            children: [
              BrickField(brickFieldList: brickFieldList),
              ScreenOverlay(gameHasStarted: gameHasStarted),
              Ball(x: ballX, y: ballY, ballKey: ballKey),
              Paddle(x: paddleX, y: paddleY),
              Score(score: score, gameHasStarted: gameHasStarted),
            ],
          ),
        ),
      ),
    );
  }
```

De onderdelen van de UI die in de stack worden weergegeven zijn toegevoegd aan de bijlage. Deze zijn te vinden in hoofdstuk 8.1.2.

5.3.2.2 Belangrijke methoden

1. Joinen van een host

```
Future<void> joinHost() async {
    await _plugin.joinHost("test-device", "uwb-test");
    _plugin.getLocation(onLocation: onLocation);
}
```

   Na het klikken op de 'Play' knop in het home scherm wordt het game scherm gestart. De initialisatie methode in dit geval direct na het openen van dit scherm een joinHost methode om te zoeken naar hosts. Wanneer er beschikbare hosts zijn er een verbinding tot stand is gekomen worden er locatie gegevens op gehaald via de callback functie.

2. ophalen locatie gegevens

```
void onLocation(Map location) {
    var _direction = location["direction"];
    var _directionArray = _direction.split(",");
    double _x = double.parse(_directionArray[0]);
    double _y = double.parse(_directionArray[1]);

    setState(() {
      _distance = double.parse(location["distance"]);
      if (_x == 0.0 && _y == 0.0) {
        _angle = null;
      } else {
        _angle = math.atan2(_x, _y);
        _xPosition = _x;
      }
    });
}
```

   Net zoals in het host scherm worden er hier lactie gegevens opgehaald. In dit geval wordt er gebruik gemaakt van de doorgegeven x positie om straks het platform te kunnen bewegen.

3. Genereren van de brickFieldList

```
List<Brick> generateBrickFieldList() {
    List<Brick> brickFieldList = [];

    for (double y = -0.5; y >= -0.8; y -= 0.1) {
      for (double x = -0.9; x <= 1; x += 0.45) {
        brickFieldList
            .add(Brick(x: x, y: y, color: brickColor, brickKey:
GlobalKey()));
      }
    }
    return brickFieldList;
}
```

Tijdens het opzetten van het game scherm wordt er ook in de initialisatie een brickFieldList gegenereerd die wordt doorgegeven aan de BrickField class om een stenen veld te creëren. Hier bevat elke steen een global key die later wordt gebruikt voor het berekenen van collision. De x en y waarden die worden gebruikt in de for loops bepalen de positie van elke brick.

4. Starten van het spel

```
void startGame() {
    if (gameHasStarted == false) {
      gameHasStarted = true;
      Timer.periodic(const Duration(milliseconds: 8), (Timer
timer) {
        checkWallCollision();
        if (ballY < -0.4) {
          checkBrickCollision();
        }
        if (ballY > -0.8) {
          checkPlatformCollision();
        }
        moveBall();
        movePlatform();
        if (isPLayerDead()) {
          playerIsDead = true;
          timer.cancel();
          _showDialog('G A M E  O V E R', Colors.redAccent);
        }
        if (score == 20) {
          timer.cancel();
          _showDialog('Y O U  W I N', Colors.green);
        }
      });
    }
}
```

Nadat er een verbinding tot stand is gekomen en er een stenen veld is gegenereerd, kan de speler het spel starten door op het scherm te klikken. Vervolgens wordt de bovenstaande methode aangeroepen. Deze zorgt ervoor dat er met behulp van een timer er elke milliseconde checks worden uitgevoerd en er beweging plaatsvind in het spel. Hier worden ook de win en verlies condities gecontroleerd en wordt er een dialoog venster getoond bij het winnen of verliezen van het spel.

5. Bewegen van de bal De bal in het spel wordt bewogen door de bal x en y coördinaten door te geven. De x coördinaat loopt op van -1, wat de linker boundery van het scherm aangeeft tot +1, wat de rechter boundery aangeeft van de het scherm. Hetzelfde geld voor het y coördinaat die loopt van boven in het scherm met -1, naar onderin het scherm +1.

   Elke miliseconde wordt er aan de hand van de richting van de bal de x en y waarden opgehoogd of verlaagt. Dit gebeurd als volgt:

```
void moveBall() {
    setState(() {
      ///vertical movement
      if (ballYDirection == direction.DOWN) {
        ballY += 0.01;
      } else if (ballYDirection == direction.UP) {
        ballY -= 0.01;
      }

      ///horizontal movement
      if (ballXDirection == direction.RIGHT) {
        ballX += 0.02;
      } else if (ballXDirection == direction.LEFT) {
        ballX -= 0.02;
      }
    });
}
```

6. Bewegen van het platform Het bewegen van het platform gebeurd door de x coördinaat van platform te updaten. Het x coördinaat is gebaseerd op de x waarde die wordt doorgegeven via de plugin. Na het ophalen van de locatie gegevens kan op deze manier makkelijk het platform worden bewogen.

