Universiteit Gent

Project in het kader van het Vakoverschrijdend Projectvak
in de Bachelor Elektrotechniek

IoT en Solid: Een applicatie voor privégegevens met LwM2M

Groep 3
Axl Bomhals
Kobe Hens
Thijs Paelman
Flor Sanders

Prof. Johan Bauwelinck
Prof. Jeroen Hoebeke
dr. Abdulkadir Karaağaç
ir. Bart Moons

Academiejaar 2019-2020

Inhoudsopgave

1 Inleiding

Het begrip Internet of Things (IoT) zoals we dit nu kennen werd in 1999 geïntroduceerd door Kevin Ashton en wordt beschreven als een systeem van verbonden toestellen die met elkaar kunnen communiceren zonder tussenkomst van mensen ('Machine to Machine' - M2M).

Hoewel dit concept vrij nieuw is, zijn deze zgn. slimme, verbonden apparaten minder en minder weg te denken uit onze directe leefomgeving. Via toepassingen als domotica, smart cities, smart healthcare (IoMT) en industriële IoT (IIoT) heeft IoT zich kunnen profileren als essentiële technologie voor industry 4.0.

De laatste jaren hebben IoT-toepassingen ook een stevige opmars gekend, wat resulteert in enkele indrukwekkende voorspelde statistieken voor 2020 [1]:

- 30 miljard IoT apparaten online, met een groei van 127 per seconde.
- 1.3 biljoen USD in jaarlijkse uitgaven omtrent IoT wereldwijd.
- Slechts 0.06 % van de beschikbare toestellen zijn momenteel verbonden.

Ondanks de vooruitgang van de laatste jaren zijn er nog steeds significante struikelblokken die overwonnen moeten worden alvorens massa-adoptie van IoT kan worden gerealiseerd, waaronder platformfragmentatie, privacy, controle, abandonware, opslag en beveiliging [2].

De problemen gerelateerd aan opslag en privacy zijn ook van toepassing op andere delen van het hedendaagse internet. Als reactie hierop zijn tal van maatregelen genomen en projecten opgestart. Het Solid-project valt zeker onder deze noemer en werd in 2016 door Tim Berners-Lee, de uitvinder van het wereldwijde web, opgestart [3].

Het doel van Solid is om het lezen en schrijven van gedecentraliseerde data op het internet mogelijk te maken. Deze data dient onder controle en in bezit van de gebruiker te blijven zodat privacy een garantie en geen gok is. De principes waar Solid-toepassingen op gebouwd zijn worden in 3 punten opgesomd [4]:

- Gegevens in eigen handen.
- Modulair ontwerp.
- Hergebruik van bestaande data.

Het project dat door Solid ondernomen wordt is van enorme omvang, maar als ze in hun opzet slagen zou dit kunnen resulteren in een nieuwe revolutie van het web [4].

2 Motivatie

Zoals al aangehaald in de inleiding kent IoT een zeer sterke groei die gepaard gaat met enkele problemen. In dit project trachten wij hiervoor een oplossing te bieden door de wereld van IoT te koppelen aan Solid en op die manier de beste eigenschappen van beide te combineren.

Het is de rijke connectiviteit tussen toestellen (typisch voor IoT) en de gedistribueerde opslag (mogelijk gemaakt door Solid) die samen benut zullen worden.

De motivatie hiertoe wordt uitgelegd aan de hand van een situatieschets: het huis van de toekomst. Dit zal uitgerust zijn met allerhande IoT-apparaten zoals ingebedde sensortoestellen die onderling informatie uitwisselen om een geautomatiseerd geheel te vormen.

Een goed voorbeeld hiervan zijn de regensensoren die vandaag de dag reeds in combinatie met automatische ramen gebruikt worden. Via eenvoudige logica kan dan worden geïmplementeerd zo dat bij mooi weer het huis periodiek wordt verlucht en dat bij regenval de ramen gesloten worden om waterschade te voorkomen.

Een belangrijke vraag bij dit proces is waar de data worden opgeslagen. Met de hedendaagse platformen is het vaak zo dat dit op servers (en dus ook onder controle) van firma's (de installateur, de producent) gebeurt. Dit is bv. dikwijls het geval voor de metingen van de opbrengst van zonnepanelen [5].

Er vallen enkele problemen te identificeren gerelateerd aan deze aanpak:

- De toegang tot de data is enkel gegarandeerd zolang de firma dit toelaat.
- De consument heeft weinig zicht op wie er juist toegang heeft tot deze data en op welke manier deze gebruikt worden.
- Het is vaak moeilijk om de data uit deze platformen op te halen om er zelf mee aan de slag te gaan.
- Door platformfragmentatie is het dikwijls zo dat men voor verschillende diensten verschillende websites moet raadplegen, waardoor er weinig overzicht is.

Solid brengt hier verandering in: door Solid te gebruiken is de data in jouw beheer en ben jij degene die beslist welke partijen toegang krijgen tot jouw data. Verder is het mogelijk om data van verschillende diensten te hergebruiken en te raadplegen op één platform.

De bovenstaande voorbeelden wekken mogelijks niet al te veel onrust op wat privacy betreft, maar het probleem wordt duidelijker als we zaken als aanwezigheidssensoren en spraakherkenning overwegen, waarbij de gevolgen veel groter kunnen zijn indien de data in verkeerde handen vallen.

Uit de actualiteit blijkt dat bedrijven niet altijd even betrouwbaar zijn wanneer het aankomt op de correct behandelen van persoonlijke data. Als concrete voorbeelden denken we aan de slimme spraakassistenten van Google, Amazon en Apple waarbij gesprekken afgeluisterd werden en het Facebook-Cambridge Analytica dataschandaal [6][7].

3 Projectoverzicht

Het concrete doel van dit project wordt uiteengezet d.m.v. een demonstratienetwerk waarin alle aspecten van een praktische oplossing in kleinere schaal aan bod komen. We zullen in dit verslag dan ook dit netwerk als vertrekpunt gebruiken om de verschillende concepten te introduceren.

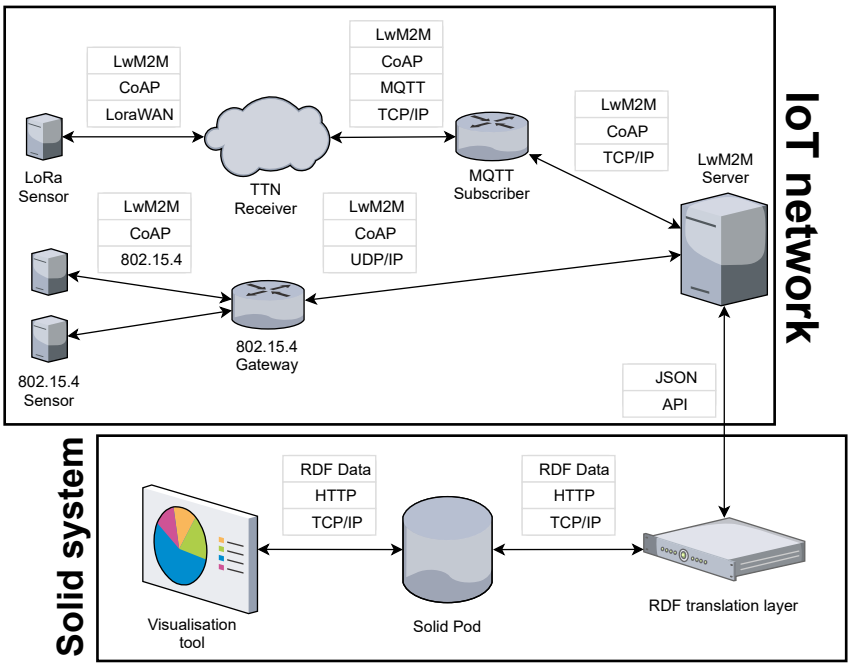
Figuur 1 geeft een overzicht van de verschillende componenten in het demonstratienetwerk, op welke manier ze tot elkaar in relatie staan en via welke protocollen de informatie-overdracht plaatsvindt.

Fundamenteel bestaat dit netwerk uit twee secties: aan de bovenkant het IoT netwerk met sensoren en een centrale server gebruik makende van LwM2M ('Lightweight Machine To Machine') als overkoepelende communicatiestandaard. Aan de onderkant vinden we het Solid systeem terug met centraal een datapod voor de opslag van historische data met daarrond de nodige componenten om vertaling en visualisatie van deze data te faciliteren [8][4].

De taakverdeling wat de implementatie betreft ziet er als volgt uit:

- IoT netwerk
  - Opstellen LwM2M-server: Axl, Kobe & Thijs.
  - IoT LoRa-netwerk: Axl.
  - IoT IEEE 802.15.4 netwerk: Kobe.
- