Bachelorproef voorgedragen tot het behalen van de graad van bachelor in de informatica

BleiD: A platform for controlling IoT

Auteur:
Jeroen Ceyssens
1436995
3de bachelor Informatica

Coördinatoren:
Prof. dr. Kris Luyten
dr. Davy Vanacken
dhr. Sven Coppers

31 mei 2017

Dankwoord

Ik zou bij deze graag mijn woord van dank tonen aan de mensen die mij hebben gesteund in het realiseren van deze thesis. Mijn begeleider Sven Coppers heeft mij enorm goed begeleidt doorheen het project met goede advies en duidelijke feedback wanneer deze nodig was, waarvoor ik graag mijn dank aan hem wil betuigen. Verder wil ik graag mijn promotor Davy Vanacken en co-promotor Kris Luyten bedanken voor de concrete en duidelijke feedback doorheen het project en het adviseren van interessante en relevante technieken waar ik inspiratie uit kon opdoen.

Ik wil hierbij ook het hele onderwijsteam bedanken voor mij de technieken te leren en de mogelijkheid te geven om onderzoeken zoals deze uit te voeren. Als laatste zou ik ook graag mijn ouders willen bedanken voor mij de kans te geven deze studie te realiseren en voor de sterke steun die zij mij hebben gegeven in het verloop van mijn studieperiode.

Inhoudsopgave

Inleiding

In het dagelijkse leven maken mensen steeds vaker gebruik van digitale apparatuur. Deze apparatuur wordt gebruikt voor verschillende doeleinden zoals het maaien van gras of het maken van koffie. Voor gebruiksgemak en uitbreiding te garanderen, worden deze apparaten verbonden via een bepaalde vorm van communicatie (meestal draadloze communicatie). Deze communicatievormen zorgen ervoor dat gebruikers kunnen aangeven wat het apparaat moet doen, wat voor soort informatie het apparaat moet delen en hoe deze de informatie beschikbaar stelt. Voorbeelden van communicatievormen zijn onder andere: Bluetooth, Ethernet, Zigbee[1], ...

We noemen apparaten waarmee we kunnen verbinden via dit soort communicatievormen, IoT (*"Internet of Things"*) apparaten. Danzkij de grote versnelling in de evolutie van communicerende apparaten en technologie, zoals vermeld in het boek *"Digitising the Industry - Internet of Things Connecting the Physical, Digital and Virtual Worlds"* door Ovidiu Vermesan en Peter Friess[2], krijgen we een steeds grotere aanvraag naar dit soort IoT apparatuur. Juist omdat er zo een grote groei en variatie is aan IoT apparaten met verschillende communicatievormen, proberen we hier een standaard voor te ontwikkelen. Met deze standaard willen we ervoor zorgen dat we apparaten op eenzelfde, consistente manier kunnen aanspreken. Wanneer een nieuw apparaat beschikbaar is moeten we hier dus rechtstreeks mee kunnen interageren, zonder het moeten installeren van extra software en zonder het moeten aanpassen van de gebruikte hardware. Met andere woorden willen we apparaten zo veel mogelijk compatibel maken met elkaar.

Het is bijna onmogelijk dit te verwezenlijken voor alle apparaten in de wereld, aangezien deze allen een andere manier van werken hebben. Als we een standaard zouden stellen voor de communicatiemethode, dan zou al de huidige apparatuur in de wereld aangepast moeten worden naar deze standaard. Dit is fysiek een onmogelijke taak en is zeer kostelijk, dus is dit geen goede oplossing. Een andere mogelijke oplossing is al de toekomstige apparaten dezelfde vorm van communicatie te laten gebruiken. We spreken dan nog steeds over een periode van mogelijks decennia vooraleer al de apparaten wereldwijd hier gebruik van gaan maken, aangezien deze moet verspreiden over de hele wereld. We weten ook niet wat voor soort technologie in de toekomst komt en hoe lang onze "standaard" hierbij relevant zal blijven. Aangezien technologie continu

blijft evolueren, hebben we steeds een grotere noodzaak aan up-to-date protocollen voor communicatie die steeds betere snelheden, afstanden en services moeten ondersteunen. Een standaard behouden voor toekomstige apparaten is dus ook geen geldige oplossing voor zowel de communicatie tussen apparaten in het heden, als die in de toekomst.

Er bestaan al veel verschillende technieken die het interageren met IoT apparaten proberen te versimpelen. De techniek die wij toepassen bestaat uit het creëren van een abstractielaag tussen de gebruiker en zijn apparatuur in de vorm van een tussenliggend apparaat, de hub.

Hoofdstuk 1

Gerelateerde werk

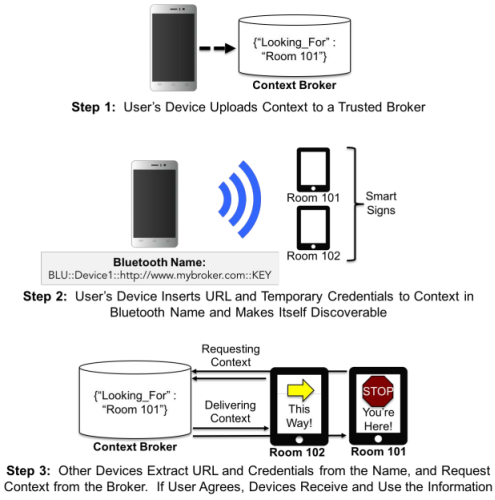
We bespreken hier enkele voorbeelden van technieken die, net zoals ons, een abstractielaag voor de interactie met IoT apparaten proberen te vormen. Niet al de vermelde technieken hebben volledig hetzelfde doel als ons en hebben als resultaat dus ook een totaal andere aanpak voor de abstractie. Toch is het belangrijk om deze verschillende technieken te bestuderen en vergelijken zodat we op ieder moment kunnen beslissen wat de beste oplossing is voor een bepaald probleem. Iedere techniek heeft ten slotte een toepassing waarop deze gespecialiseerd is en waar deze het meest geschikt voor is.

1.1 Bluewave

Dit is een techniek die toegepast wordt in omgevingen waarbij apparaten hun "context" moeten delen.[3] Normaal gesproken kost het een aardige hoeveelheid tijd om setups uit te voeren rond het delen van context. *Bluewave* voorziet een simpele manier om devices rechtstreeks context te laten sharen, zonder dat hiervoor uitgebreide installaties voor uitgevoerd moeten worden. In deze techniek wordt er gebruikt gemaakt van twee connecties, namelijk Bluetooth en Ethernet/Wi-Fi. Een typische toepassing van *Bluewave* is een situatie waarin de "omgeving" gegevens wilt verzamelen van de "gebruiker". Een voorbeeld hiervan is een omgeving met lichten die zijn kleur aanpassen afhankelijk van de positie van de mensen in de buurt. Bijvoorbeeld vijf mensen dichtbij het apparaat en de lampen worden rood, enkel mensen ver weg en de lampen worden groen.

De werking is als volgt (zie Figuur 1.1 voor representatie): *Bluewave* heeft een betrouwbare server waar het apparaat info over de gebruiker opslaat via een internetverbinding. Het apparaat van de gebruiker stelt deze info ter beschikking door zijn Bluetooth naam te veranderen naar een url, waardoor externe apparaten niet rechtstreeks hoeven te connecteren met de gebruiker. De url verwijst naar de betrouwbare server met extra parameters zoals een "device id" die vertelt welk apparaat opgezocht wordt, de contexten die opgevraagd worden en zo nog andere noodzakelijke elementen. De externe apparaten die de

context info van de gebruiker willen verzamelen, gaan via Ethernet of Wi-Fi de nodige info opvragen van de betrouwbare server via de gegeven url. Afhankelijk van de ontvangen info zullen de externe apparaten zich aanpassen en is de communicatie compleet.



Figuur 1.1: De algemene werking van *Bluewave* (Source: *"Bluewave: Enabling Opportunistic Context Sharing via Bluetooth Device Names"*[3])

1.2 iBeacon

Toen iOS 7 uitkwam had Apple hierbij een nieuwe technologie geïntroduceerd, namelijk de *iBeacon*[4]. Bij *iBeacon* wordt gebruik gemaakt van Bluetooth Low Energy (ook wel "Bluetooth Smart" genoemd) als connectie.[1] Het principe van *iBeacon* is de mogelijkheid bieden om een regio rond een apparaat (de "*beacon*") te creëren, waarbij externe iOS apparaten kunnen kijken of ze deze regio binnenkomen of verlaten (zie Figuur 1.2). Verder kunnen de externe iOS apparaten een schatting maken hoe ver ze van de "*beacon*" verwijderd zijn. Dit vormt nieuwe locatiemogelijkheden voor applicaties zonder dat deze de hardware moeten aanpassen of speciale software moeten installeren. Met andere woorden levert de *iBeacon* een abstractielaag voor een vorm van "location awareness" te delen met apparaten.

Bij Bluetooth LE worden advertenties met informatie over het apparaat verzonden naar andere externe apparaten in de buurt die daarna mogelijks willen verbinden. *iBeacon* verzendt de volgende informatie via deze advertenties[4]:

[1] Voor meer uitleg over Bluetooth LE zie sectie 4.1.1.

Figuur 1.2: Toestel in de buurt van een iBeacon regio (Source: *"Getting Started with iBeacon"*[4])

- **UUID (Universally Unique IDentifier)[5]:** Dit is een identifier die definieert welke applicatie gebruikt wordt door het apparaat die deze advertisement verzendt. De grootte van deze identifier is 16 bytes en blijft voor iedere locatie hetzelfde. Met andere woorden hoef je maar één keer een identifier te genereren. Deze identifier behoort tot het Bluetooth LE protocol, waarbij deze origineel gebruikt wordt voor de services te definiëren die beschikbaar worden gesteld.
- **Major:** Dit is een unieke identifier (met grootte 2 bytes) die helpt beslissen waar de *iBeacon* zich bevindt. Als voorbeeld kan de *iBeacon* in een subregio zitten van een groter geheel, deze subregio wordt dus gespecifeerd door de "Major".
- **Minor:** Dit is een unieke identifier (met grootte 2 bytes) die gebruikt kan worden voor een "Major" nog verder op te splitsen in aparte subregio's. Als resultaat krijgen we een verdeling van verschillende onderliggende regio's die gebruikt worden voor preciezere locaties te bepalen.

Via deze advertenties komen apparaten rechtstreeks te weten waar ze zich ongeveer bevinden (afhankelijk van de signaalsterkte), zonder dat ze moeten verbinden met de *iBeacon*. Applicaties hoeven hierdoor geen speciale methodes te voorzien om connecties te vormen en met services te interageren. Er kan direct van de advertenties gebruik gemaakt worden om consistent te blijven werken. Hiervoor moet enkel toestemming gevraagd worden aan de gebruiker om gebruik te mogen maken van *iBeacons*. Android heeft standaard geen *iBeacon* support, maar Android developers kunnen hier gebruik van maken via libraries[2] of door Bluetooth LE advertenties te ontcijferen.

[2]Voorbeeld van een library voor *IBeacon* support voor Android is *"AltBeacon"*[6]

BlueSentinel[7] is een voorbeeld van een applicatie die gebruik maakt van *iBeacon* voor communicatie. Het doel van *BlueSentinel* is een systeem te ontwikkelen dat kan controleren hoeveel mensen er in eenzelfde kamer zitten en om de bewegingen van deze mensen op te vangen. In deze applicatie passen ze wel het originele protocol van *iBeacon* aan zodat de "beacons" meerdere regio's gaan adverteren in de plaats van slechts één. Vanwege deze reden zijn er speciale "beacons" nodig die dit ondersteunen. Ook al moet er een speciale "beacon" extra geïmplementeerd worden, het origineel doel van het verzamelen van informatie zonder connecteren met de *IBeacon* blijft behouden.

1.3 Wi-fi Aware

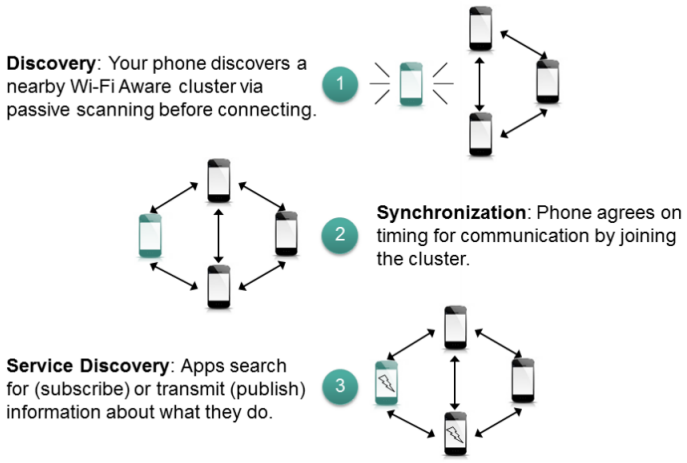
Dit is een nieuwe technologie gemaakt door de Wi-Fi Alliance[3] met als doel efficiëntere mobiele interacties tussen apparaten te ondersteunen zonder dat hiervoor een vast Wi-Fi netwerk nodig is.[8] De werking gaat als volgt (zie Figuur 1.3): Wi-Fi Aware maakt gebruik van een ad-hoc Wi-Fi netwerk tussen apparaten waarin gegevens ontvangen en uitgezonden worden. Een nieuw apparaat kan in een Wi-Fi Aware ad-hoc netwerk terecht komen door hier "passief" naar te scannen. Van zodra dit apparaat in het netwerk is gekomen, past het zijn signaalritme aan, aan die van het ad-hoc netwerk. Hierdoor kunnen apparaten continu zoeken naar andere apparaten en services zonder hiervoor constant te moeten ontvangen en zenden van gegevens (ze hoeven enkel nog maar op het ritme te zenden/ontvangen).

"Publish" en "Subscribe" berichten worden doorheen het netwerk verzonden door de apparaten, waarbij deze hun services aanbieden of zoeken naar andere services. Van zodra er een apparaat en service is gevonden waarmee geïnteragеerd wilt worden, wordt naar dit apparaat een notificatie gestuurd en wordt een aparte dataconnectie tussen de apparaten gestart. Deze apparaten kunnen dan met elkaar gegevens uitzenden via een ander protocol dan Wi-Fi Aware.

Een ad-hoc netwerk van apparaten wordt bij Wi-Fi Aware een "cluster" genoemd. In zo een cluster hebben verschillende apparaten een rol die ze moeten vervullen. De verschillende rollen in een Wi-Fi Aware cluster zijn:

- **Anchor Master:** Het apparaat met deze rol houdt de timing bij van het signaalritme van de cluster.
- **Master:** Zendt de "discovery" signalen van de cluster uit en synchroniseert de toegangspunten.
- **Sync:** Propageert de timing gegeven door de "Anchor Master" door naar de andere "Masters".

[3]Dit is een organisatie zonder winst die zich bezig houdt met het certificeren van Wi-apparaten, meer info over hun op: http://www.wi-fi.org/ .



Figuur 1.3: Standaard werking van Wi-Fi Aware (Source: *"Wi-Fi Aware™: Discover the World Nearby"*[8])

Deze rollen wisselen continu doorheen de apparaten zodat deze allemaal bijdrage leveren aan de cluster en zodat het stroom- en dataverbruik verdeeld wordt. Dankzij de gesynchroniseerde signaalritmes wordt het energieverbruik van Wi-Fi Aware in het algemeen beperkt. Verbindingen hoeven hierdoor ook niet continu open te staan zolang dat er geen aparte dataconnectie gestart is tussen twee apparaten. De aparte dataconnecties tussen apparaten zijn afhankelijk van de applicatie, maar zullen in de meeste gevallen peer-to-peer connecties zijn zoals Wi-Fi Direct[9]. Dit zorgt ervoor dat apparaten rechtstreeks met elkaar kunnen interageren zonder dat er andere tussenliggende apparaten verder nodig zijn.

Hoofdstuk 2

Scenario's

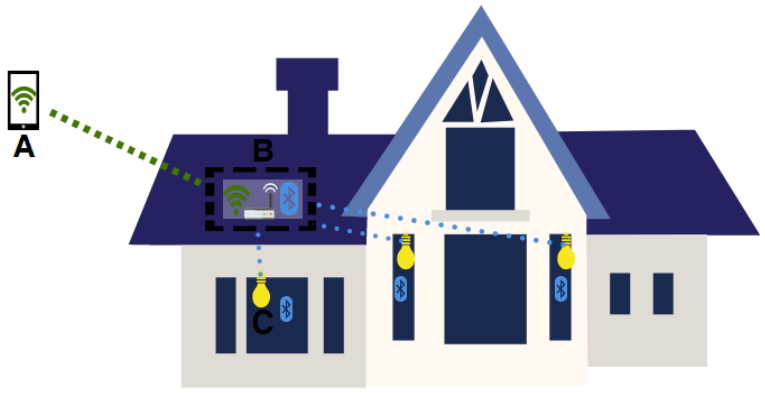
Ons systeem bestaat uit een hub met een Android applicatie, waarbij de hub verbindt met verschillende apparaten in de buurt en de smartphone met de applicatie verbinding maakt met de hub. De hub maakt gebruik van Bluetooth Low Energy[10] om te connecteren en interageren met de apparaten. De applicatie op de smartphone verbindt vervolgens via Wi-Fi met de hub en kan via deze de geconnecteerde apparaten aansturen. We hebben de hub de naam *BleiD* gegeven (staat kort voor "Bluetooth Low Energy Interaction Device") en de applicatie "BleiD Home". Om een beter idee te geven hoe *BleiD* met de applicatie gebruikt wordt als systeem, beschrijven we het volgende scenario.

2.1 Algemeen scenario

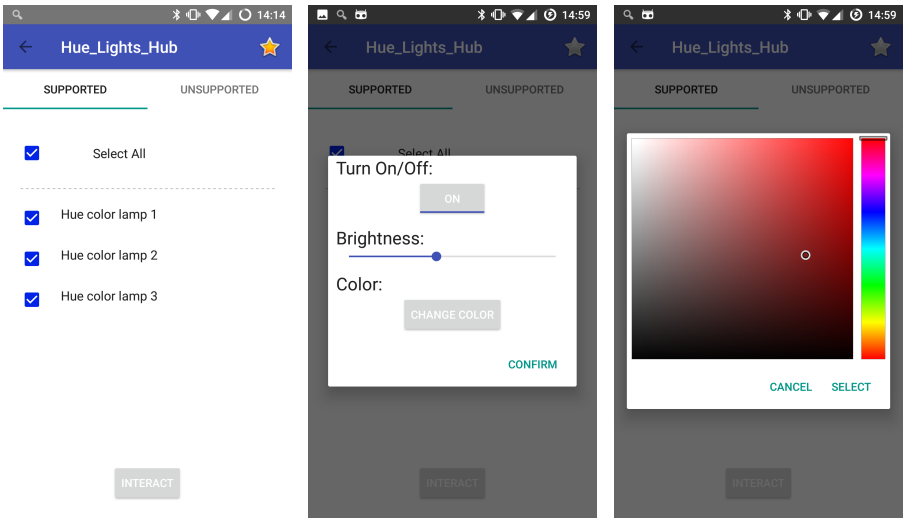
Initiatie De inwoner van een huis heeft een smartphone met Android. In het huis is er verlichting die bestuurd kan worden en waarbij ieder individueel licht van een andere firma kan zijn dan de rest van de verlichting. De inwoner moest voorgaand steeds apart met ieder licht verbinden voor deze te kunnen besturen en heeft vanwege deze reden een applicatie moeten installeren voor iedere firma waarvan deze verlichting heeft. Hij plaatst de *BleiD* ergens in het huis, geeft deze stroom en start het apparaat op.

Scenario De gebruiker komt binnen in het huis en zijn smartphone verbindt met het Wi-Fi netwerk. Hij krijgt een notificatie op zijn smartphone dat hij kan verbinden met de *BleiD* in het huis. De gebruiker geeft aan dat hij hiermee wilt verbinden waardoor een scherm op de smartphone wordt geopend. Op het scherm ziet hij de verschillende apparaten die de hub heeft gevonden, waarmee deze is verbonden en waarmee deze kan gaan communiceren. Het doel van onze gebruiker is om al de lichten op het benedenverdiep aan te zetten, dus selecteert hij al de lampen die hiertoe behoren. Hij geeft aan dat hij met deze lichten wilt interageren en klikt op de optie "Turn On", waardoor al de gekozen lichten aanspringen. De gebruiker heeft nu zijn doel bereikt en sluit de applicatie.

Dit scenario wordt getoond in Figuur 2.1 met protocollen die gebruikt zouden kunnen worden voor het scenario te laten werken. In het scenario wordt de applicatie door de gebruiker geïnstalleerd, waardoor deze al de eerder geïnstalleerde applicaties hierdoor kan gaan vervangen. Voor een voorbeeld van de interactie met de lichten op de applicatie zie Figuur 2.2.



Figuur 2.1: Voorstelling van *BleiD* bij een verlichtingsinstallatie in huis.



Figuur 2.2: Voorbeelden van interactie met lichten in een applicatie.

Buiten de verlichting heeft de gebruiker nog andere apparaten die hij zou willen besturen. Dit kunnen zowel zijn eigen apparaten zijn, als de apparaten die in de buurt komen van het huis. Voor hiermee rekening te houden biedt de hub een boekhoudingssysteem aan die de gebruiker de mogelijkheid geeft om al de

gevonden apparaten te beheren en te kiezen met welke deze wilt interageren. Het onderscheid tussen de verschillende apparaten moet duidelijk gemaakt worden aan de gebruiker zodat deze weet met welke apparaten hij kan interageren. Zelfs wanneer de gebruiker een grote hoeveelheid apparaten bezit blijft het systeem overzichtelijk en wordt de complexiteit voor de gebruiker beperkt gehouden. Voorbeelden van elementen in een boekhoudingssysteem zijn: Het bijhouden van een geschiedenis, de gebruiker notificeren van belangrijke updates, feedback geven over de staat van apparaten, ...

Er zijn twee soorten apparaten die gebruikt kunnen worden in ons systeem en in dit scenario. Apparaten die ondersteund zijn door de hub en apparaten die niet ondersteund zijn. De reden hiervoor ligt in het feit dat de hub geen support kan bieden voor alle soorten apparaten, waardoor de niet-ondersteunde apparaten anders behandeld moeten worden. Vanwege deze reden is het dus belangrijk om een duidelijk onderscheid te maken tussen deze twee gevallen.

2.2 Gebruik van ondersteunde apparaten

Scenario De gebruiker heeft in zijn huis een aantal apparaten die al zijn verbonden met de hub. Hij heeft net een nieuw apparaat gekocht voor een ouder apparaat in het huis te vervangen en installeert deze in het huis. De gebruiker wilt het nieuwe apparaat net zoals het oude apparaat gaan gebruiken via de hub. De gebruiker opent de Android applicatie ("BleiD Home") en ziet in een lijst het oude apparaat. Hij kiest ervoor om de hub het oude apparaat te laten vergeten en probeert de hub te laten connecteren met het nieuwe apparaat. Deze geeft aan dat er een apparaat losgekoppeld moet worden van de *BleiD,* omdat deze zijn maximale capaciteit heeft bereikt. De gebruiker kiest een eerder verbonden apparaat die losgekoppeld moet worden en vervolgens verbindt de hub met het nieuwe apparaat. Het apparaat wordt ondersteund door *BleiD*, waardoor hiermee rechtstreeks geïnterageerd kan worden zonder dat verdere setup nodig is. De gebruiker interageert met het apparaat via het menu van "BleiD Home" en sluit daarna de applicatie af. Het nieuwe apparaat heeft nu succesvol het oude apparaat vervangen en kan rechtstreeks gebruikt worden via de hub.

Voor ondersteunde apparaten biedt *BleiD* de mogelijkheid voor met deze apparaten te communiceren via een soort van "Plug and Play"[11]. Het is niet nodig om verdere setup uit te voeren aangezien het apparaat rechtstreeks werkt zodra deze is verbonden. Dit maakt het voor de gebruiker eenvoudiger om nieuwe apparaten te installeren en te gebruiken, aangezien deze niet hoeft te onderzoeken hoe het apparaat intern werkt en geen speciale software of hardware hiervoor moet installeren. .

Helaas geldt dit niet voor ieder apparaat omdat het onmogelijk is voor *BleiD* om alle soorten apparaten te ondersteunen. We hebben dus een alternatieve

aanpak voor de interactie nodig voor apparaten die niet ondersteund worden door de *BleiD*. Beiden soorten apparaten zijn van belang ten opzichte van de gebruiker zolang hij ze wilt gebruiken via de hub. Het is aan de *BleiD* om de mogelijkheid te bieden en aan de gebruiker om te kiezen of hij deze wilt gebruiken.

2.3 Gebruik van niet-ondersteunde apparaten

Scenario Net zoals het vorige scenario heeft de gebruiker een nieuw apparaat die hij wilt gaan gebruiken via de *BleiD*. Dit nieuwe apparaat wordt niet rechtstreeks door de *BleiD* ondersteund, maar heeft de protocollen nodig om deze te kunnen gebruiken via de hub. De gebruiker installeert het apparaat in het huis, opent de applicatie "BleiD Home" en zoekt in de lijst naar het apparaat. Hij selecteert het apparaat en geeft aan dat hiermee verbonden moet worden. Er opent een popup die aangeeft dat *BleiD* dit apparaat niet kent en vraagt de gebruiker of hij zelf de interactie wilt definiëren. De gebruiker geeft aan van wel en selecteert uit mogelijke opties hoe er met het apparaat omgegaan moet worden en dat de applicatie dit moet onthouden. Hij interageert met het apparaat en sluit vervolgens de applicatie. De gebruiker opent daarna de applicatie opnieuw om met hetzelfde apparaat te interageren. De vorige interactie wordt automatisch beschikbaar gesteld en de gebruiker gebruikt deze voor het apparaat te besturen, waarbij vervolgens de applicatie gesloten wordt.

Om de interactie met niet-ondersteunde apparaten toe te laten, moet de gebruiker zelf kunnen kiezen wat er met deze apparaten moet gebeuren. De reden waarom dit noodzakelijk is, komt door het feit dat de *BleiD* geen weet kan hebben van alle soorten apparaten. Nadat de gebruiker de interactie heeft gedefinieerd is het mogelijk deze opnieuw te gebruiken elke keer dat er met het apparaat geïnterageerd wilt worden. Dit zorgt ervoor dat het principe van "Plug and Play" in latere fases behouden blijft, aangezien de interactie niet meer hoeft aangepast te worden (maar wat wel nog gedaan kan worden).

We leggen in de komende hoofdstukken vooral de focus op de ondersteunde apparaten, zodat we zo veel mogelijk specifieke interactie voor de gebruiker zijn apparaten kunnen leveren.

Hoofdstuk 3

Implementatie Systeem

3.1 Opbouw

Voor een beter idee te krijgen over hoe onze implementatie is opgedeeld, zie Figuur 3.1. Zoals te zien op dit schema maken we gebruik van "Philips Hue Lights" voor de implementatie. Dit zijn speciale lampen, gecreëerd door Philips, die vrij bestuurd kunnen worden door gebruikers via Wi-Fi. Hiervoor worden standaard speciale applicaties/programma's voor gebruikt aangeboden door Philips.[1]

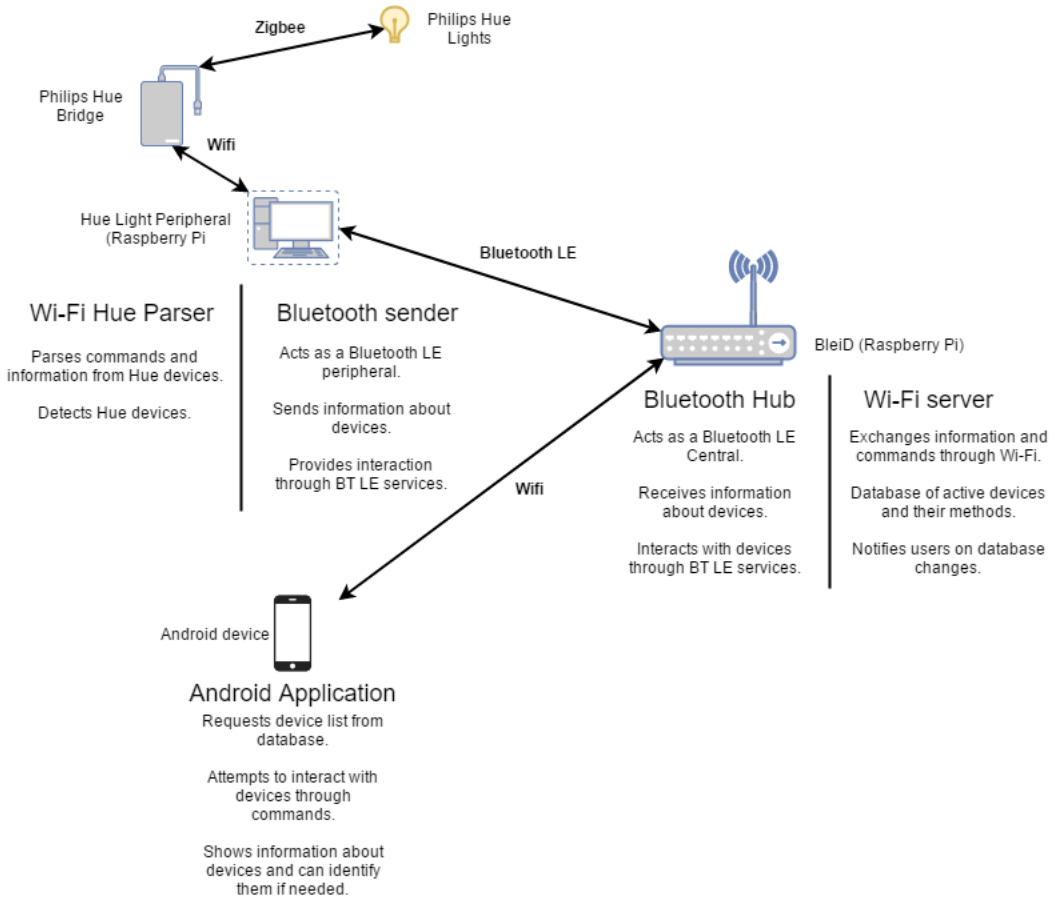
Met de "Philips Hue Lights" wordt niet rechtstreeks verbonden, maar via een tussenliggend apparaat (de "Hue Lights Peripheral") die met Bluetooth LE de interactie specifieert. De verbindingen doorheen het systeem verlopen als resultaat als volgt: De hub staat op het Wi-Fi/Ethernet netwerk als server en de applicatie verbindt hiermee als client via Wi-Fi. De hub scant verder ook als "central" via Bluetooth LE naar "peripherals" en kan hier verbinding mee maken.[2] De "Hue Lights Peripheral" stelt zichzelf als Bluetooth LE "peripheral" en biedt bepaalde services en kenmerken aan. De hub kan nu verbinden met dit apparaat waardoor gegevens gelezen en geschreven kunnen worden. De "Hue Lights Peripheral" is zelf verbonden als client met de "Philips Hue Bridge" via Wi-Fi, aangezien de bridge op het netwerk staat als server. Uiteindelijk is de "Philips Hue Bridge" verbonden met de verlichting via Zigbee[1] waar signalen over verstuurd worden.

Aangezien wij de verbinding en de interactie tussen de "Philips Hue Bridge" en de "Philips Hue Lights" niet hoeven te implementeren, bestaat de implementatie van het systeem over het algemeen uit drie delen:

1. De Android applicatie die verbinding maakt met de *BleiD;*
2. De *BleiD* die verbinding maakt met zowel de Android applicatie, als de "Hue Lights Peripheral";

[1] Meer info over Philips Hue Lights is te vinden op hun officiële website: http://www2.meethue.com

[2] Voor meer informatie over de werking van Bluetooth LE, zie sectie 4.1.1.



Figuur 3.1: Opbouw van de implementatie.

3. De "Hue Lights Peripheral" die zorgt voor de interactie met de "Philips Hue Bridge" en deze beschikbaar stelt via Bluetooth LE als peripheral.

Deze drie delen zijn aparte apparaten met elk een eigen implementatiemethode en structuur.

3.2 BleiD

3.2.1 Principe

We hebben de keuze om *BleiD* zelf te laten verbinden met willekeurige apparaten die deze detecteert, of om deze een interface te geven waarbij we kiezen met welke apparaten deze moet verbinden. Automatisch connecteren heeft het voordeel dat de gebruiker zelf niet veel moet interageren met het systeem, wat

de complexiteit voor de gebruiker vermindert. Het nadeel hierbij is dat we niet zeker weten welke apparaten verbonden zijn met de *BleiD*. Een interface daarentegen, heeft het voordeel dat we altijd kunnen kiezen welke apparaten verbonden moeten zijn. Dit geeft meer controle over het systeem, met als nadeel dat de gebruiker hierdoor meer zal moeten interageren. Toch kiezen we om gebruik te maken van de tweede optie, omdat we de gebruiker zo veel mogelijk controle willen geven. Verder vermijden we hiermee ook het risico dat geen enkel apparaat verbonden is die de gebruiker wilt gebruiken, maar wel willekeurige apparaten die hij niet nodig heeft.

De oorzaak van dit probleem is dat er een limiet is aan het aantal connecties die gelijktijdig geopend kunnen worden met Bluetooth LE. Deze limiet is afhankelijk van verschillende factoren, zoals bijvoorbeeld de hoeveelheid dataverkeer in de geopende connecties, en kan daardoor gaan variëren. Connecties kunnen dus best ook door de hub afgesloten worden nadat interactie door de gebruiker afgerond is.

We beschrijven het stappenplan voor de communicatie tussen de *BleiD* en de externe apparaten als volgt: *BleiD* blijft scannen naar nieuwe apparaten en onthoudt de apparaten die gevonden worden. Er tracht een verbinding gemaakt te worden met de nieuw gevonden apparaten om hun Bluetooth LE services en kenmerken te ontdekken. Als *BleiD* de services en kenmerken herkent, worden deze opgeslagen als ondersteunde services. Als *BleiD* deze niet herkent, worden deze opgeslagen als niet-ondersteunde services. Als de hub verbinding wilt maken met een extern apparaat, maar een interactie wordt ondertussen uitgevoerd door de gebruiker, dan wordt er gewacht tot deze is afgelopen vooraleer verder te gaan met het proberen verbinden. Als er een signaal voor interactie van de gebruiker komt, gaat *BleiD* met een of meerdere opgeslagen apparaten hun services communiceren. De apparaten waarmee gecommuniceerd wordt, zijn de apparaten die aangewezen zijn door de gebruiker voor interactie.

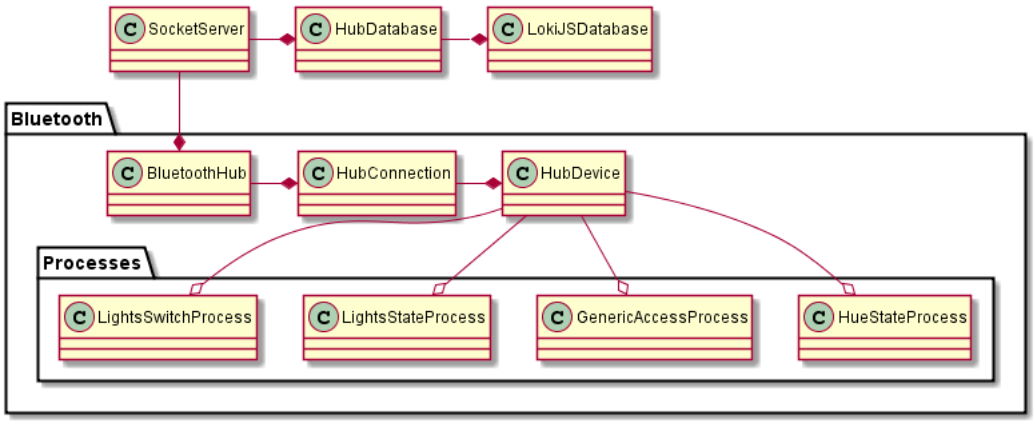
Voor te kunnen interageren met apparaten moeten deze eerst verbonden zijn, dus wordt er eerst geprobeerd te connecteren. Als op dit moment de limiet van het aantal open connecties is bereikt, zullen sommige verbindingen onderbroken moeten worden. Welke verbindingen dit zijn, wordt gekozen door de gebruiker. Zodra de verbindingen worden onderbroken, worden ze opgeslagen voor later gebruik. De gegevens van services en kenmerken worden gelezen en aangepast door ”read” en “write” requests te sturen naar de apparaten, afhankelijk van de gekozen interactie. Nadat de *BleiD* niet meer gebruikt wordt door de gebruiker, worden alle gemaakte verbindingen afgesloten en kan het scannen naar apparaten en het zoeken naar services/kenmerken hervat worden.

3.2.2 Implementatie *BleiD*

Voor de *BleiD* te representeren maken we gebruik van de Raspberry Pi 3[12], een microcontroller die standaard ondersteuning heeft voor zowel Wi-Fi als Bluetooth 4.1 (Bluetooth LE supported).[13] Voor het systeem van de *BleiD* te

implementeren maken we voor de programmeertaal gebruik van Node.js[14]. Node.js is eigenlijk "event-driven" Javascript wat meestal gebruikt wordt voor server code, maar ook gebruikt kan worden voor het maken van applicaties en software. De reden voor het kiezen van Node.js is vanwege zijn uitstekende package systeem die ons toelaat om snel support te bieden voor specifieke technieken en services.

Het package systeem dat Node.js gebruikt, genaamd "npm", is een van de grootste verzamelingen van open-source libraries wereldwijd.[15] Node.js developers kunnen vrij gebruik maken van deze libraries en tegelijkertijd hier zelf ook aan bijdragen. Een voorbeeld hiervan is "noble"[3] die wij gebruiken voor de Bluetooth Low Energy communicatie.



Figuur 3.2: Klassediagram *BleiD* Node.js implementatie.

Figuur 3.2 is een representatie van het klassendiagram van de *BleiD* Node.js implementatie. In dit klassediagram zijn de hoofdklasses van het programma vermeld en werken ze onderling als volgt: "SocketServer" is een klasse die een server opstart op het IP-adres van de Raspberry Pi en luistert op een bepaalde poort voor inkomende socket connecties. Deze klasse heeft een "BluetoothHub" instantie waarvan deze events opvangt die nieuwe gegevens bevatten en waarop deze methodes uitvoert, afhankelijk van de commando's verstuurd door de sockets. De nieuwe gegevens van de "BluetoothHub" worden verzonden naar de "HubDatabase", waar deze worden opgeslagen. De gegevens in de "HubDatabase" worden gebruikt om naar de sockets te verzenden zodra deze requests versturen. Deze data wordt opgeslagen in JSON-formaat[16] (= "Javascript Object Notation") in een "LokiJS"[4] database en wordt ook in JSON-formaat naar de

[3] Deze library laat apparaten scannen als central naar Bluetooth Low Energy peripherals en laat hun ook de interactie met deze apparaten implementeren. Link: https://www.npmjs.com/package/noble

[4] Dit is een database gebaseerd op "NoSQL" en staat als javascript package beschikbaar op npm, voor meer informatie hierover zie http://lokijs.org/#/.

sockets verzonden. Dit formaat wordt globaal ondersteund door verschillende programmeertalen, inclusief Java wat we gebruiken voor de Android applicatie. Dit is ook de algemene standaard voor Javascript (af te leiden uit de naam), dus is hier veel ondersteuning voor en laat ons toe consistent te werken. Voorbeelden van data's die opgeslagen worden in de "HubDatabase" zijn gegevens van gevonden apparaten, verbonden apparaten, ondersteunde en niet-ondersteunde services, ...

De "BluetoothHub" zoekt naar Bluetooth LE "peripherals" en geeft ze door aan de "HubConnection" zodra deze gevonden worden. Er wordt geprobeerd een verbinding te maken met de "peripherals" zodat hun services ontdekt kunnen worden. Zodra de services ontdekt zijn worden deze geanalyseerd, opgeslagen en worden naar kenmerken van deze services gezocht. Op ieder moment waarbij een nieuwe ontdekking wordt gedaan, wordt deze doorgegeven aan het hoger liggend systeem met behulp van de eerder genoemde events. De apparaten worden opgeslagen in "HubDevice" instanties, die elk hun eigen services hebben. Deze services zijn oftewel instanties van voorgedefinieerde klasses, of zijn enkel standaard JSON informatie. De voorgedefinieerde klasses stellen services voor die de hub specifiek ondersteunt.

De "Process" klassen stellen deze verschillende ondersteunde services van het systeem voor. Deze bevatten zowel de informatie over de service, als de informatie over zijn kenmerken. Het zijn namelijk de kenmerken die de verschillende soorten operaties over Bluetooth definiëren. Daarom is het belangrijk voor de kenmerken de juiste acties uit te laten voeren, aangezien deze niet altijd dezelfde operaties ondersteunen. De standaard operaties die gebruikt worden voor te interageren met Bluetooth kenmerken zijn "read" en "write", voor te lezen en schrijven van data.

We maken in totaal gebruik van vier Bluetooth LE services, waaronder drie die wij zelf gedefinieert hebben en één service die gespecifieert is door de Bluetooth SIG (Special Interest Group)[17]. De services die wij zelf hebben gedefinieerd zijn: "LightsState", "LightsSwitch" en "HueState". "LightsState" haalt informatie op over de verschillende Hue lichten (vooral "read" operaties) en "LightsSwitch" zorgt voor de interactie met deze lichten. "HueState" daarentegen, verzamelt informatie over de volledige staat van de "Hue Bridge". Deze informatie bevat bijvoorbeeld de lijst van verbonden lichten met hun respectievelijke namen en nummers. We hebben verder nog de service "GenericAccess", gedefinieert door de Bluetooth SIG. Als apparaten deze service ondersteunen laat het ons toe om apparaatinformatie te ontdekken, zoals een apparaat zijn interne naam.

3.3 Android Applicatie

3.3.1 Principe

De Android applicatie heeft twee hoofdfunctionaliteiten in het systeem: de gebruiker toelaten om met verschillende apparaten te interageren en het voorzien van een boekhoudingssysteem voor de hub. De interactie met de apparaten gebeurt volledig door commando's te versturen naar de server die de echte communicatie met de apparaten uitvoert. Het boekhoudingssysteem bestaat vooral uit de communicatie tussen de applicatie en de database van de server. De database op de server houdt verschillende gegevens over de apparaten bij, waarop de applicatie operaties uitvoert gespecifieerd door de gebruiker. Het boekhoudingssysteem op de applicatie bestaat onder meer uit: Notificaties over apparaten tonen, geschiedenis van apparaten bijhouden, het kunnen specifiëren van een type apparaat, een "watchlist" bijhouden van persoonlijke apparaten die voorrang krijgen, sorteerfuncties voor de verschillende lijsten en updates over de huidige staat van de hub.

Voor het interageren met de apparaten moet er onderscheid gemaakt worden tussen de ondersteunde en niet-ondersteunde apparaten. Beiden kunnen niet op dezelfde manier afgehandeld worden, aangezien bij ondersteunde apparaten veel meer functionaliteit kan voorzien worden dan voor niet-ondersteunde apparaten. Wij leggen in de implementatie de focus volledig op de ondersteunde apparaten, specifiek ons eigen geïmplementeerd apparaat.

3.3.2 Communicatie met de *BleiD*

We maken voor de communicatie tussen de Android applicatie en de *BleiD* gebruik van een andere communicatievorm dan tussen de *BleiD* en de externe apparaten, aangezien deze andere doeleinden moet voorzien. Bij deze communicatie maken we namelijk gebruik van Ethernet, of in het algemeen Wi-Fi (IEEE 802.11)[18]. Voor te communiceren via dit protocol hebben we de hub die op het thuisnetwerk is verbonden als server en de applicatie die gebruikt wordt als client. Van zodra dat het Android toestel is verbonden met het thuisnetwerk, kan deze met de hub verbinden op hetzelfde netwerk via zijn lokaal IP-adres en poortnummer. Als de Android applicatie verbonden is met een ander netwerk dan het thuisnetwerk, kan deze met de hub verbinden via zijn extern IP-adres in plaats van zijn lokaal IP-adres.

We definiëren de communicatie tussen de Android applicatie (client) en de *BleiD* (server) als volgt: Op de hub wordt er gewacht tot een nieuwe socket verbinding maakt met zijn IP-adres en poortnummer. De Android applicatie heeft de mogelijkheid tot het maken van een socket connectie met een server. De gebruiker geeft aan dat hij via de Android applicatie wilt verbinden met de hub via zijn IP-adres en poortnummer, waardoor er een socket connectie tracht gemaakt te worden. De hub ontdekt dat een apparaat wilt verbinden en accepteert de connectie. Een socket wordt op de hub en op de Android applicatie bijgehouden, waardoor de TCP connectie succesvol werd gecreëerd.

De Android applicatie kan nu gegevens van de *BleiD* aanvragen via de TCP connectie en deze weergeven in de applicatie. De gebruiker geeft aan dat hij met een van deze gegevens wilt interageren, waarop de Android applicatie een commando uitzend naar de server. De server voert het gewenste commando uit en verstuurt, afhankelijk van het commando, bepaalde feedback terug naar de client. Deze feedback wordt opnieuw door de applicatie verwerkt en getoond aan de gebruiker via de UI (User Interface) van de applicatie. Nadat de gebruiker klaar is met interageren met de applicatie, wordt de socket op beiden apparaten afgesloten en is de communicatie compleet.

3.3.3 Implementatie Android Applicatie

De Android applicatie is opgebouwd met behulp van Android Studio[5], een IDE ("Integrated Development Environement") gebaseerd op IntelliJ IDEA[6] en speciaal ontworpen voor het maken van Android applicaties. In deze IDE maken we gebruik van Java[19] als programmeertaal en XML om de UI (User Interface) van de applicatie te ontwerpen. In het algemeen kunnen UI elementen ook toegevoegd worden via Java code, maar de basisstructuur wordt standaard vastgelegd door XML. De applicatie bestaat uit vier hoofdschermen, namelijk de volgende: LoginActivity, BleiDHomeActivity, DeviceInfoActivity en DeviceInteractionActivity.

Een "Activity" is in Android een apart venster waarin functionaliteiten geïmplementeerd kunnen worden. Deze "Activities" kunnen zelf ook verschillende "Fragments" hebben die deelschermen van de "Activities" moeten voorstellen.

3.3.3.1 LoginActivity

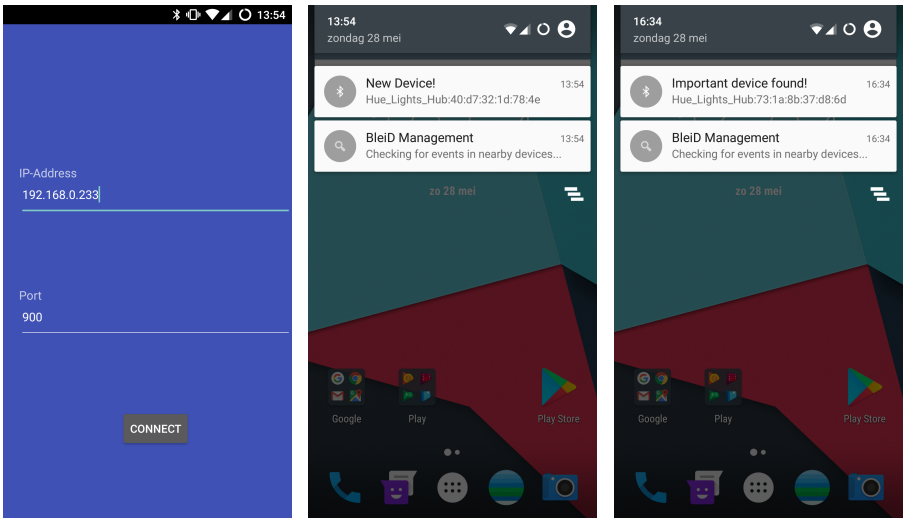
Dit is de eerste Activity die getoond wordt bij het openen van de applicatie. De gebruiker kan hier proberen te connecteren met een *BleiD* op het netwerk door middel van zijn IP-adres en poortnummer (zie Figuur 3.3). Om de communicatie te initialiseren wordt een socket aangemaakt en wordt een dialoogvenster geopend die wacht totdat een connectie gevormd is. Als er voor een lange tijd geen connectie gevormd kan worden zal deze terugkeren naar de LoginActivity en moet er een nieuwe poging gedaan worden. Van zodra een connectie gevormd is, wordt een nieuwe BleiDHomeActivity gestart en wordt de LoginActivity bewaart op de Android "Activity Stack"[7]. Bij het vormen van een succesvolle connectie met de server/hub, wordt ook een background service opgestart die luistert naar updates verstuurt door de server. Deze updates worden getoond in de vorm van notificaties (zie Figuur 22) aan de gebruiker. De background service blijft actief doorgaan zelfs wanneer de applicatie is afgesloten. Dit zodat

[5] https://developer.android.com/studio/index.html

[6] https://www.jetbrains.com/idea/

[7] Dit is een stack die Activities die eerder bezocht waren bewaard zodat deze later terug geopend kunnen worden. Meer info: https://developer.android.com/guide/components/activities/tasks-and-back-stack.html

de gebruiker altijd op de hoogte is van wijzigingen en de applicatie direct kan starten wanneer nodig.



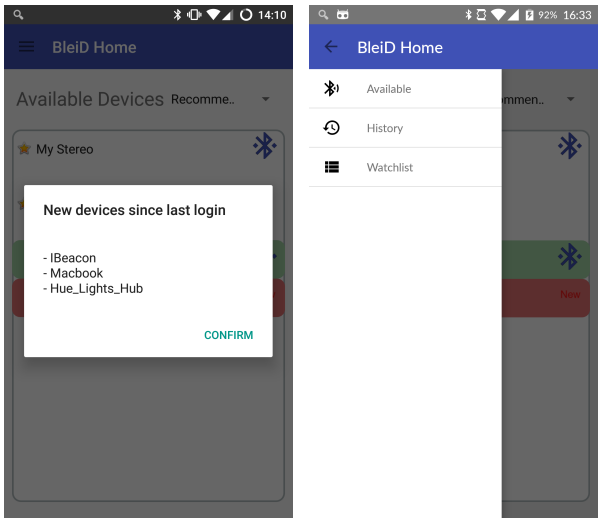
Figuur 3.3: Applicatie screenshots van de LoginActivity en verschillende notificaties.

3.3.3.2 BleiDHomeActivity

Bji het openen van deze Activity wordt aan de server gevraagd welke apparaten gevonden zijn, welke verbonden zijn en welke nieuw zijn bijgekomen. Om de tien seconden en bij iedere interactie wordt een update gevraagd van de server zodat de UI op ieder moment de juiste informatie toont. De applicatie ontvangt deze informatie en toont eerst een melding met al de nieuwe apparaten sinds de laatste login (zie Figuur 3.4). Deze "Activity" heeft verder ook een NavigationDrawer die wisselt tussen verschillende "Fragments" met ieder een eigen functionaliteit. De "Fragments" die wij hebben zijn (zie Figuur 23): "available devices", "history" en "watchlist". De "Activity" houdt de connectie met de server bij zodat de "Fragments" hier geen weet van hoeven te hebben en zodat dezelfde connectie behouden kan worden over de verschillende schermen.

Bij het openen van een nieuwe "Activity" wordt ook hier de BleiDHomeActivity tijdelijk opgeslagen op de Android "Activity Stack" zodat hier later naar teruggekeerd kan worden. De timer om een update van de server aan te vragen iedere tien seconden, wordt tijdelijk stopgezet tot de "Activity" weer opnieuw getoond wordt.

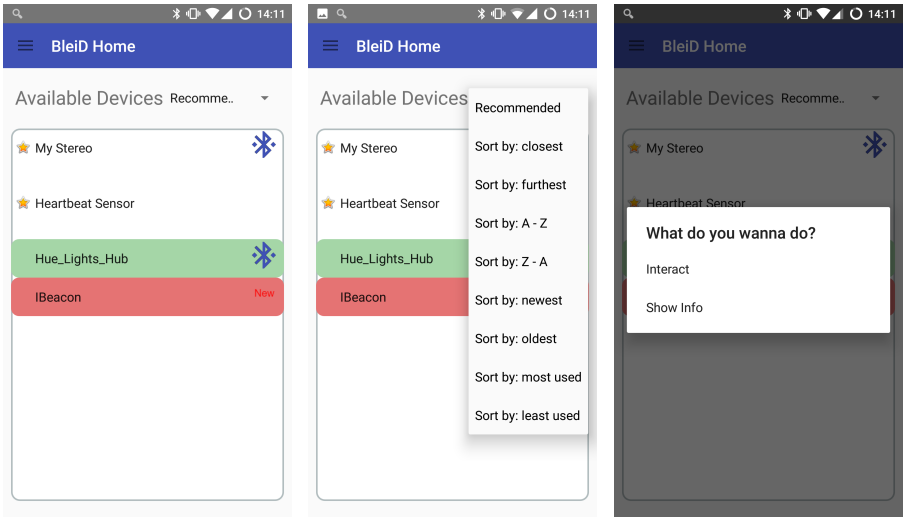
Available Devices Deze "Fragment" heeft een lijst van beschikbare apparaten (zie Figuur 24) die meegegeven wordt door de "Activity" die verbinding heeft met de server. Op de lijst van beschikbare apparaten wordt in het groen



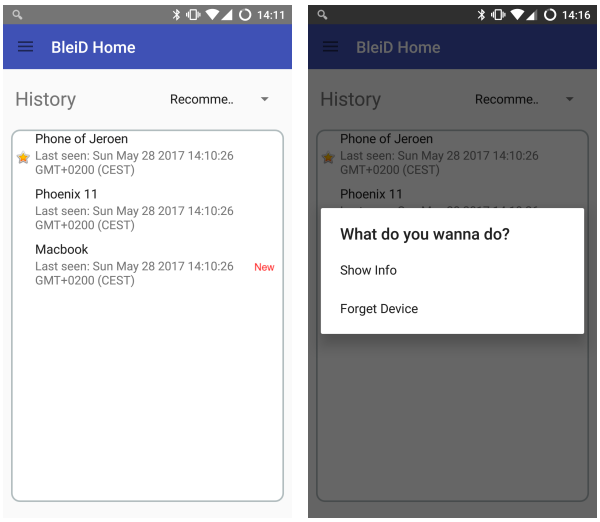
Figuur 3.4: Applicatie screenshot van de BleiDHomeActivity "nieuwe apparaten" en de navigation drawer.

getoond welke apparaten specifiek ondersteund zijn en in het rood met welke niet geconnecteerd (en dus ook niet geïnterageerd) kan worden. Verder geeft het sterretje aan welke apparaten de gebruiker in zijn "watchlist" heeft en het bluetooth teken met welke apparaten op dit moment verbonden zijn. De "New" achteraan een apparaat geeft aan welke dat er nieuw zijn sinds de laatste keer dat de gebruiker de applicatie heeft geopend. Verder kan er gesorteerd worden op de apparaten waarbij "Recommended" de apparaten in de "watchlist" vanboven plaatst, gevolgd door de apparaten waarmee het meest geïnterageerd zijn (een gebruiker zou die sneller opnieuw willen gebruiken). De gebruiker kan een apparaat aanklikken om hiermee oftewel te beginnen interageren ("DeviceInteractionActivity"), oftewel de informatie over het apparaat te bekijken en aan te passen ("DeviceInfoActivity").

History Deze "Fragment" biedt een geschiedenis van apparaten die eerder gevonden waren, maar nu niet meer in de buurt zijn (zie Figuur 24). Ook hier geeft een sterretje aan welke apparaten op de "watchlist" staan, de "New" welke apparaten dat nieuw zijn sinds de laatste login en kan er gesorteerd worden zoals bij de "available devices". Verder wordt bij ieder apparaat vermeld wanneer deze voor het laatst is gezien zodat de gebruiker een idee heeft van wanneer deze weg zijn gegaan. In de apparaten zit de laatst bekende "rssi" opgeslagen zodat er gesorteerd kan worden op degene die het dichtst bij waren gekomen toen ze het laatst waren gezien. De gebruiker heeft de mogelijkheid om oftewel de informatie te bekijken over het apparaat ("DeviceInfoActivity"), oftewel voor dit apparaat te vergeten (verwijderen uit de geschiedenis en de "watchlist").

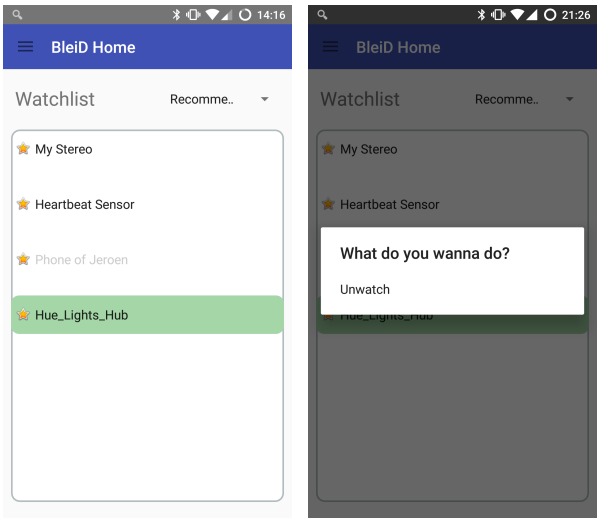


Figuur 3.5: De verschillende functies aangeboden door de ”available devices” Fragment.



Figuur 3.6: De verschillende functies aangeboden door de ”history” Fragment.

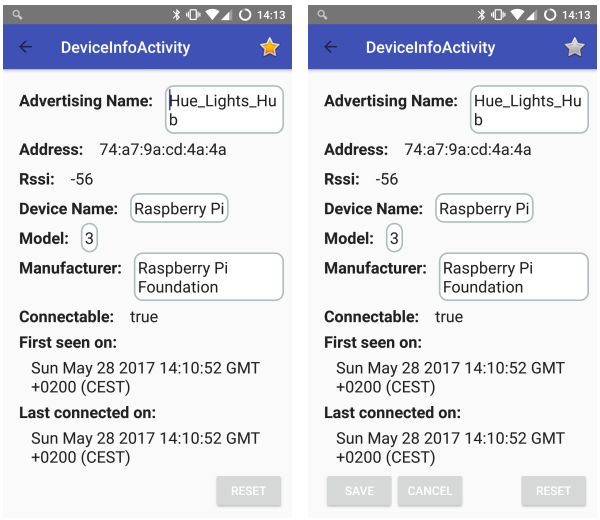
Watchlist In deze "Fragment" worden de verschillende apparaten getoond die de gebruiker als belangrijk ziet (zie Figuur 25). De apparaten met een zwarte naam zijn degene die gevonden zijn en die met een lichtgrijze zijn de apparaten die niet meer in de buurt zijn, maar wel nog in de geschiedenis staan. Ook hier wordt een groene kleur gegeven aan apparaten die specifiek ondersteund zijn door de hub. De gebruiker heeft de mogelijkheid om het apparaat niet meer te volgen en deze zo rechtstreeks uit de "watchlist" te halen.



Figuur 3.7: De verschillende functies aangeboden door de "watchlist" Fragment.

3.3.3.3 DeviceInfoActivity

Bij het openen van deze "Activity" wordt een "DeviceData" object meegegeven waaruit deze gegevens kan lezen. De informatie over het gekozen apparaat wordt getoond in de respectievelijke (zie Figuur 3.8) en sommige velden kunnen ook aangepast worden door de gebruiker. In het geval dat de hub dus niet de juiste waardes heeft kunnen ophalen, kan de gebruiker deze zelf kiezen. Zodra er een aanpassing wordt gedaan worden de knoppen "Save" en "Cancel" getoond die de nieuwe gegevens doorgeven aan de server of de aanpassing annuleren. Door het klikken op de "Reset" knop, worden alle wijzigingen aan dit apparaat weggegooid en worden de oude waardes bepaalt door de hub terug getoond. De ster rechtsboven geeft aan of dit apparaat gevolgd wordt door de gebruiker en of dat deze in de "watchlist" staat. Klikken op de ster laat de gebruiker het apparaat volgen of onvolgen zodat de gebruiker hiervoor niet naar de "watchlist" hoeft te gaan.



Figuur 3.8: Applicatie screenshot van DeviceInfoActivity

3.3.3.4 DeviceInteractionActivity

Deze "Activity" stelt de informatie over een bepaald apparaat voor. We maken in deze Activity gebruik van "Fragments", omdat we hier een TabBarLayout gecombineerd met een ViewPager hebben. Deze geven de mogelijkheid om te wisselen tussen verschillende vensters via een navigatie tabbar of door naar links of rechts te vegen over het scherm. De voorwaarde om hier gebruik van te kunnen maken, is dat al de vensters waartussen gewisseld mag worden, "Fragments" zijn. We hebben de volgende Fragments in deze Activity: SupportedFragment en UnsupportedFragment.

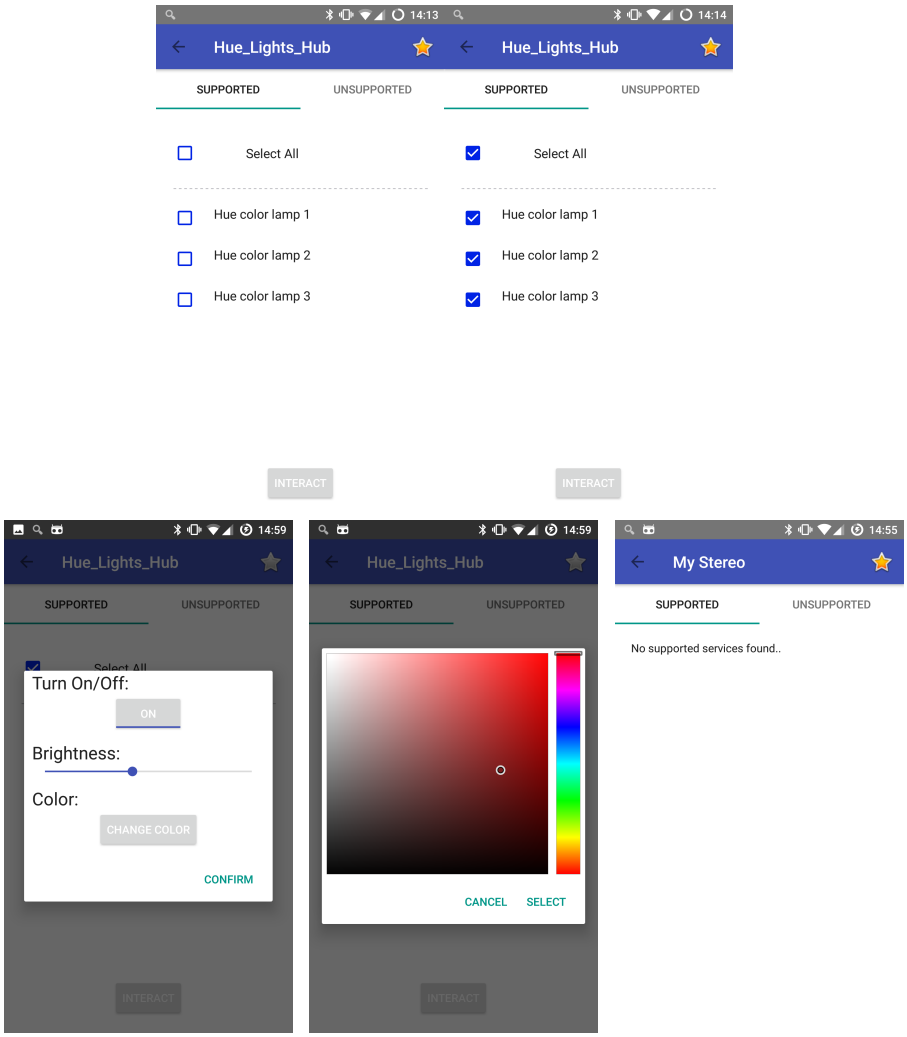
SupportedFragment Deze Fragment is enkel bruikbaar als het gekozen apparaat, de services van de *BleiD* bezit. Dit betekent dat de services door ons systeem ondersteund zijn en dat hiermee geïnterageerd kan worden (zie Figuur 3.9, eerste/tweede venster). In ons geval hebben we ondersteuning voorzien voor onze eigen Bluetooth LE protocollen, waarbij we met "Philips Hue Lights" kunnen interageren. Er kan gekozen worden om deze lampen rechtstreeks aan/uit te zetten of deze te selecteren en meer geavanceerde opties uit te voeren. Dit wordt gedaan door op de "Interact" knop te klikken, waardoor een nieuw dialoog tevoorschijn komt (zie Figuur 3.9, derde venster).

In dit dialoog is het mogelijk om verschillende acties uit te voeren voor de geselecteerde lampen. Al de interacties worden rechtstreeks naar de server verstuurd in de vorm van commando's, waarop deze de juiste acties uitvoert. Voor het kiezen van de kleur van de lichten is een apart dialoog voorzien met een Colorpicker[8] (zie Figuur 3.9, vierde venster). Zodra geklikt wordt op de

[8]Deze is niet zelf geïmplementeerd, maar hier wordt de volgende library voor gebruikt:

"Select" optie wordt een signaal verstuurd naar de server en wordt de kleur van de lichten aangepast. We hebben ervoor gekozen om de lampen niet rechtstreeks aan te passen iedere keer dat er over een kleur heen wordt gegaan, omdat dit een grote hoeveelheid signalen over het Wi-Fi netwerk en de Bluetooth LE connectie verzendt.

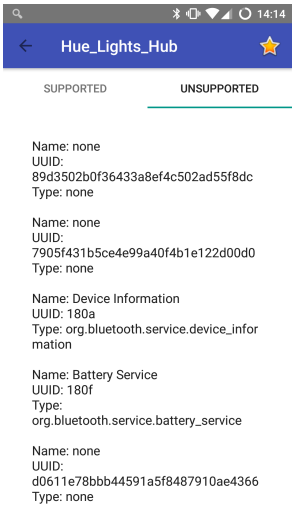
Als het apparaat geen supported services heeft, wordt in de plaats het scherm getoond op Figuur 3.9, vijfde venster.



Figuur 3.9: Applicatie screenshots van SupportedFragment met dialogen in DeviceInteractionActivity.

https://github.com/danielnilsson9/color-picker-view

UnsupportedFragment Deze Fragment toont de services die het verbonden apparaat aanbiedt, maar die niet ondersteund zijn door ons systeem (zie Figuur 3.10). Dit is de laatste tab in onze Activity en ook deze informatie wordt up to date gehouden via de server. We zouden deze Fragment eventueel nog kunnen gebruiken voor de interactie met niet-ondersteunde services te definiëren in latere stadia van het onderzoek.



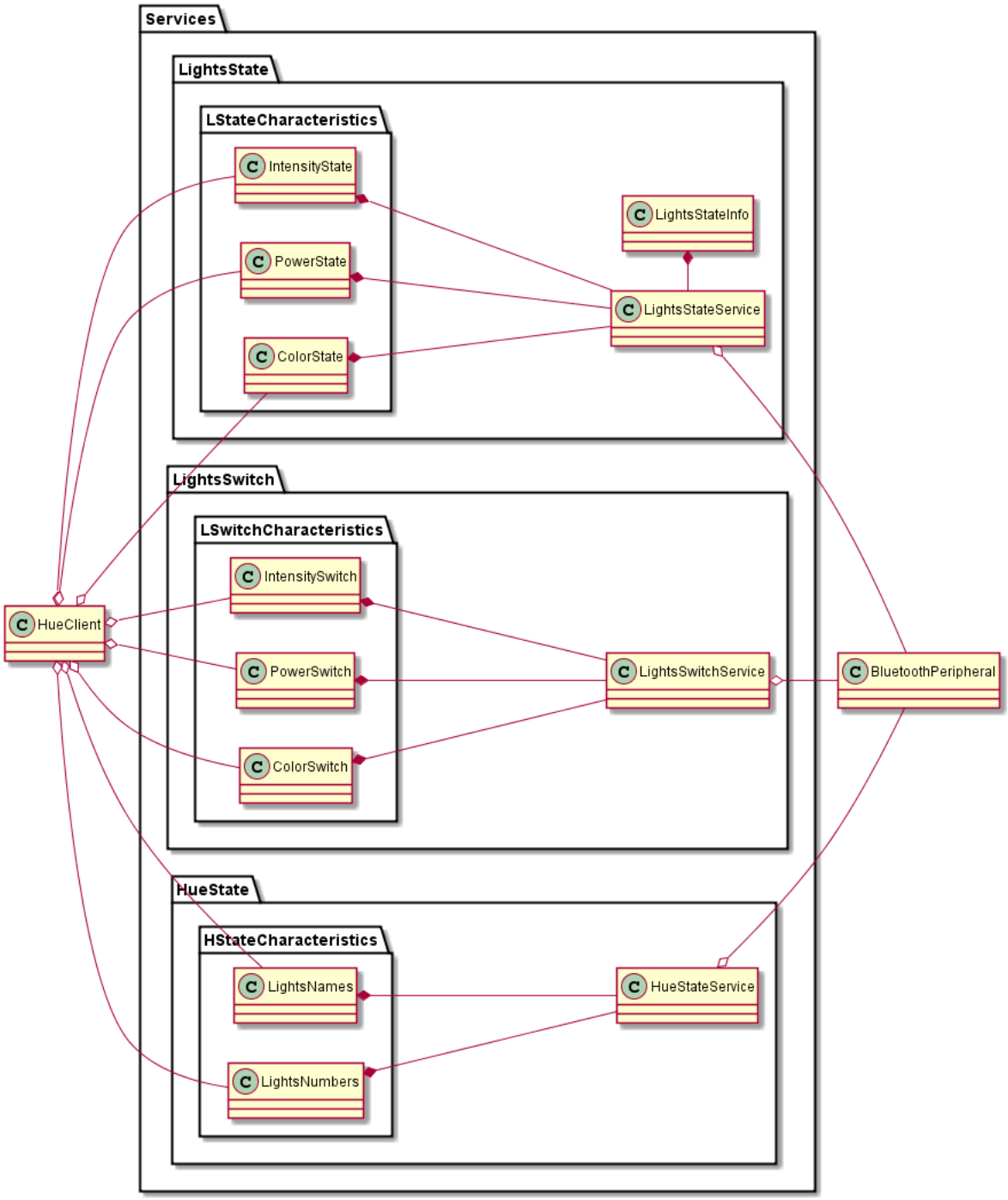
Figuur 3.10: Applicatie screenshot van UnsupportedFragment in DeviceInteractionActivity.

3.4 Hue Light Peripheral

We maken voor dit apparaat ook gebruik van een Raspberry Pi 3 zoals voor de *BleiD*, aangezien deze zowel Bluetooth LE als Wi-Fi ondersteuning aanbiedt. We hebben Wi-Fi nodig voor te kunnen communiceren met de "Philips Hue Bridge", aangezien deze zichzelf beschikbaar stelt op het netwerk als server. Voor developers biedt Philips de "Hue API" aan, waarmee ze zelf programma's en applicaties kunnen schrijven voor te interageren met de verlichting. De data over de "Philips Hue Light" wordt vrijgegeven in JSON-formaat, zodat de developers ze eenvoudig kunnen aanspreken.

Wij maken voor dit apparaat gebruik van Node.js als programmeertaal, aangezien deze zowel een "Bluetooth Low Energy Peripheral library" ondersteunt ("bleno"[9]), als libraries gebaseerd op de "Hue API". De library die wij gebruiken voor te interageren met de "Philips Hue Lights" is de "node-hue-api" package[10].

[9]Is gemaakt door hetzelfde team als de "noble" library vermeld in sectie 3.2.2. Link:



Figuur 3.11: Klassediagram Hue Light Peripheral implementatie.

Op Figuur 3.11 zien we het klassediagram met de hoofdklassen van de "Hue Light Peripheral". Het programma start met de "BluetoothPeripheral" die de "bleno" package gebruikt voor zichzelf te adverteren via Bluetooth LE. In deze klasse worden de verschillende services aangemaakt die we eerder ook zagen in de *BleiD* implementatie, namelijk de "LightsStateService", de LightsSwitchService" en de "HueStateService". We adverteren deze services samen met de advertentie-signalen van het apparaat, door hun UUID mee te geven aan de paketten. Iedere service heeft bepaalde kenmerken die interactie met de "HueClient" aanbieden.

Er is maar één "HueClient" instantie doorheen het hele systeem, aangezien wij één vaste connectie vormen met de "Philips Hue Bridge" waar signalen doorheen gestuurd worden. Ieder kenmerk maakt dus met andere woorden gebruik van dezelfde "HueClient" instantie voor hun acties uit te voeren.

We kunnen de kenmerken groeperen aan de hand van hun bovenliggende service zodat we in totaal drie groepen hebben (zie "Characteristics" in Figuur 3.11). De eerste groep behorend tot de "LightStateService" heeft de volgende kenmerken: "IntensityState", "ColorState" en "PowerState". Deze drie kenmerken geven de mogelijkheid aan centrals voor informatie over een bepaald licht te ontdekken, namelijk de intensiteit, de kleur en of het apparaat aan of uit staat.

De groep van de "LightSwitchService" bestaat uit dezelfde drie kenmerken, maar in de plaats van "States" zijn het "Switches". De reden hiertoe is dat deze kenmerken niet de staat over een licht verzamelen, maar deze aanpassen. Voor de kenmerken te gebruiken moeten de Bluetooth LE berichten in het juiste formaat zijn, aangezien in dit formaat gespecifieerd wordt welk licht moet aangepast worden en wat deze zijn nieuwe waardes zijn. Het is dus niet dat ieder apparaat de "Philips Hue Lights" kan gaan aanpassen zonder kennis van de verwachtte berichten.

De laatste groep behorende tot de "HueStateService" zijn slechts twee kenmerken, namelijk de "LightsNames" en de "LightsNumbers". Zoals de naam suggereert geven deze de mogelijkheid om de nummers en namen van de verbonden lichten op te vragen. Dit is globale informatie over de "Hue Bridge" zelf, aangezien hier op ieder moment meerdere lichten aan gekoppeld kunnen worden via Zigbee.

---

https://www.npmjs.com/package/bleno

[10] Dit is ook de meest geavanceerde Node.js library voor de "Hue API". Link: https://www.npmjs.com/package/node-hue-api

Hoofdstuk 4

Gebruikte technieken

We hebben in ons systeem gebruik gemaakt van verschillende communicatievormen. We maakten tussen de Android applicatie en de hub gebruik van Wi-Fi/Ethernet en voor de communicatie tussen de hub en de externe apparaten van Bluetooth LE. Er zijn verschillende redenen waarom we gekozen hebben voor deze manier van werken. We beschrijven in dit hoofdstuk eerst hoe dat de protocollen die wij gebruiken in zijn werk gaan, waarop we ze vergelijken met andere soort protocollen. We hebben deze vergelijking gebruikt in het verloop van het onderzoek om te beslissen welke protocollen we gingen toepassen.

4.1 Gebruikte communicatievormen

4.1.1 Bluetooth Low Energy

Doorheen de jaren heeft Bluetooth al verschillende versies gekend.[17] "Bluetooth Low Energy " (ook wel "Bluetooth Smart" genoemd) is een implementatie van Bluetooth die beschikbaar is voor gebruik vanaf de komst van Bluetooth versie 4.0. Andere versies die dit protocol ondersteunen zijn Bluetooth versies 4.1, 4.2 en 5.0 (dit is de laatste nieuwe versie).[10][13] Bluetooth LE is energiezuiniger en heeft een verder bereik dan de traditionele Bluetooth BR/EDR, maar helaas bezit deze een veel tragere snelheid.[17] De trage snelheid van Bluetooth LE is echter geen probleem voor de standaard operaties waarvoor deze implementatie gebruikt wordt.

Over het algemeen ondersteunen Bluetooth LE verbindingen alleen maar het lezen/schrijven van gegevens en het notificeren van wijzigingen. Vanwege deze redenen wordt bij standaard Bluetooth LE gebruik, geen verbinding opengehouden. Het voldoet om enkel een verbinding te maken wanneer gegevens gelezen of geschreven moeten worden, waarna deze dus ook weer afgesloten kan worden. Dit zorgt ervoor dat het energieverbruik van het gehele systeem beperkt blijft.[1]

[1] Vanwege deze reden krijgt deze implementatie de naam "Low Energy".

In het algemeen maakt Bluetooth LE gebruik van twee soorten apparaten: "centrals" en "peripherals".[10] Peripherals zijn apparaten die zichzelf, en zijn services, via Bluetooth LE adverteren zodat andere apparaten hier gebruik van kunnen maken. Centrals daarentegen, zijn apparaten die scannen naar peripherals in de buurt en hiermee verbinding kunnen maken. Nadat een verbinding is gestart door een central met een bepaalde peripheral, kan deze hiermee interageren via de aangeboden services.

Als een apparaat gebruik wilt maken van Bluetooth LE moet deze dus bepaalde services ondersteunen. Deze services kunnen door het apparaat of door de Bluetooth SIG ("Bluetooth Special Interest Group")[17] gedefinieerd worden, maar moeten altijd door het apparaat zelf geïmplementeerd worden. Iedere service biedt bepaalde kenmerken aan die gebruikt kunnen worden voor het verzamelen en/of het veranderen van gegevens. Ieder kenmerk kan een optionele omschrijving bezitten die vertelt waar het kenmerk voor staat.

Services, kenmerken en omschrijvingen worden gedefinieerd door een UUID[5][13]. Dit is een unieke identifier die gebruikt wordt door apparaten voor de service, kenmerk of omschrijving te herkennen, zodat de juiste actie uitgevoerd kan worden. Voor services, kenmerken en omschrijvingen die gedefinieerd zijn door de Bluetooth SIG wordt een 16-bit UUID gebruikt. Bij zelf gedefinieerde services, kenmerken en omschrijvingen moeten daarentegen 128-bit UUID's gebruikt worden.[13]

Dit protocol wordt in de maatschappij gebruikt voor verschillende doeleinden waaronder het ontvangen van sensormetingen, het besturen van verlichting en het delen van apparaat gegevens. Deze doeleinden zijn van algemeen nut in verschillende omgevingen waaronder huis- en bedrijfinstallaties, waar wij ons ook met de *BleiD* op focussen.

4.1.2 Client - Server via Wi-Fi/Ethernet

Een server stelt zichzelf beschikbaar op een netwerk via een netwerkpoort en wacht voor binnenkomende connecties. Van zodra een client probeert te verbinden met de server, wordt er een netwerkconnectie tussen de twee gestart en kunnen gegevens naar elkaar verzonden worden. Typisch zijn client en server aparte apparaten, maar deze kunnen ook voorkomen in hetzelfde apparaat als deze probeert te verbinden als client met de poort die hij als server heeft opengezet.

Om de verbinding te maken tussen client en server, maken we gebruik van een socket[20]. Een socket is een netwerkinterface dat gebruik maakt van het TCP[21] protocol voor twee apparaten te verbinden (in dit geval de client en de server). Deze verbinding is full-duplex wat betekent dat langs beiden kanten tegelijkertijd data verzonden en ontvangen kan worden. Vanwege deze reden kan de client data blijven verzenden naar de server zonder te moeten wachten op een antwoord van de server. Meerdere apparaten kunnen gelijktijdig op deze manier met de server connecteren. Het systeem kan hier continu mee omgaan

aangezien iedere client zijn eigen socket krijgt op de server. Dit zorgt ervoor dat alle gegevens altijd naar het juiste overeenkomstige apparaat worden verzonden.

Bij Wi-Fi/Ethernet is zowel het lezen, als het schrijven van gegevens tussen de client en de server mogelijk. Welke data er verzonden wordt en welke vorm deze data aanneemt, is volledig afhankelijk van het systeem zelf. Zo is er ondersteuning voor specifieke protocollen mogelijk en kan data geëncrypteerd worden moest dit gewenst zijn. Er zijn met andere woorden weinig limieten voor de data bij het gebruiken van Wi-Fi of Ethernet als communicatie, maar dit brengt zowel voordelen als nadelen met zich mee.

4.2 Evalutatie communicatievormen

4.2.1 Communicatie tussen Hub en externe apparaten

Het belangrijkste voor de communicatie tussen de hub en de externe apparaten is dat het protocol veel ondersteund wordt en dat het een vaste vorm van communiceren gebruikt. Het eerste gedeelte komt door het feit dat we zo veel mogelijk apparaten met de hub willen kunnen besturen. Hoe meer apparaten de hub ondersteund, hoe meer dat deze gebruikt kan worden voor verschillende doeleinden. Voor het tweede gedeelte geven we een voorbeeld van een communicatie die geen vaste vorm van communiceren gebruikt, namelijk Wi-Fi.

Bij Wi-Fi is het noodzakelijk voor programma's en applicaties om zich op elkaar af te stemmen, aangezien deze hetzelfde soort protocol en encryptie moeten ondersteunen. Met andere woorden, ze moeten dezelfde soorten berichten verzenden over de dataverbinding. Vanwege deze redenen is het via Wi-Fi/Ethernet moeilijker om de soorten aangeboden services te ontdekken van applicaties. Er is niet één vaste communicatievorm die alles definieert. Via Wi-Fi Aware is het mogelijk voor deze soort services te ontdekken, wat ons probleem zou oplossen. Helaas is er nog niet veel ondersteuning voor Wi-Fi Aware bij IoT apparaten en is het niet zeker wanneer deze wel meer ondersteunt zal worden.

Een protocol dat wel door veel apparaten ondersteund wordt en wat een vaste vorm van communiceren gebruikt, is Bluetooth. We willen de gebruiker zo veel mogelijk apparaten gelijktijdig laten gebruiken zonder dat constant gewisseld moet worden. Daarom hebben we Bluetooth Low Energy verkozen over zijn traditionele variant (Bluetooth BR/EDR), aangezien deze meer gelijktijdige connecties toelaat en omdat de belangrijkste services ontdekt kunnen worden zonder verbinding te maken.[2] Bluetooth Low Energy heeft minder ondersteunende apparaten dan de traditionele Bluetooth, maar support voor Bluetooth Low Energy blijft groeien en wordt steeds vaker gebruikt vanwege zijn energiezuinigheid. Vanwege deze reden heeft het voordelen naar de toekomst toe om Bluetooth Low Energy te gebruiken.

[2] Apparaten hebben bij Bluetooth LE de keuze om hun services te adverteren in de advertentiesignalen.

4.2.2 Communicatie tussen Android applicatie en Hub

In het begin van het onderzoek dachten wij ook gebruik te maken van Bluetooth LE voor de communicatie tussen de Android applicatie en de hub. Dit bleek echter niet de beste oplossing te zijn aangezien hier enkele limieten aan verbonden waren. Een voorbeeld hiervan is dat de hub Bluetooth "central" en "peripheral" mode tegelijkertijd moest kunnen gebruiken, wat standaard enkel mogelijk is met twee Bluetooth adapters of door constant tussen de twee te wisselen. Dit bracht ons idee voor Bluetooth LE te gebruiken voor de communicatie tussen de applicatie en de hub in twijfel. We moesten hier dus een alternatief voor gaan zoeken, wat uiteindelijk Ethernet/Wi-Fi geworden is.

Om deze keuze te maken hebben we verschillende soorten draadloze communicaties beschouwd en deze met elkaar vergeleken via een tabel (zie Tabel 4.1). De communicatievormen die we hebben beschouwd zijn: Bluetooth BR/EDR, Bluetooth LE, Ethernet/Wi-Fi en Wi-Fi Direct[9]. Wi-Fi Aware hebben we niet vergeleken met deze communicaties, aangezien deze standaard verwacht dat je een protocol hebt klaarstaan voor de echte verbinding (typisch Wi-Fi Direct).[3]

Voor de communicatie tussen de Android applicatie en de hub hebben we veel support nodig, aangezien we willen dat alle gebruikers hier gebruik van kunnen maken. We willen ze verder ook niet weerhouden van andere applicaties in de tussentijd te gebruiken. Als we met dit in het achterhoofd tabel 4.1 bekijken, dan kunnen we al vaststellen dat Wi-Fi Direct niet de juiste optie is. Gebruikers zouden dan moeten verbinden met de hotspot van de hub, waardoor Wi-Fi chips die geen "multi-role" ondersteunen moeten loskoppelen van hun huidige Wi-Fi verbinding. Wi-Fi Direct is verder ook een protocol dat relatief weinig gebruikt wordt bij IoT apparaten, dit terwijl wij een protocol zoeken dat wel door veel apparaten gebruikt wordt. We hebben nu nog de keuze tussen Bluetooth BR/EDR, Bluetooth LE en Ethernet/Wi-Fi.

Zowel Bluetooth LE als Bluetooth BR/EDR hebben het probleem, dat als we deze met de Bluetooth LE connectie van de hub en de apparatuur combineren, we een extra adapter nodig hebben. We zouden deze kunnen aansluiten op de Raspberry Pi en dit zo proberen te implementeren wat zorgt voor extra complexiteit, maar als we kijken naar Ethernet/Wi-Fi is dit niet nodig en merken we nog enkele voordelen op. Als we met de hub gebruik maken van Ethernet/Wi-Fi kan de Android applicatie deze overal bereiken waar netwerkverbinding is, door middel van zijn IP adres en poortnummer. Dit zorgt ervoor dat we de mogelijkheid hebben om de apparaten te besturen buiten zijn lokale omgeving.

Bij het gebruiken van Bluetooth voor de communicatie zouden wij net zo dicht bij de hub moeten staan als de apparaten zelf, wat ons het doel van de hub zou in twijfel doen trekken. Verder ondersteunt Ethernet/Wi-Fi een veel snellere dataconnectie zolang de netwerkverbinding hier sterk genoeg voor

[3]Zoals de naam suggereert, probeert het de apparaten enkel "aware" te maken van services door gebruik te maken van Wi-Fi.

is.[5] Het enige nadeel wat Ethernet/Wi-Fi in dit geval heeft is dat dit protocol geen "Plug and Play" ondersteunt. We moeten dus met andere woorden zelf de data en de werking van de communicatie implementeren. Gelukkig is dit geen groot probleem aangezien we zowel de Android applicatie, als de hub zelf implementeren. Als we andere apparaten zouden laten gebruik maken van de hub op deze manier, zou dit wel tot problemen kunnen leiden.

[5] In de meeste gevallen is dit wel van toepassing, vooral in vergelijking met de snelheid van Bluetooth.

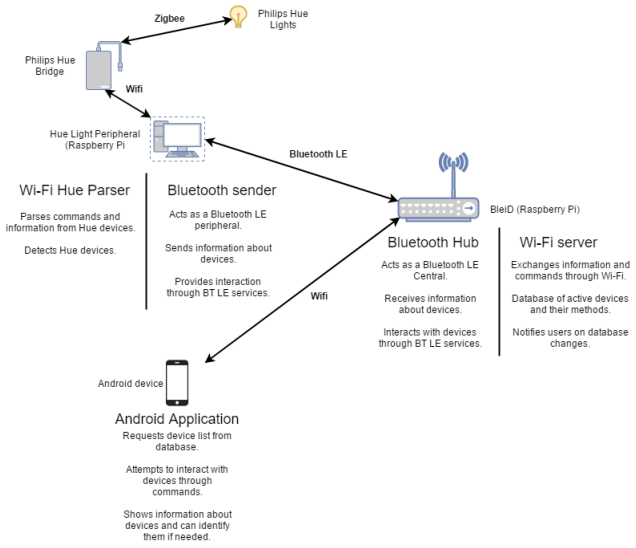
	Bluetooth BR/EDR	**Bluetooth LE**	**Ethernet/Wi-Fi**	**Wi-Fi Direct[9]**
Werking	Wisselen tussen slaves en master	Verbinding tussen peripheral en central	Rechtstreeks verbonden via het netwerk (via IP en poort)	Lokale hotspot voor te verbinden
Afstand	$<$ 10 m	50 - 100 m	/	10 - 30 m
Snelheid	2 - 3 Mbps	256 kbps	(Afhankelijk van het netwerk)	max. 65 Mbps
Voordelen	Mogelijkheid tot offline subnet vormen	Mogelijkheid tot offline subnet vormen	Overal bereikbaar waar netwerk-verbinding is	Mogelijkheid tot lokaal subnet vormen
	Sneller dan Low Energy mode	Lage batterij consumptie	Mogelijkheid tot subnet vormen	WPA2 encryptie
	Veel ondersteuning voor	Zeer lange afstand	Kan in samenwerking met andere protocol op RPI3[4]	Kan in samenwerking met ander protocol op RPI3
		Veel ondersteuning (blijft groeien)	Veel ondersteuning voor	
Nadelen	Extra adapter nodig	Extra adapter nodig	Meestal stuursysteem nodig (geen Plug&Play)	Op dit moment weinig gebruikt
	Korte afstand	Trage snelheid		Soms stuursysteem nodig (geen Plug&Play)
	Beperkt aantal slaves (7)	Beperkt aantal verbindingen		

Tabel 4.1: Vergelijking van communicatievormen.

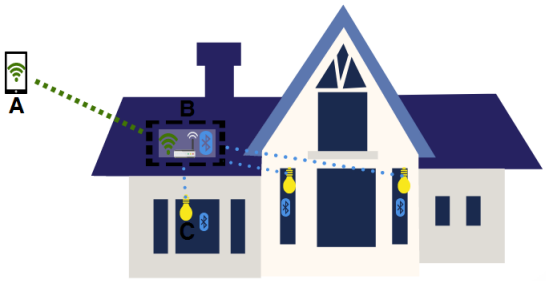
Hoofdstuk 5

Toepassing van systeem

In de implementatie (zie Figuur 5.1) zijn we een ander systeem bekomen dan hetgeen dat gesteld was in hoofdstuk 2 "Scenario's" (zie Figuur 5.2). Het vermelde scenario was een voorbeeld van wat we met ons onderzoek wilden bereiken, maar er zijn uiteindelijk meerdere manieren van aanpak. Wij zijn specifiek voor het huidige systeem gegaan, omdat wij gebruik wilden maken van "Philips Hue Lights" die origineel geen Bluetooth LE ondersteunen maar wel beschikbaar waren in onze omgeving. We evalueren ons systeem door nieuwe scenario's te omschrijven die de werking en het gebruik omschrijven.



Figuur 5.1: Opbouw van ons systeem.



Figuur 5.2: Voorstelling van het scenario.

5.1 Interactie met de "Hue Light Peripheral"

Scenario De gebruiker bevindt zich op zijn werk en herinnert dat hij per ongeluk de verlichting in zijn huis heeft laten aanstaan. Hij opent de applicatie en verbindt met de *BleiD* van zijn thuis, waarop hij kiest voor te interageren met de "Hue Light Peripheral". De gebruiker ziet dat hij de verbonden verlichting kan besturen en kiest ervoor om al de lampen uit te zetten. De lichten worden uitgezet en de gebruiker sluit de applicatie af.

De "Hue Light Peripheral" die wij gecreëerd hebben kan, zoals in het scenario hierboven, als een echt apparaat in de maatschappij gebruikt worden. Door zo een extra apparaat te implementeren en deze door het systeem te laten ondersteunen, hebben we aangetoond dat we een abstractielaag kunnen vormen voor de interactie met dit apparaat. De gebruiker wist in dit geval niet hoe er met de "Hue Light Peripheral" geconnecteerd moest worden en welke soorten berichten deze verwachtte. Al deze acties werden door de *BleiD* en de applicatie gedefinieert en de gebruiker hoefde enkel aan te geven wat hij met het apparaat wilde doen.

We hadden eerder in het onderzoek gesteld dat we verschillende Bluetooth LE apparaten tegelijkertijd wilden besturen, wat uiteindelijk het besturen van één apparaat werd die zelf een lijst van lichten controleerde. Het basisprincipe bleef hier hetzelfde bij, externe apparaten konden bestuurd worden door een minimale hoeveelheid interactie en kennis over de apparaten. Het resulterende systeem zorgt dus uiteindelijk met deze setup voor de interactie dat gewenst is door de gebruiker. Specifiek voor het vermelde scenario, is deze interactie het besturen van verlichting in een huisinstallatie.

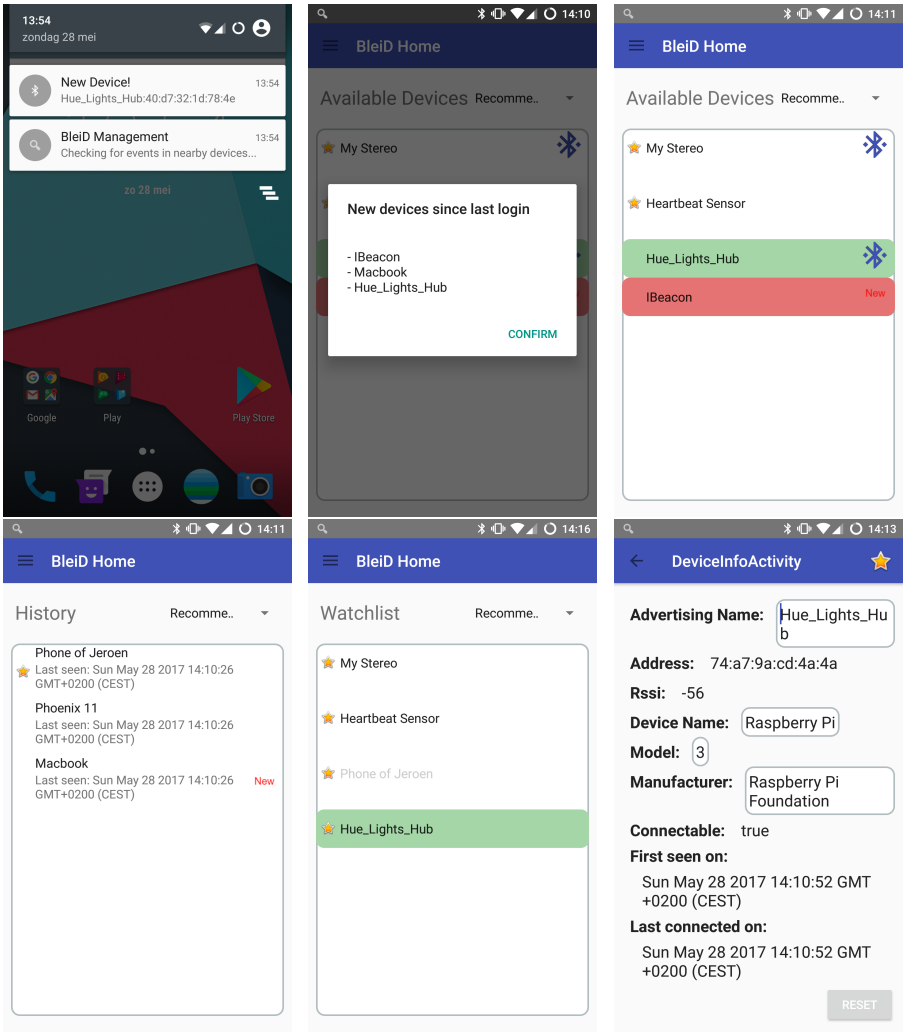
5.2 Interactie met het boekhoudingssysteem

Scenario Een apparaat komt in de buurt van het huis waar de hub is geïnstalleerd, waardoor de gebruiker een notificatie "New Device" ontvangt. De gebruiker klikt een paar uur later op deze notificatie waarop de applicatie

geopend wordt. De gebruiker krijgt direct een meldig met al de apparaten die nieuw gevonden zijn sinds de laatste keer dat hij de applicatie opende. Na het afsluiten van de melding ziet de gebruiker de lijst van verbonden apparaten en apparaten in de buurt en sorteert deze lijst op laatste nieuwe apparaten. Hij ziet het apparaat dat nieuw gevonden was niet in deze lijst omdat deze nu uit de buurt is en navigeert daarom naar de geschiedenis van gevonden apparaten. Hier geeft hij enkele apparaten aan die vergeten mogen worden en ziet het nieuwe apparaat met zijn bijhorende gegevens. Via deze gegevens bepaalt de gebruiker wat voor soort apparaat het is en kiest ervoor om deze in de "watchlist" te plaatsen. De gebruiker sluit de applicatie en na een paar uur komt het apparaat weer in de buurt, waarop de gebruiker een nieuwe notificatie "Watched device found" krijgt. De gebruiker interageert met het apparaat en sluit de applicatie af.

Het boekhoudingssysteem vermeldt in dit scenario is te zien op Figuur 5.3. Het doel van dit boekhoudingssysteem is volledig naar de gebruiker toe gericht. We willen het systeem meer "intelligible" (begrijpbaar) naar de gebruiker toe maken zodat deze geen problemen heeft met de interactie. Voor dit te bereiken zijn duidelijke indicaties en feedback nodig die de gebruiker voldoende informatie geven over de staat van het systeem.

De gebruiker heeft een beter overzicht over de apparaten en de hub, waardoor interageren met de apparaten simpeler wordt en de gebruiker meer interactie heeft met de hub. Zo kan de gebruiker bijvoorbeeld in de geschiedenis kiezen welke apparaten te vergeten zodat deze lijst niet suboptimaal gevuld wordt met ongewenste apparaten. We willen voor de gebruiker snelle interactie voorzien zodat deze rechtstreeks aan de apparaten kan die hij wilt besturen. Hier helpen ook de "watchlist" en de sorteerfuncties bij, aangezien deze ervoor zorgen dat de gebruiker kan kiezen welke apparaten vanboven getoond moeten worden. Zelfs als er dan grote lijsten van apparaten zijn, kan de gebruiker nog steeds zijn gewenste apparaten snel vinden.



Figuur 5.3: Voorbeelden van boekhoudingssysteem in de applicatie.

Hoofdstuk 6

Vergelijking met gerelateerd werk

We hebben al vermeld dat we bepaalde technieken gebruiken in ons systeem, maar nog niet expliciet waarom we deze gebruiken. Er zijn verschillende technieken in de IoT waar we de focus op hadden kunnen leggen, voorbeelden hiervan zagen we eerder in hoofdstuk 1. Of we de juiste techniek keuzes hebben gemaakt bij het bouwen van ons systeem is afhankelijk van verschillende factoren. Voor deze te kunnen evalueren is het belangrijk om ze te vergelijken met de bestaande technieken en hun voor- en nadelen af te wegen.

6.1 Bluewave

Er zijn veel gebieden in de IoT die handig gebruik kunnen maken van *Bluewave*. "Context sharing" is vaak een complex probleem met verschillende limiteringen. Zo is de mogelijkheid om context te kunnen delen met verschillende soorten apparaten niet vanzelfsprekend en heeft meestal een noodzaak tot het installeren van bepaalde software. *Bluewave* brengt een abstractielaag in de IoT interactie door verschillende apparaten gebruik te laten maken van zijn principes om de info van gebruikers te verzamelen, zonder dat deze apparaten met elkaar moeten verbinden. Er is dus geen probleem met de compatibiliteit van apparaten omdat *Bluewave* de manier specifieert waarop de informatie vrijgegeven wordt. De apparaten hoeven dan enkel nog maar te kiezen welke gegevens zij willen vrijgeven, daarna kunnen deze gegevens rechtstreeks gebruikt worden door externe apparatuur. Het belangrijkste aspect om dit te laten werken is natuurlijk dat de externe apparatuur zelf ook gebruik gaat maken van *Bluewave*. Dit zou betekenen dat alle gebruikers van deze apparaten hun Bluetooth namen naar urls moeten veranderen en hun gegevens moeten vrijgeven, wat zeker niet door iedereen gewenst is.

Het grootste verschil tussen ons systeem en *Bluewave* ligt in de manier

waarop gebruik wordt gemaakt van het tussenliggend apparaat. Bij *Bluewave* is er een betrouwbare server die zich op eender welke locatie kan bevinden en waar enkel de gegevens van apparaten opgeslagen worden. In ons systeem maken we gebruik van een hub die zich dichter bij de apparaten bevindt en met deze apparaten zal interageren. Hierbij wordt dus niet alleen informatie over deze apparatuur opgeslagen, maar wordt deze informatie dus ook gebruikt en aangepast door de hub. Vanwege deze redenen wordt *Bluewave* dus beter gebruikt in situaties waarbij gegevens enkel gelezen mogen worden. Verder is het bij *Bluewave* van belang dat gebruikers zelf niet interageren met het systeem, maar dat dit voor hun allemaal automatisch gebeurt. Bij ons systeem is het de gebruiker die met het systeem interageert waarop het systeem de gewenste acties uitvoert op de verbonden apparatuur. De situaties waarbij de twee van toepassing zijn, zijn dus compleet verschillend. De ene is beter voor gebruik in openbare plaatsen met een groot aantal gebruikers, terwijl de andere beter is voor lokaal gebruik zoals in een huisinstallatie. Toch wordt bij beiden technieken een zeer gelijkende methode gebruikt, een tussenliggend apparaat die zorgt voor de abstractie. De twee technieken hebben in het algemeen een ander doel voor ogen en hebben daarom dus een andere manier van gebruik als resultaat.

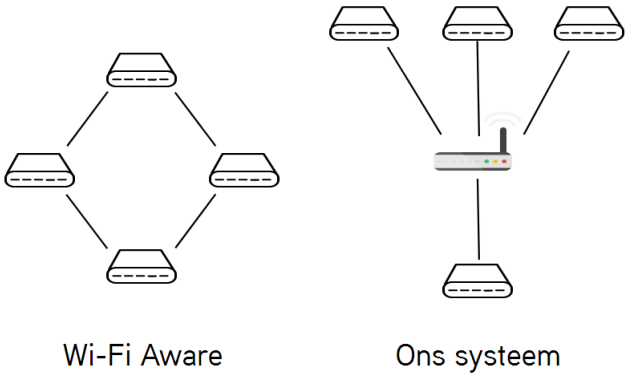
6.2 IBeacon

IBeacon heeft een zeer verschillende werking ten opzichte van ons systeem, maar is zeer gelijkend aan *Bluewave*. Bij *IBeacon* ligt de focus op het ontdekken van de locatie van gebruikers. Deze focus is een pak lager dan de hoeveelheid ondersteunt door *Bluewave* aangezien hier apparaten zelf kunnen kiezen welke gegevens nodig zijn. Ondanks deze beperking zijn er een aantal voordelen aan het gebruiken van *IBeacon*. Apparaten kunnen bijvoorbeeld eenvoudig hun eigen locatie ontdekken zonder dat hier extra hardware voor geïnstalleerd moet worden zoals sensoren. Als je hetzelfde effect wilt bereiken met *Bluewave* dan zal specifiek de locatie opgeslagen moeten worden op de server, wat lang niet zo nauwkeurig zal zijn als de methode van *IBeacon*. Dit is vanwege het feit dat hier gebruik gemaakt moet worden van de locatie via GPS of Wi-Fi.[22] Verder moet de Bluetooth naam van het apparaat bij *IBeacon* niet omgevormd worden naar een url en hoeft geen "gevoelige" informatie over de gebruiker gedeeld worden. Voor verschillende applicaties kan vanwege deze redenen dus eerder de voorkeur gelegd worden bij *IBeacon,* zolang dat deze genoeg gegevens biedt voor de applicatie.

Wat opvalt bij *IBeacon* en *Bluewave* is dat beiden technieken niet rechtstreeks verbinden met de apparaten, maar informatie vrijgeven via een indirecte communicatie zoals adverterende signalen. Ons systeem maakt wel gebruik van verbinding tussen apparaten aangezien wij meer willen bereiken dan enkel het ophalen van gegevens. Om te kunnen interageren met apparaten hebben we een verbinding nodig om bepaalde acties te kunnen uitvoeren. Voor dit te verduidelijken bespreken we nu "Wi-Fi Aware" die, net zoals ons systeem, het toelaat te verbinden met apparaten.

6.3 Wi-Fi Aware

Wi-Fi Aware is nog relatief nieuw en wordt dus nog niet massaal gebruikt in de maatschappij. Er zijn aspecten in Wi-Fi Aware die vergelijkbaar zijn met *Bluewave* en *IBeacon*, zoals het ontdekken van services en apparaten via signalen in de cluster. Maar in tegenstelling tot deze technieken wordt er bij Wi-Fi Aware wel een verbinding gemaakt tussen apparaten van zodra deze met elkaar willen interageren. De manier waarop connectie wordt gevormd tussen apparaten in Wi-Fi Aware is verschillend aan de connectie gevormd tussen de apparaten in ons systeem. Om hier een betere voorstelling van te krijgen zie Figuur 6.1:



Figuur 6.1: Vergelijking Wi-Fi Aware met ons eigen systeem.

Bij ons systeem maken we een verbinding tussen apparaten door middel van een tussenliggende hub die zorgt voor een abstractielaag voor de interactie. Bij Wi-Fi Aware daarentegen, wordt gebruik gemaakt van een ad-hoc netwerk tussen de apparaten waardoor deze rechtstreeks met elkaar verbonden zijn. Er is bij Wi-Fi Aware met andere woorden geen middenstation nodig. De reden dat deze opstelling bij Wi-Fi Aware mogelijk is, komt door het feit dat al deze apparaten gebruik maken van dezelfde service. Al de apparaten moeten gebruik maken van Wi-Fi Aware, waardoor deze altijd dezelfde vorm van interactie ondersteunen. Ons systeem daarentegen, laat de verschillende apparaten toe om andere services te gebruiken zolang de hub deze maar kan gebruiken. Met andere woorden hebben wij één enkele "master" die zorgt voor de verbinding met de apparaten en services, terwijl Wi-Fi Aware al de apparaten dezelfde vorm van communicatie en services laat gebruiken.

Beiden systemen hebben uiteindelijk hun voor- en nadelen. Wi-Fi Aware kan bijvoorbeeld geen ondersteuning bieden voor externe apparaten die geen Wi-Fi gebruiken, maar ons systeem wel.[1] En bij ons systeem moet alle verbinding lopen door de hub, wat zorgt voor een bottleneck in de verbindingssnelheid en

[1] Buiten de Android applicatie die gebruikt wordt om met de hub te interageren

de vrijheid van directe interactie tussen apparaten ontneemt die Wi-Fi Aware aanbiedt. Ook hier is ons systeem beter bruikbaar voor thuis- en werklocaties en is Wi-Fi Aware beter bruikbaar voor openbare locaties met veel apparaten.

Als we Wi-Fi Aware vergelijken met een systeem zoals *Bluewave*, wat ook beter is voor openbare locaties, dan zien we veel gelijkenissen tussen de twee. Beiden systemen laten toe gegevens over andere apparaten te ontdekken via hun signalen. Het is echter bij de manier waarop de gegevens worden opgeslagen waar het grootste verschil ligt. Bij *Bluewave* worden de gegevens extern opgehaald via een server, terwijl bij Wi-Fi Aware een directe verbinding tussen de apparaten gevormd wordt.

Als we kijken naar de methode van *Bluewave*, dan zien we dat hierbij het eenvoudig is om "read-only" gegevens van verschillende apparaten snel op te halen. Alles wordt namelijk opgeslagen op een server die via HTTP bereikbaar is en door extra parameters snel gelezen kan worden. Vanwege deze reden wordt *Bluewave* best gebruikt in gevallen waarbij een apparaat feedback geeft, afhankelijk van de gelezen gegevens van gebruikers. Wi-Fi Aware is minder bruikbaar in deze situatie omdat dan een peer-to-peer verbinding gevormd zou moeten worden tussen ieder apparaat. Dit kan zorgen voor een grote overhead voor de kleine hoeveelheid data dat echt nodig is, met als gevolg een grote energieconsumptie. Wi-Fi Aware kan vanwege deze reden beter gebruikt worden in situaties waarbij apparaten, of veel data met elkaar moeten delen, of naar elkaar moeten lezen en schrijven. Applicaties voor entertainment zoals smartphone games zijn hier een perfect voorbeeld van.

Hoofdstuk 7

Discussie

Dankzij de interactie geleverd door de *BleiD* hoeft de gebruiker geen applicatie meer te downloaden voor ieder soort apparaat, enkel nog maar de applicatie "BleiD Home". Dit zorgt ervoor dat de gebruiker alle apparaten tegelijkertijd kan besturen zonder tussen applicaties te moeten wisselen. De hoeveelheid interactie tussen de gebruiker en zijn apparatuur is verminderd omdat de *BleiD* deze voor de gebruiker automatiseerd.

We proberen met de *BleiD* ervoor te zorgen dat verschillende apparaten op een consistente manier gebruikt kunnen worden. In normale situaties moeten gebruikers met ieder soort apparaat op een andere manier interageren via bijvoorbeeld andere applicaties, omdat deze van andere firma's kunnen zijn of omdat deze compleet andere functionaliteiten bieden. Het is ook belangrijk dat er gezorgd wordt voor een intuïtieve manier van interactie zodat gebruikers minder problemen hebben met het gebruiken van andere of nieuwe apparaten. We probeerden dit na te streven met het implementeren van het boekhoudingssysteem, wat bepaalde feedback geeft aan de gebruiker en hem de mogelijkheid geeft om de verschillende apparaten te beheren.

Helaas zijn er nog enkele limieten verbonden aan het systeem. Zo kan bijvoorbeeld geen te groot aantal apparaten tegelijkertijd bestuurd worden, dankzij de limiet aan het aantal connecties die gelijktijdig gevormd kunnen worden met de connectievorm die we huidig gebruiken. Voor een standaard huisinstallatie zoals in de scenario's is dit nog geen probleem, maar bij grotere bedrijven die gebruik willen maken van een *BleiD* kan dit wel voor problemen zorgen. Hierbij zal het aantal apparaten die verbonden moeten worden en waarmee geinterageerd moet worden sterk oplopen.

Verder is er nog een limiet aan het aantal soorten apparaten die *BleiD* ondersteund. Als een apparaat niet ondersteund is dan zou de gebruiker zelf moeten kunnen kiezen hoe het systeem moet interageren met het apparaat. Dit zorgt ervoor dat *BleiD* kan werken voor apparaten die deze eerder niet ondersteunde. Het nadeel hiervan is dat de gebruiker zelf moet specifiëren hoe om te gaan met het apparaat, wat tegen het principe gaat van een hub die alle interactie zou

moeten voorzien. Helaas kan dit nadeel niet vermeden worden omdat het aantal soorten apparaten te groot is om deze allen rechtstreeks te kunnen ondersteunen met de hub.

Hoofdstuk 8

Conclusie

Het aantal IoT apparaten die gebruikt worden in de maatschappij blijft continu stijgen, waardoor dat de gebruikte technieken ook steeds blijven toenemen. Vanwege deze reden is er een enorme vraag naar het vinden van manieren die de interactie versimpelen en de apparaten compatibel met elkaar maken. Dit is een complex probleem waar nog geen optimate oplossing voor bestaat en ook die van ons systeem is niet optimaal. Er zijn verschillende limieten verbonden aan onze huidige hub op het gebied van gebruikte technieken, ondersteunde apparaten en algemeen gebruik. Nu moeten we ons de vraag stellen of dat onze hub de eerder vermelde problemen oplost, ondanks zijn limieten.

Zodra een extern apparaat (Android toestel) verbinding maakt met de hub, kan deze via de hub al de apparatuur vinden die gevonden en verbonden zijn en deze beheren en besturen. Het extern apparaat kan rechtstreeks interageren met de verbonden apparatuur, zonder dat deze weet moeten hebben van specifieke protocollen of software. Deze gevormde abstractielaag voor de interactie tussen IoT apparaten, lost met andere woorden ons eerder gestelde compatibiliteitsprobleem op.

De hub doet dus conceptueel wat het moet doen, maar het geïmplementeerde model is nog beperkt. Deze kan altijd nog uitgebreid worden voor het ondersteunen van meerdere technieken en apparaten, maar het concept zal altijd hetzelfde blijven, namelijk de abstractielaag vormen voor de interactie en communcatie. Het toevoegen van het extra apparaat zorgde ook voor een extra hoeveelheid complexiteit, aangezien we eerst ervoor moesten zorgen dat we de "Philips Hue Lights" konden omvormen tot een Bluetooth LE peripheral. Dit toont aan dat we niet alleen een abstractielaag kunnen voorzien voor de interactie, maar dat we ook een bepaalde service kunnen beschikbaar stellen via een ander protocol.

Dit is een belangrijk aspect in de IoT, aangezien we ervoor kunnen zorgen dat apparaten bereikbaar zijn via protocollen die eerst niet gebruikt konden worden. Dit is vooral van belang voor gevallen waarbij we een apparaat willen

gebruiken dat maar één type protocol ondersteunt, of dat enkel oude en ineffectieve protocollen beschikbaar heeft. We kunnen dit apparaat dan omvormen tot een ander protocol, waardoor we deze op meerdere manieren kunnen gebruiken. Dit principe lijkt op eerste zicht op de abstractielaag gevormd door de hub, maar dit is niet volledig correct. Het verschil tussen de twee, is dat de hub zorgt voor de abstractielaag voor verschillende apparaten die bepaalde protocollen gebruiken, ook ondersteunt door de hub. De "Hue Light Peripheral" doet het omgekeerde, een bepaalde service wordt omgezet naar een ander protocol zodat deze via daar gebruikt kan worden door andere systemen. Het maakt voor de "Hue Light Peripheral" geen verschil welk protocol dat de service eerder gebruikte, want hij wordt specifiek geïmplementeerd om ermee te kunnen omgaan.

We hebben dus uiteindelijk twee apparaten in onze implementatie die helpen met de compatibiliteit af te handelen, maar beiden hebben een ander doel en behandelen dit op een andere manier.

Hoofdstuk 9

Toekomstig Werk

9.1 Niet-ondersteunde apparaten

Op dit moment moet het systeem de apparaten ondersteunen om hiermee om te kunnen gaan. Niet-ondersteunde apparaten worden op dit moment nog niet goed afgehandelt, maar zijn een belangrijk aspect van het systeem. Zowel ondersteunde als niet-ondersteunde apparaten moeten gebruikt kunnen worden voor de abstractielaag compleet te maken. Zodra wij deze ondersteund hebben, zou het mogelijk moeten zijn voor alle apparaten te besturen, zonder weet te moeten hebben van hun achterliggende werking.

Een mogelijkheid voor dit ondersteunen is door de gebruiker zelf te laten kiezen welke interactie moet gebeuren. Dit kan gaan van het kiezen van de berichtstructuur tot het zelf definiëren van een user interface die gebruikt moet worden. Verder is er de mogelijkheid om de hub te laten scannen naar de services en te laten ontdekken hoe ermee moet omgegaan worden. Dit vraagt een zeer complex algoritme met waarschijnlijk AI ("Artificial Intelligence") aspecten voor te bepalen welke services gebruikt worden. Er zijn zo veel services in de wereld dat het onmogelijk is om deze allemaal rechtstreeks te ondertussen. Hoe er omgegaan wordt met niet-ondersteunde apparaten is dus nog een apart onderzoek waar verder op ingegaan kan worden.

9.2 Niet-ondersteunde protocollen

We ondersteunen in ons huidig onderzoek enkel het gebruik van Bluetooth Low Energy voor de externe apparaten. We zouden dit probleem kunnen verhelpen door meerdere protocollen te ondersteunen voor de externe apparaten, zoals Bluetooth BR/EDR, Wi-Fi en/of Zigbee. Dit zou de abstractielaag vergroten waardoor meerdere apparaten ondersteund worden. Het ondersteunen van extra protocollen, op welke manier dan ook, zou ervoor kunnen zorgen dat meer mensen gebruik kunnen maken van ons systeem. Hoe groter de abstractielaag

wordt hoe meer dat wij streven naar een generieke methode voor alle soorten apparaten te kunnen afhandelen.

We kunnen hiervoor ook een andere richting uitgaan en net zoals we gedaan hebben in de voorbeeld opstelling, zelf een extern apparaat maken die interageert met een systeem via zijn protocol. Deze kunnen dan beschikbaar gesteld worden via protocollen die wij wel ondersteunen, zoals in ons geval Bluetooth Low Energy. De complexiteit hiervan ligt heel hoog en het is bijna onmogelijk om een extern apparaat te maken voor ieder systeem apart, dus kunnen we ons de vraag stellen of dit ons probleem wel oplost.

9.3 Interactiemogelijkheden

Om meer apparaten dan enkel Android toestellen toegang te bieden tot het systeem van de *BleiD,* zouden we kunnen gebruik maken van een andere interactiemogelijkheid. Voorbeelden hiervan zijn het maken van Apple app, een Windows app of misschien beter in het algemeen, een website. Het maken van een website die kan interageren met de *BleiD* zorgt voor enkele nieuwe mogelijkheden, aangezien de meeste apparaten webpagina's kunnen bezoeken. We zouden ook een alternatieve versie voor ieder soort apparaat kunnen maken. Dit zorgt ervoor dat de gebruiker kan omgaan met de *BleiD* op het apparaat dat deze het liefst gebruikt.

We kunnen wel stellen dat we over het algemeen best een extra website maken aangezien dit de meeste ondersteuning biedt voor apparaten . We maken voor de communicatie tussen de User Interface en de *BleiD* gebruik van Wi-Fi dus kunnen we de website misschien best hosten, rechtstreeks vanuit de *BleiD*. Dit handelt een heel gedeelte van onze complexiteit af en geeft ons de mogelijkheid om rechtstreeks op het systeem te werken. We hadden eerst niet gekozen voor een website te maken omdat we de mogelijkheid willen bieden aan de gebruiker voor notificaties te ontvangen van de *BleiD*. Dit is handig in situaties waarbij de gebruiker in een nieuwe omgeving terechtkomt en we willen notificeren dat er een *BleiD* op het netwerk staat.

9.4 Wi-Fi Aware

We stelden eerder al dat we mogelijks gebruik kunnen maken van Wi-Fi Aware in de applicatie. We zouden dit kunnen doen om apparaten de *BleiD* en zijn service te laten ontdekken. We doen dit dan als volgt: We registreren het programma bij Wi-Fi Aware en de *BleiD* stelt zichzelf via deze beschikbaar door zijn applicatie identifier. De gebruiker opent onze applicatie en vindt de *BleiD* via Wi-Fi Aware waarop deze aangeeft dat hij met deze wilt gaan interageren. De Wi-Fi connectie tussen de twee wordt gevormd en de gebruiker kan beginnen interageren met de aangeboden service. Op deze manier maken we gebruik van Wi-Fi Aware voor meer ondersteuning te bieden voor onze applicatie en te zorgen dat deze eenvoudiger te vinden en te gebruiken is.

Bibliografie

[1] S. Farahani, *ZigBee Wireless Networks and Transceivers*, 1st ed. Newnes, September 2008.

[2] P. F. Ovidiu Vermesan, *Digitising the Industry - Internet of Things Connecting the Physical, Digital and Virtual Worlds*. River Publishers, August 2016.

[3] A. A. de Freitas, M. Nebeling, A. S. K. K. Ranithangam, J. Yang, and A. K. Dey, "Bluewave: enabling opportunistic context sharing via bluetooth device names," in *EICS '16 Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. ACM New York, NY, USA, June 2016.

[4] *Getting Started with iBeacon*, 1st ed., Apple Inc., June 2014. [Online]. Available: https://developer.apple.com/ibeacon/Getting-Started-with-iBeacon.pdf

[5] P. Leach, M. Mealling, and R. Salz, *A Universally Unique IDentifier (UUID) URN Namespace*, Network Working Group (NTWG), July 2005, also referred to as RFC 4122. [Online]. Available: https://tools.ietf.org/html/rfc4122

[6] *Android iBeacons Integration Guide*, PlotProjects. [Online]. Available: https://www.plotprojects.com/integration-guide-for-beacons-on-android/

[7] G. Conte, M. D. Marchi, A. A. Nacci, V. Rana, and D. Sciuto, "Bluesentinel: a first approach using ibeacon for an energy efficient occupancy detection system," in *BuildSys '14 Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*. ACM New York, NY, USA, November 2014.

[8] *Wi-Fi Aware^{TM}: Discover the World Nearby*, Wi-Fi Alliance, July 2015, enabling Personalized Social, Local, and Mobile Experiences.

[9] *Wi-Fi CERTIFIED Wi-Fi Direct: Personal, portable Wi-Fi technology*, Wi-Fi Alliance, September 2014.

[10] C. Gomez, J. Oller, and J. Paradells, "Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology," *NCBI National Center for Biotechnology Information*, August 2012. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3478807/

[11] *$UPnP^{TM}$ Device Architecture 1.1*, UPnP Forum, October 2008. [Online]. Available: http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf

[12] *Raspberry Pi 3 Model B*, rs components. [Online]. Available: http://docs-europe.electrocomponents.com/webdocs/14ba/0900766b814ba5fd.pdf

[13] *Bluetooth Low Energy (BLE)*, 3rd ed., Cypress, December 2015. [Online]. Available: http://www.cypress.com/file/220246/download

[14] P. Teixeira, *Professional Node.js Building Javascript Based Scalable Software*, 2013, john Wiley & Sons, Inc. [Online]. Available: http://htchttp.s3.amazonaws.com/books/professional_node.js.pdf

[15] E. Wittern, P. Suter, and S. Rajagopalan, "A look at the dynamics of the javascript package ecosystem," in *Mining Software Repositories (MSR)*. IEEE, May 2016.

[16] *JSON Javascript Object Notation*, TutorialsPoint, 2017. [Online]. Available: https://www.tutorialspoint.com/json/json_tutorial.pdf

[17] *Bluetooth Core Specification*, Bluetooth Special Interest Group, December 2016. [Online]. Available: https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx$?doc_id{=}421043$

[18] I. C. Society, "Part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications," in *IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements*. The Institute of Electrical and Electronics Engineers, Inc, October 2009. [Online]. Available: http://luci.subsignal.org/~jow/802.11n-2009.pdf

[19] J. Gosling, B. Joy, G. Steele, G. Bracha, and A. Buckley, *The Java Language Specification*, java se 7 ed., Oracle, February 2013. [Online]. Available: https://docs.oracle.com/javase/specs/jls/se7/jls7.pdf

[20] *Programming: Socket programming*, 7th ed., IBM, February 2006. [Online]. Available: https://www.ibm.com/support/knowledgecenter/ssw_i5_54/rzab6/rzab6.pdf

[21] L. Parziale, D. T. Britt, C. Davis, J. Forrester, W. Liu, C. Matthews, and N. Rosselot, *TCP/IP Tutorial and Technical Overview*, IBM, December 2006. [Online]. Available: https://www.redbooks.ibm.com/redbooks/pdfs/gg243376.pdf

[22] M. Singhal and A. Shukla, "Implementation of location based services in android using gps and web services," *IJCSI International Journal of Computer Science Issues*, vol. 9, January 2012. [Online]. Available: http://www.ijcsi.org/papers/IJCSI-9-1-2-237-242.pdf