```
void movePlatform() {
    if (_xPosition != null) {
      setState(() {
        paddleX = 2 * -_xPosition!;
      });
    }
}
```

7. Winnen of verliezen

   Voor het winnen of verliezen wordt er een AlertDialog getoond. Deze verwacht een String voor het weergeven van de titel en een kleur voor de kleur van de AlertDialog. De volledige methode is te vinden in de bijlage in 8.1.3.7.

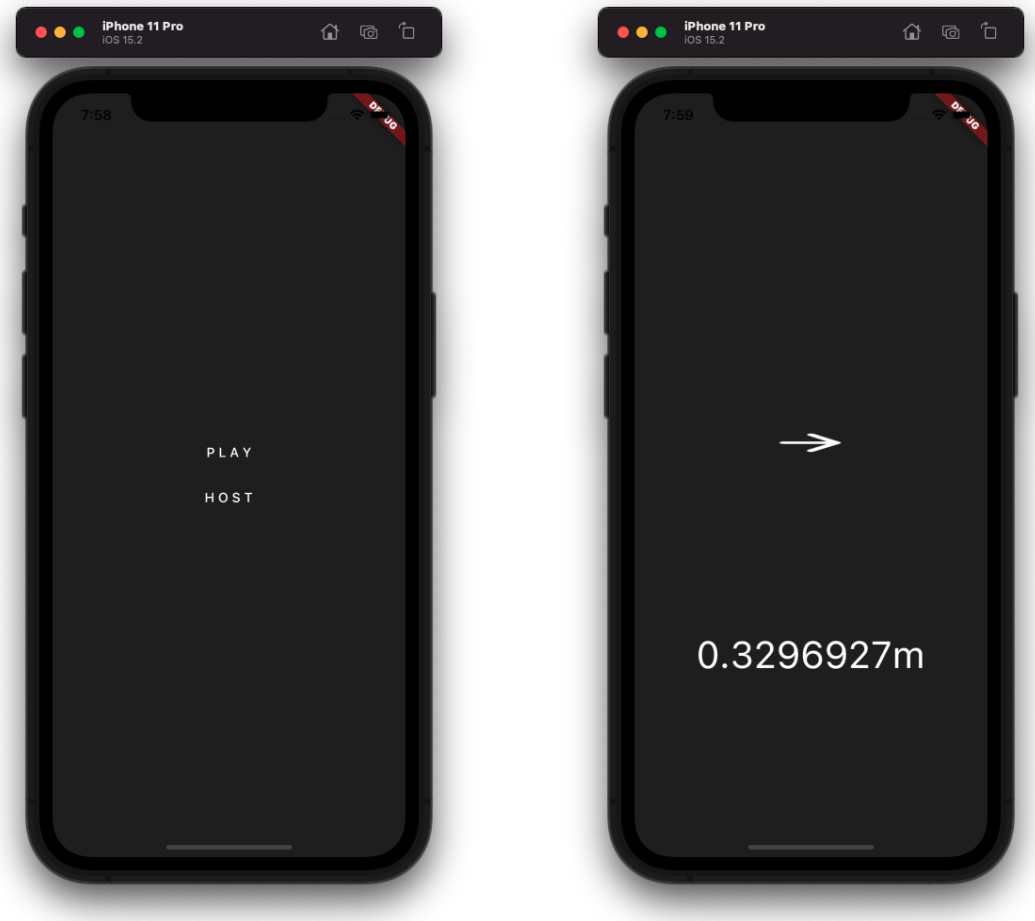
8. Collision

   Om te concluderen of er collision is tussen een bal en een steen of platform wordt er gebruik gemaakt van de findRenderObject methode [25]. Deze geeft de positie weer van de desbetreffende container waarop deze methode wordt toegepast. Vervolgens wordt er gekeken of de containers met elkaar overlappen. Wanneer dit gebeurd wordt de richting van de bal omgedraaid naar aanleiding van waar de bal de steen of platform heeft geraakt. De volledige methode hiervan is te vinden in de bijlage in 8.1.3.9.

5.3.2.3 Overige methoden

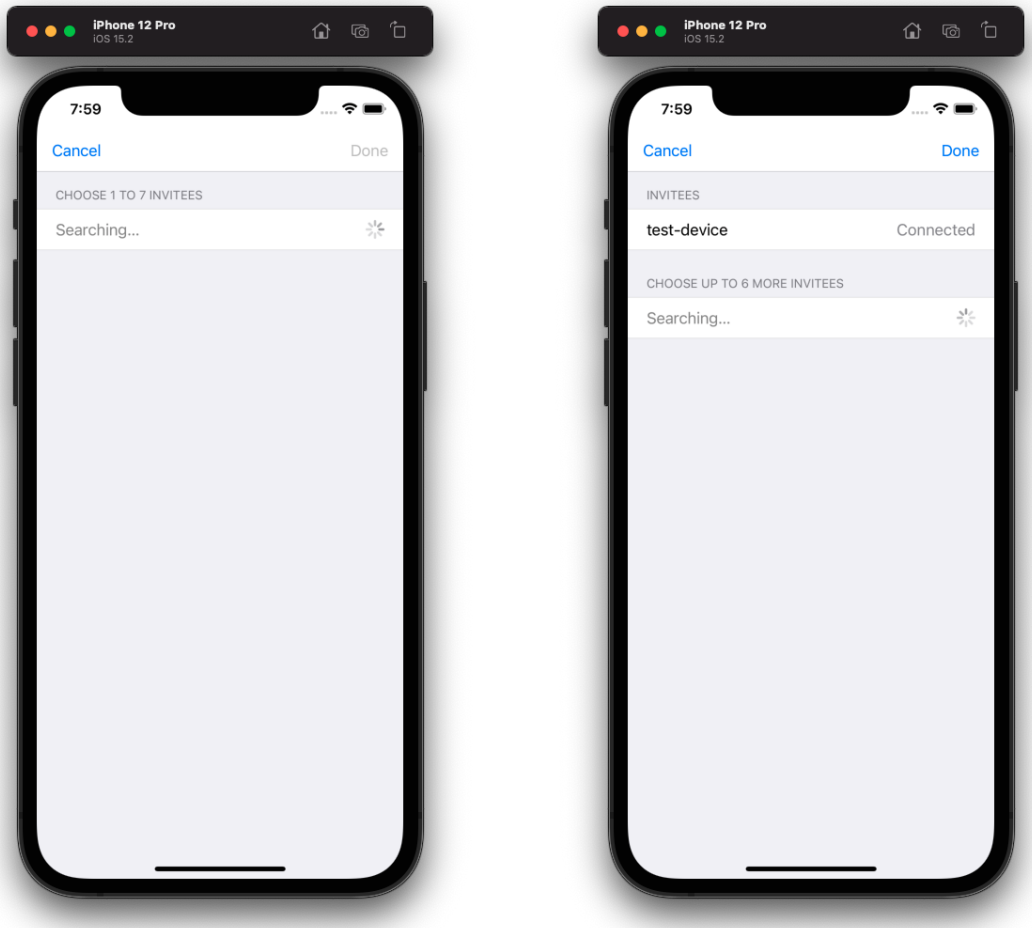
Overige methoden van de Breakout applicatie komen neer op het wijzigen van de horizontale en verticale richting van de bal, het resetten van het spel, wall collision en het controleren of het spel gewonnen is of verloren. Deze methoden zijn terug te vinden in de bijlage in 8.1.2.

5.3.3 User Interface



(a) Thuis scherm (b) Host scherm

Figuur 14: Voorbeeld applicatie - Thuis scherm en host scherm



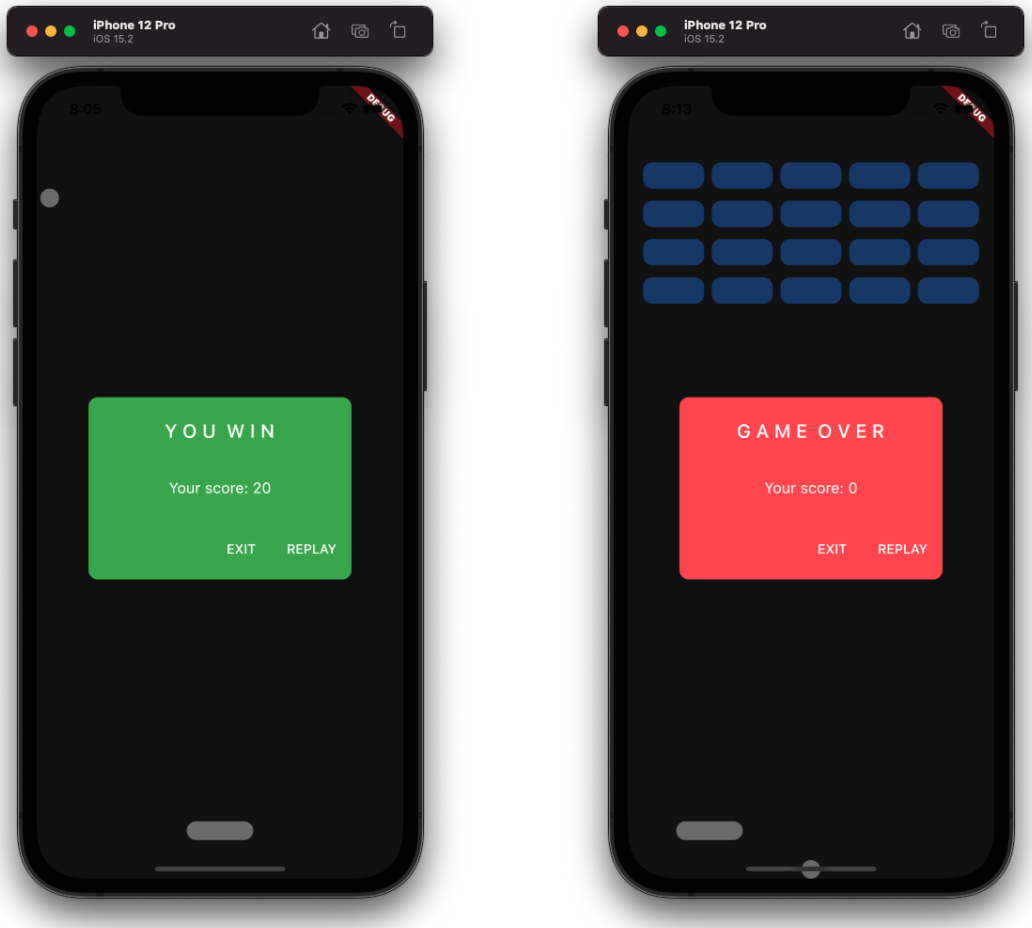
(a) Zoeken naar hosts

(b) Verbonden met host

Figuur 15: Voorbeeld applicatie - connecties

(a) Game scherm (b) Gameplay

Figuur 16: Voorbeeld applicatie - Breakout gameplay



(a) Victory scherm (b) Game over scherm

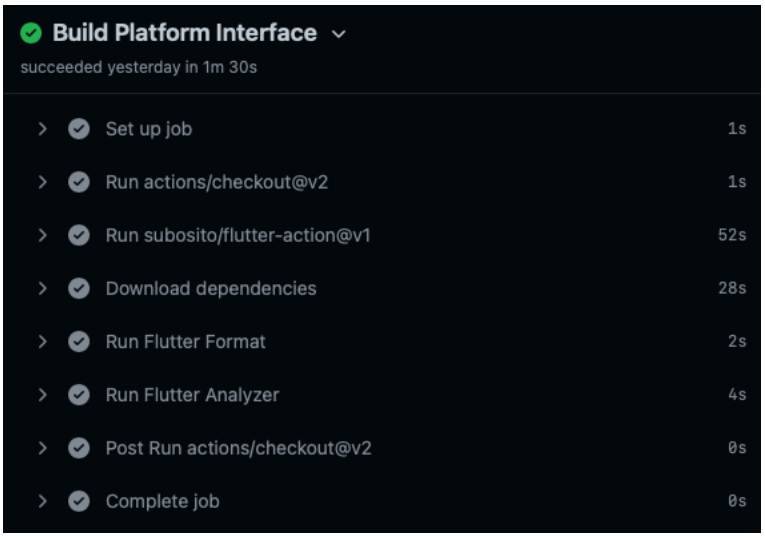
Figuur 17: Voorbeeld applicatie - Eind schermen

5.4 Code quality and testing

Alvorens er een pull request wordt gemerged op de main branch wordt er op de pull request een aantal testen gedraaid voor code quality purposes. De gehele plugin, inclusief de drietal packages, staat als 1 project op Git. In de '.github' folder van het project bevindt zich in de workflow map een yaml file voor elke package van de plugin. Deze yaml files bevatten een aantal testen die worden uitgevoerd op het moment dat er een push gaat naar Git. De testen die de yaml files bevatten komen nagenoeg voor de drie packages overeen. Met uitzondering voor de platform interface package, die geen build versie hoeft te bouwen van een example app. De belangrijkste testen zien er in de yaml file als volgt uit:

```
# Make sure the stable version of Flutter is available
- uses: subosito/flutter-action@v1
  with:
  channel: 'stable'

# Download all Flutter packages
- name: Download dependencies
  run: flutter pub get
  working-directory: ${{env.source-directory}}

# Run Flutter Format to ensure formatting is valid
- name: Run Flutter Format
  run: flutter format --set-exit-if-changed .
  working-directory: ${{env.source-directory}}

# Run Flutter Analyzer
- name: Run Flutter Analyzer
  run: flutter analyze
  working-directory: ${{env.source-directory}}

# Build iOS version of the example App
- name: Run iOS build
  run: flutter build ios --release --no-codesign
  working-directory: ${{env.example-directory}}
```

De testen die worden uitgevoerd zijn onder andere de testen om te kijken of er een stabiele versie van Flutter wordt gebruikt in de app, vervolgens worden alle dependencies gecheckt door deze te downloaden. Er wordt een Flutter format aangeroepen om te zien of de Flutter code voldoet aan de Flutter formattering eisen en er wordt een Flutter analyzer aangeroepen.[2] De Flutter analyzer checkt de Flutter code op mogelijke fouten en code kwaliteit.[9] Tot slot wordt er gekeken of er een foutloze build kan worden gedraaid. Een voorbeeld van een geslaagde test, van in dit geval de platform package, ziet er op Git als volgt uit:



Figuur 18: Ultra Wideband bandbreedte

6 Resultaten en Conclusie

In dit hoofdstuk is de uiteindelijke conclusie van deze opdracht vastgelegd. Hier wordt er aan de hand van een korte terugblik de kern van het onderzoek vastgelegd door het beantwoorden van de hoofdvraag en bijbehorende deelvragen.

Om in te kunnen gaan op het beantwoorden van de hoofdvraag worden eerst de deelvragen beantwoord.

1. Wat is UWB en wat is de achterliggende technologie achter ultra wideband?

   Zoals beschreven is in hoofdstuk 2.1 is te lezen dat UWB valt onder de radio technologieën. UWB kenmerkt zich door de brede frequentie waarop deze technologie kan en mag uitzenden. Dit betekend dat UWB beschikt over veel bandbreedte en samen met de techniek waarop UWB zijn signaal uitzend uitermate geschikt maakt voor onder andere locatiebepaling. Voor een uitgebreide uitleg voor de werking van UWB wordt dan ook verwezen naar hoofdstuk 2.1.1.

2. Wat zijn mogelijke toepassingen bij het inzetten van ultra wideband technologie op Android en Apple?

   Om te achterhalen wat de mogelijke toepassingen zijn voor het inzetten van UWB op de mobiele platformen is er eerst onderzoek gedaan naar de beschikbaarheid van UWB op de desbetreffende mobiele platformen Android en iOS. De beschikbaarheid staat uitgebreid beschreven in hoofdstuk 2.1.2. Hierin is onder andere terug te zien dat de het gebruik van UWB in mobiele apparaten alleen maar toeneemt. En dat de beschikbaarheid, en hiermee de inzet, ook steeds meer wordt uitgebreid. Wel is er te zien aan de hand van het onderzoek in hoofdstuk 2.2 en hoofdstuk 2.3 dat hier de focus van de inzet van deze technologie wordt gelegd op locatiebepaling. Dit is tevens te zien aan de beschikbare (en onbeschikbare niet publieke) API's op dit moment ondersteuning bieden voor het uitwisselen van locatie gegevens, zoals te lezen is in hoofdstuk 2.2.1, en niet zozeer zijn gemaakt voor het versturen of ontvangen van data berichten in zijn geheel. Voor mogelijke toepassingen op het gebied van mobile development moet op dit moment dan ook gedacht worden aan location based services, waarbij precisie en realtime tracking belangrijke rollen spelen.

3. Op welke manier kan je een Flutter plug-in ontwikkelen voor UWB technologie en in hoeverre is het mogelijk een plug-in te schrijven waarbij de API voor zowel iOS als Android generiek is?

   Het ontwikkelen van een Flutter plugin die een gebruiker ervan in staat stelt om de UWB technologie te gebruiken aan de native kant vereist kennis van de mogelijkheden van UWB op de native platformen en kennis van het ontwikkelen en opzetten van een Flutter plugin. Hier komen hoofdstukken 2.2, 2.3, 2.6, 2.7, en hoofdstuk 3 aan te pas. Naar aanleiding van het onderzoek hiernaar en de uitwerking van het proof of concept kan is er met de beschikbare informatie een Flutter plugin ontworpen zoals weergegeven in hoofdstuk 4.1. Het ontwerp kenmerkt zich door de variatie op de federated architectuur waarbij er samen met de ontworpen API rekening wordt gehouden met het later toevoegen van een Android package. Dit maakt, hoewel er dus geen public API bekend is voor Android, het mogelijk om toch op deze manier, zover als dat mogelijk is, een generieke API te schrijven. Voor de uitwerking van dit ontwerp wordt er verwezen naar hoofdstuk 5.

4. Welke applicatie zou als proof of concept de mogelijkheden en functionaliteiten van deze technologie goed kunnen weergeven en hoe ziet deze eruit?

   Zoals voorheen aangegeven leent UWB zich vooral goed uit voor location based services waarbij realtime tracking en precisie belangrijke rollen spelen. De Flutter plugin bevat in dit geval dan ook een voorbeeld applicatie die laat zien hoe de ontwikkelde Flutter plugin kan worden gebruikt om zo een dergelijke applicatie te ontwikkelen. De voorbeeld applicatie die hiervoor ontwikkeld is laat de gebruiker van de plugin een verbinding opzetten tussen twee iPhone gebruikers om vervolgens met elkaar realtime locatie gegevens ten opzichte van elkaar te delen. Eén telefoon gebruikt de realtime locatie data om het klassieke 2d spel "Breakout"te kunnen spelen. Dit is mogelijk door het gebruik van de realtime locatie tracking mogelijkheden van de plugin om het platform in het spel te kunnen bewegen. De andere telefoon geeft in dit geval de realtime locatie data weer aan de gebruiker van de telefoon waarop het Breakout

spel wordt gespeeld. Dit laat onder andere zien dat de precisie en realtime tracking mogelijkheden van de plugin en achterliggende UWB technologie nauwkeurig en accuraat genoeg is om de realtime locatie en beweging van de telefoon als input te kunnen gebruiken. De plugin en de voorbeeld applicaties leggen hiermee de basis voor Flutter developers voor het ontwikkelen van Flutter applicaties die gebruik willen maken van deze precisie en realtime tracking mogelijkheden. Voor een uitgebreide beschrijving van de uitwerking hiervan, een voorbeeld van de user interface en hoe de applicatie tot stand is gekomen, wordt er verwezen naar de hoofdstukken 3, 5.1, 5.3 en 5.3.3.

Tot slot kan de hoofdvraag beantwoord worden. De hoofdvraag luidt als volgt:

Hoe kan UWB technologie worden toegepast in mobiele apps, en specifiek in Flutter?

Om te achterhalen hoe UWB technologie kan worden toegepast in apps, en specifiek Flutter is er eerst onderzoek gedaan naar de werking van UWB. Hier is er gekeken naar welke kenmerkende eigenschappen UWB beschikt waardoor het de laatste jaren zijn weg heeft gevonden naar het mobiele platform. Vervolgens is er gekeken naar de huidige beschikbaarheid en de ondersteuning voor UWB op de platformen iOS en Android. Hier is onder andere terug te zien dat iOS het platform aanbiedt voor developers om applicaties te kunnen bouwen met UWB technologie op gebied van realtime locatiebepaling. Om te zien hoe dit precies wordt toegepast in een app is er een proof of concept gebouwd voor iOS. Hier is er kennis gemaakt met de UWB technologie en de frameworks die op dit moment worden aangeboden voor het mobiele platform. Ook hier is goed terug te zien dat UWB zich uitermate goed schikt voor applicaties die vertrouwen op location based services waarbij realtime locatie updates en precisie belangrijke rollen spelen. Hierna is er onderzoek verricht om te kijken hoe deze UWB technologie naar het Flutter platform kan worden gehaald. Hier is er onderzoek gedaan naar het opzetten van een Flutter plugin en is er aan de hand van de geboden architectuur en bijbehorende API een Flutter plugin ontwikkeld die ondersteuning biedt voor het gebruik van UWB op iOS apparaten en flexibel genoeg is om later andere platformen zoals Android toe te kunnen voegen. Verder is er een voorbeeld applicatie opgezet die het gebruik van de plugin weergeeft voor developers en hiermee laat zien wat de mogelijkheden van de plugin zijn. Kort samengevat leent UWB zich uitermate goed voor realtime locatiebepaling en is toe te passen in Flutter aan de hand van de geschreven Flutter plugin. Wel is er aan de hand van dit onderzoek goed terug te zien dat er een platform ontbreekt voor development mogelijkheden op gebied van Android. Maar met de stijgende populariteit van UWB is dit niet meer dan een kwestie van tijd. Tevens leent UWB zich op dit moment nog niet voor een verbinding tussen twee telefoons met verschillende operating systemen door de verschillende manieren waarop Android en iOS op dit moment een sessie opzetten. Ook, hoewel UWB zich uitermate goed leent voor location based services, is terug te zien dat UWB zich op dit moment álleen beperkt tot het aanbieden van een platform voor location based services. Een mobiel platform waarbij er op dit moment data uitgewisseld kan worden aan de hand van het versturen van UWB berichten laat op dit moment nog op zich wachten.

7 Aanbevelingen

In dit hoofdstuk worden de aanbevelingen gedaan. Hier wordt er onder andere gekeken naar de toevoeging van pigeon en een Android package, ondersteuning voor trackers en anchors, en de federated architecture 2.0.

7.1 Android package

Het is inmiddels geen verassing dat het toevoegen van een Android package aan de plugin in de aanbevelingen is terecht gekomen. Echter ligt er aan de hand van de gemaakte architectuur keuzes en API calls een goede basis om een Android package op een later tijdstip toe te kunnen voegen. Dit kan ook, zoals beschreven is in hoofdstuk 2.6.2, door een developer gebeuren die niet de auteur is van de plugin. Wel kan de auteur zoals voorheen aangegeven de package 'endorsen' zodat deze automatisch mee wordt geïmplementeerd bij het gebruiken van de plugin. Hoe een package toe te voegen is aan de plugin is te vinden in de readme file van de UWB plugin.

7.2 Pigeon

Pigeon is een tool die code genereert om communicatie tussen Flutter en het host platform type-safe, gemakkelijker en sneller te maken.[27] De methodchannel, zoals beschreven is in hoofdstuk 2.7, is niet type-safe. Het ontvangen en versturen van berichten van en naar de cliënt vanuit de host is afhankelijk van het declareren van dezelfde datatypen en argumenten om te kunnen werken. Hier kunnen dus 'relatief' makkelijk fouten in ontstaan. Hier biedt de Pigeon package een alternatief ten opzichte van het gebruik van de methodchannel. Pigeon genereert namelijk de benodigde code om berichten te kunnen versturen op een manier die type-safe is. Aan de hand van een pigeon.dart file, waarin de benodigde calls worden gedefinieerd, wordt er code gegenereerd aan zowel de native kant als de dart kant. Dit elimineert de benodigdheid om ervoor te zorgen dat de namen en datatypen van berichten overeenkomen tussen de dart en native kant. Meer hierover is te lezen in de volgende bron: [27].

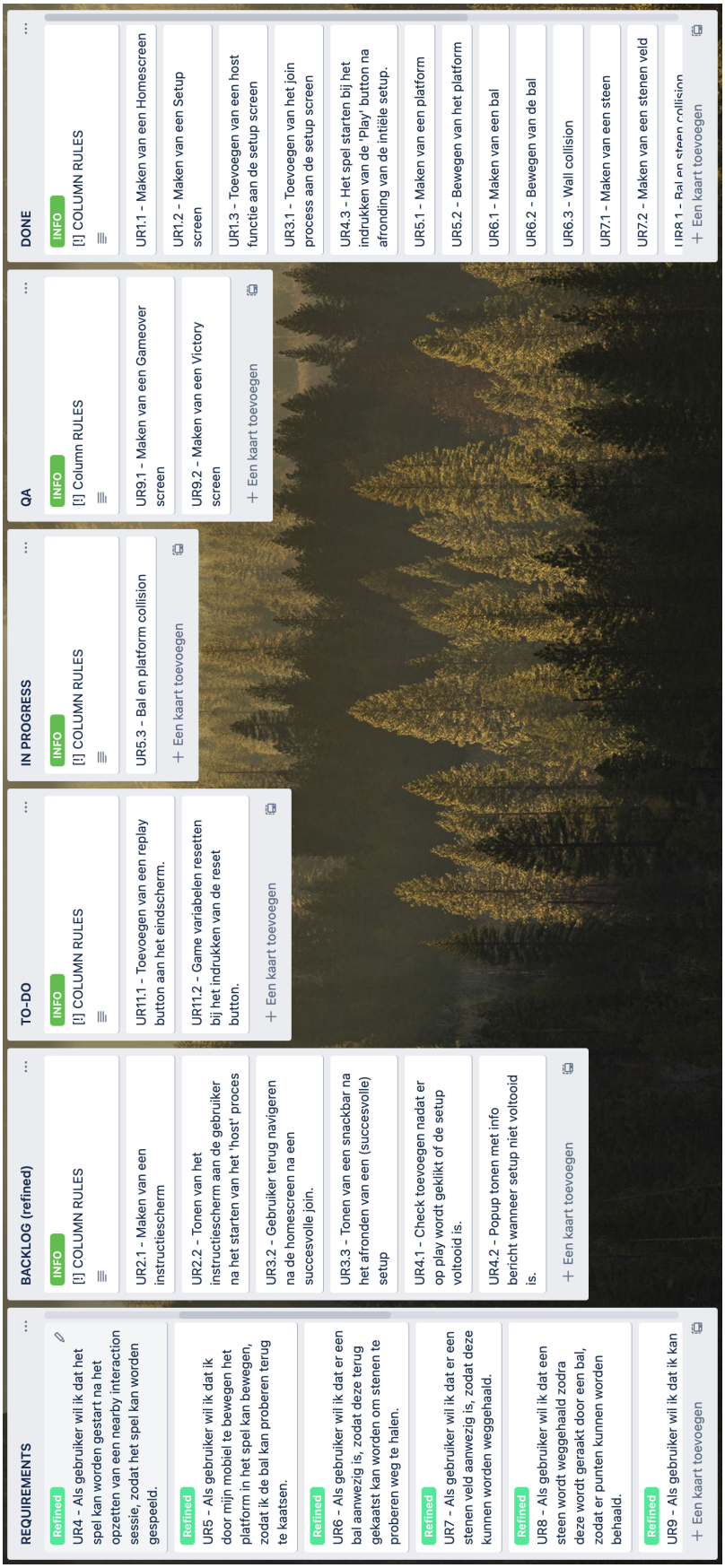
7.3 Trackers en anchors

Zoals te lezen is, is de UWB techniek op het mobiele platform op dit moment voornamelijk gebaseerd op locatiebepaling. De ondersteuning hiervan wordt langzaam steeds meer uitgebreid. Hoewel Apple ook beschikt over de Apple Airtag heeft Apple, in samenwerking met third parties, de mogelijkheid ontwikkeld voor developers om, al dan wel niet met de Apple Airtag, te werken met trackers en anchors op het iOS platform. De mogelijkheid bestaat namelijk al om developer kitjes te bestellen om uiteindelijk applicaties mee te realiseren die gebruik maken van onder andere trackers en/of anchors. Voorbeelden hiervan zijn onder andere de 'MK UWB Kit Mobile edition' [13] en de UWB development kit 'Murata Type2BP EVK'. [10]. De aanbeveling hier zou neerkomen op het onderzoeken van een developer kit, het gebruik ervan, en kijken of deze eventueel toegevoegd kan worden aan de plugin.

7.4 Federated 2.0

De plugin maakt op dit moment gebruik van een variatie op de federated architectuur zoals die beschreven 2.6.2. Echter wordt er op dit moment een gekeken naar een wat nieuwere federated architectuur die in dit verslag wordt benoemd als de 'federated 2.0' architectuur. Het idee van deze federated 2.0 architectuur is dat de logica zoveel mogelijk uit de native kant wordt gehaald en wordt verplaatst naar de dart kant. Hierbij zou er in dit geval per protocoll of extentie een Dart class komen die een wrapper is, oftewel een kopie vormt, van de protocoll of extentie class van de native kant. Deze worden dan elk vervolgens individueel gekoppeld met een methodchannel. Dit heeft als voordeel dat een Flutter developer zich meer kan focussen op het schrijven van dart code. En de calls naar de native kant niks meer doen als het aanroepen van een methode zoals die ook plaatvindt aan de dart kant. Door de logica te verplaatsen naar de Dart kant wordt er tevens ook makkelijker overzicht bewaard over de code gezien een developer niet continu hoeft te switchen tussen de verschillende packages.

8 Bijlagen



Figuur 19: Kanban bord

8.1 Overige breakout code

8.1.1 Host scherm onderdelen

8.1.1.1 Loading indicator

```
class LoadingIndicator extends StatelessWidget {

  final double x;
  final double y;
  final bool waitingForPeer;

  LoadingIndicator({required this.x, required this.y, required
this.waitingForPeer});

  @override
  Widget build(BuildContext context) {
    return
    !waitingForPeer ? Container() : Container(
      alignment: Alignment(x, y),
      child: const CircularProgressIndicator(
        valueColor: AlwaysStoppedAnimation<Color>(Colors.white),
      ),
    );
  }
}
```

8.1.1.2 Error Message

```
class ErrorMessage extends StatelessWidget {
  final double x;
  final double y;
  final String? error;

  ErrorMessage({required this.x, required this.y, required
this.error});

  @override
  Widget build(BuildContext context) {
    return
      error == null ? Container() : Container(
      alignment: Alignment(x, y),
      child: Text(
        error!,
        style: const TextStyle(color: Colors.red),
      ),
    );
  }
}
```

8.1.1.3 Arrow

```
class Arrow extends StatelessWidget {

  final double x;
  final double y;
  final double? angle;

  Arrow({required this.x, required this.y, required this.angle});

  @override
  Widget build(BuildContext context) {
    return
      angle == null ? Container() : Container(
```

```
        alignment: Alignment(x, y),
        child: Transform(
          transform: Matrix4.identity()..rotateX(1.8 * math.pi * 2),
          child: Transform.rotate(
            angle: angle!,
            child: const Icon(Icons.arrow_upward_rounded,
                color: Colors.white, size: 100),
          ),
        ),
      );
    }
  }
```

8.1.1.4 Dinstance

```
  class Distance extends StatelessWidget {
    final double x;
    final double y;
    final double? distance;

    Distance({required this.x, required this.y, required this.distance});

    @override
    Widget build(BuildContext context) {
      return
        distance == null ? Container() : Container(
        alignment: Alignment(x, y),
        child: Text(
          '${distance}m',
          style: const TextStyle(fontSize: 40, color: Colors.white),
        ),
      );
    }
  }
```

8.1.2 Game scherm onderdelen

8.1.2.1 Ball

```
  class Ball extends StatelessWidget {
    final double x;
    final double y;
    final GlobalKey ballKey;

    Ball({required this.x, required this.y, required this.ballKey});

    @override
    Widget build(BuildContext context) {
      return Container(
        alignment: Alignment(x, y),
        child: Container(
          key: ballKey,
          decoration:
              const BoxDecoration(shape: BoxShape.circle, color:
  Colors.white),
          width: 20,
          height: 20,
        ),
      );
    }
  }
```

8.1.2.2 Brick

```
class Brick extends StatelessWidget {
  final double x;
  final double y;
  Color color;
  final GlobalKey brickKey;

  Brick(
      {required this.x,
      required this.y,
      required this.color,
      required this.brickKey});

  @override
  Widget build(BuildContext context) {
    return Container(
      alignment: Alignment(x, y),
      child: ClipRRect(
        borderRadius: BorderRadius.circular(10),
        child: Container(
          key: brickKey,
          color: color,
          height: MediaQuery.of(context).size.height / 30,
          width: MediaQuery.of(context).size.width / 6,
        ),
      ),
    );
  }
}
```

8.1.2.3 Brick field

```
class BrickField extends StatelessWidget {
  final List<Brick> brickFieldList;

  BrickField({required this.brickFieldList});

  @override
  Widget build(BuildContext context) {
    return Stack(
      children: [
        for (var brick in brickFieldList)
          Brick(
              x: brick.x,
              y: brick.y,
              color: brick.color,
              brickKey: brick.brickKey)
      ],
    );
  }
}
```

8.1.3 Paddle

```
class Paddle extends StatelessWidget {
  final double x;
  final double y;

  Paddle({required this.x, required this.y});

  @override
  Widget build(BuildContext context) {
    return Container(
```

```
        alignment: Alignment(x, y),
        child: ClipRRect(
          borderRadius: BorderRadius.circular(10),
          child: Container(
            color: Colors.white,
            height: 20,
            width: MediaQuery.of(context).size.width / 5.5,
          ),
        ),
      );
    }
  }
```

8.1.3.1 Score

```
  class Score extends StatelessWidget {
    int score;
    bool gameHasStarted;

    Score({required this.score, required this.gameHasStarted});

    @override
    Widget build(BuildContext context) {
      return Container(
        alignment: const Alignment(0, 0),
        child: Text(
          gameHasStarted ? '$score' : '',
          style: const TextStyle(fontSize: 80, color: Colors.white),
        ),
      );
    }
  }
```

8.1.3.2 Screen overlay

```
  class ScreenOverlay extends StatelessWidget {
    final bool gameHasStarted;

    ScreenOverlay({required this.gameHasStarted});

    @override
    Widget build(BuildContext context) {
      return Container(
        alignment: const Alignment(0, -0.2),
        child: Text(
          gameHasStarted ? '' : 'T A P  T O  P L A Y',
          style: const TextStyle(color: Colors.white),
        ),
      );
    }
  }
```

8.1.3.3 isPLayerDead

```
  bool isPLayerDead() {
      if (ballY >= 1) {
        return true;
      }
      return false;
  }
```

8.1.3.4 toggleHorizontalMovement

```
void toggleHorizontalMovement() {
    setState(() {
      if (ballXDirection == direction.RIGHT) {
        ballXDirection = direction.LEFT;
      } else {
        ballXDirection = direction.RIGHT;
      }
    });
}
```

8.1.3.5 toggleVerticalMovement

```
void toggleVerticalMovement() {
    setState(() {
      if (ballYDirection == direction.DOWN) {
        ballYDirection = direction.UP;
      } else {
        ballYDirection = direction.DOWN;
      }
    });
}
```

8.1.3.6 wall Collision

```
void checkWallCollision() {
    setState(() {
      ///Vertical
      if (ballY >= 0.9) {
        ballYDirection = direction.UP;
      } else if (ballY <= -0.9) {
        ballYDirection = direction.DOWN;
      }

      ///horizontal
      if (ballX >= 1) {
        ballXDirection = direction.LEFT;
      } else if (ballX <= -1) {
        ballXDirection = direction.RIGHT;
      }
    });
}
```

8.1.3.7 Show Dialog

```
void _showDialog(String title, Color color) {
    showDialog<Dialog>(
      context: context,
      barrierDismissible: false,
      builder: (BuildContext context) {
        return AlertDialog(
          backgroundColor: color,
          shape: const RoundedRectangleBorder(
            borderRadius: BorderRadius.all(Radius.circular(10.0)),
          ),
          title: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Text(
                title,
                style: TextStyle(color: Colors.white),
              ),
```

```
            ],
          ), // To display the title it is optional
          content: Column(
            mainAxisSize: MainAxisSize.min,
            children: [
              Text(
                '\nYour score: $score',
                textAlign: TextAlign.center,
                style: const TextStyle(color: Colors.white),
              ),
            ],
          ),
          actions: [
            TextButton(
              onPressed: () {
                resetGame();
              },
              child: const Text('REPLAY'),
              style: TextButton.styleFrom(primary: Colors.white),
            ),
          ],
        );
      },
    );
}
```

8.1.3.8 Resetten van het spel

```
void resetGame() {
    Navigator.pop(context);
    brickFieldList.clear();
    brickFieldList = generateBrickFieldList();
    setState(() {
      gameHasStarted = false;
      playerIsDead = false;
      paddleX = 0;
      ballX = 0;
      ballY = 0;
      score = 0;
    });
}
```

8.1.3.9 Collision

```
RenderBox? ballBox =
        ballKey.currentContext?.findRenderObject() as RenderBox?;
    final Size? ballSize = ballBox?.size;
    final Offset? ballPosition = ballBox?.localToGlobal(Offset.zero);
    bool collide = false;

    double? ballPositionY = ballPosition?.dy;
    double? ballPositionX = ballPosition?.dx;
    double? ballSizeHeight = ballSize?.height;
    double? ballSizeWidth = ballSize?.width;

    for (Brick brick in brickFieldList) {
      RenderBox? brickBox =
          brick.brickKey.currentContext?.findRenderObject() as
RenderBox?;
      final Size? brickSize = brickBox?.size;
      final Offset? brickPosition =
brickBox?.localToGlobal(Offset.zero);

      double? brickPositionY = brickPosition?.dy;
```

```
        double? brickPositionX = brickPosition?.dx;
        double? brickSizeHeight = brickSize?.height;
        double? brickSizeWidth = brickSize?.width;

        if (ballPosition != null && brickPosition != null) {
          collide = ballPosition.dx < brickPosition.dx +
brickSize!.width &&
              ballPosition.dx + ballSize!.width > brickPosition.dx &&
              ballPosition.dy < brickPosition.dy + brickSize.height &&
              ballPosition.dy + ballSize.height > brickPosition.dy;

          if (collide) {
            setState(() {
              brickFieldList.remove(brick);
              score++;
              bool didToggle = false;

              if (ballPositionY! - (brickPositionY! + brickSizeHeight!)
< 1 &&
                      ballPositionY - (brickPositionY + brickSizeHeight)
> -8 ||
                  ballPositionY + ballSizeHeight! - brickPositionY < 8 &&
                      ballPositionY + ballSizeHeight - brickPositionY >
-1) {
                print('VERTICAL TOGGLE AT SCORE: $score');
                didToggle = true;
                toggleVerticalMovement();
              }

              if (ballPositionX! - (brickPositionX! + brickSizeWidth!) <
1 &&
                      ballPositionX - (brickPositionX + brickSizeWidth)
> -15 ||
                  ballPositionX + ballSizeWidth! - brickPositionX < 15 &&
                      ballPositionX + ballSizeWidth - brickPositionX >
-1) {
                toggleHorizontalMovement();
                didToggle = true;
                print('HORIZONTAL TOGGLE AT SCORE: $score');
              }
              if (!didToggle) {
                print('NO TOGGLE');
              }
            });
          } else {
            setState(() {
              brick.color = Colors.blueAccent;
            });
          }
        }
        if (collide) {
          break;
        }
      }
}
```

8.1.4 Overig

8.1.4.1 Main

```
void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
```

```
  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: Homescreen(),
    );
  }
}
```

8.1.4.2 Homescreen

```
class Homescreen extends StatelessWidget {
  const Homescreen({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.grey[900],
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            TextButton(
              onPressed: () {
                Navigator.push(
                    context,
                    MaterialPageRoute(
                        builder: (context) => const BreakoutGame()));
              },
              child: const Text(
                'P L A Y',
                style: TextStyle(color: Colors.white),
              ),
            ),
            TextButton(
              onPressed: () {
                Navigator.push(
                    context,
                    MaterialPageRoute(
                        builder: (context) => const HostScreen()));
              },
              child: const Text(
                'H O S T',
                style: TextStyle(color: Colors.white),
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

Referenties

[1] URL: https://stringfixer.com/nl/Ultra-wideband.

[2] Code formatting. URL: https://docs.flutter.dev/development/tools/formatting.

[3] Core/java/android/uwb - platform/frameworks/base - git at google. URL: https://android.googlesource.com/platform/frameworks/base/+/refs/heads/master/core/java/android/uwb/.

[4] Core/java/android/uwb/anglemeasurement.java - platform/frameworks/base - git at google. URL: https://android.googlesource.com/platform/frameworks/base/+/refs/heads/master/core/java/android/uwb/AngleMeasurement.java.

[5] Core/java/android/uwb/distancemeasurement.java - platform/frameworks/base - git at google. URL: https://android.googlesource.com/platform/frameworks/base/+/refs/heads/master/core/java/android/uwb/DistanceMeasurement.java.

[6] Core/java/android/uwb/rangingmanager.java - platform/frameworks/base - git at google. URL: https://android.googlesource.com/platform/frameworks/base/+/refs/heads/master/core/java/android/uwb/RangingManager.java.

[7] Core/java/android/uwb/uwbaddress.java - platform/frameworks/base - git at google. URL: https://android.googlesource.com/platform/frameworks/base/+/refs/heads/master/core/java/android/uwb/UwbAddress.java.

[8] Core/java/android/uwb/uwbmanager.java - platform/frameworks/base - git at google. URL: https://android.googlesource.com/platform/frameworks/base/+/refs/heads/master/core/java/android/uwb/UwbManager.java.

[9] Debugging flutter apps. URL: https://docs.flutter.dev/testing/debugging.

[10] Development kits for apple. URL: https://www.nxp.com/products/wireless/secure-ultra-wideband-uwb/secure-uwb-development-kits-that-interoperate-with-apple-u1:UWB_DEV_KITS?tid=vanUWB-Apple-U1.

[11] Ultra-wideband. URL: https://www.sciencedirect.com/topics/engineering/ultra-wideband.

[12] Uwb public api. URL: https://android-review.googlesource.com/c/platform/frameworks/support/+/1975487.

[13] Mk uwb kit mobile edition, Mar 2022. URL: https://www.themobileknowledge.com/product/mk-uwb-kit-mobile-edition/.

[14] Apple. Discoverytoken. URL: https://developer.apple.com/documentation/nearbyinteraction/nisession/3564775-discoverytoken.

[15] Apple. Iphone 11 - specificaties. URL: https://www.apple.com/nl/iphone-11/specs/.

[16] Apple. Multipeer connectivity. URL: https://developer.apple.com/documentation/multipeerconnectivity.

[17] Apple. Nisession. URL: https://developer.apple.com/documentation/nearbyinteraction/nisession.

[18] Apple. Nisession direction. URL: https://developer.apple.com/documentation/nearbyinteraction/ninearbyobject/3601347-direction.

[19] Apple. Nskeyedunarchiver. a decoder that restores data from an archive referenced by keys. URL: https://developer.apple.com/documentation/foundation/nskeyedunarchiver.

[20] Apple. Uiwindowscene - a scene that manages one or more windows for your app. URL: https://developer.apple.com/documentation/uikit/uiwindowscene.

[21] Apple. About ios 14 updates, Oct 2021. URL: https://support.apple.com/en-us/HT211808.

[22] Apple. How to use airdrop on your iphone, ipad, or ipod touch, Dec 2021. URL: https://support.apple.com/en-us/HT204144.

[23] Apple. Multiepeer connectivity framework, 2022. URL: https://developer.apple.com/documentation/multipeerconnectivity.

[24] Apple. Nearby interaction framework, 2022. URL: https://developer.apple.com/documentation/nearbyinteraction.

[25] Flutter. Findrenderobject method null safety. URL: https://api.flutter.dev/flutter/widgets/Element/findRenderObject.html.

[26] Flutter. Flutter plugins and packages. URL: https://docs.flutter.dev/development/packages-and-plugins/using-packages.

[27] Flutter. Pigeon: Dart package, Mar 2022. URL: https://pub.dev/packages/pigeon.

[28] Google. Developing packages and plugins. URL: https://docs.flutter.dev/development/packages-and-plugins/developing-packages.

[29] Google. Writing custom platform-specific code. URL: https://docs.flutter.dev/development/platform-integration/platform-channels.

[30] Itunews. Ultra-wide band (uwb) for the portable internet. URL: https://www.itu.int/itunews/manager/display.asp?lang=en&year=2004&issue=09&ipage=ultrawide&ext=html.

[31] KJ Kim. Samsung expects uwb to be one of the next big wireless technologies. URL: https://news.samsung.com/global/samsung-expects-uwb-to-be-one-of-the-next-big-wireless-technologies.

[32] Cynthia Peter. Understanding and using flutter packages. URL: https://www.educative.io/edpresso/understanding-and-using-flutter-packages.

[33] Arjun Sha, 8.6, 7.9, and 8.2. What is ultra wideband technology (uwb) - explained, Aug 2020. URL: https://beebom.com/ultra-wideband-technology-explained/.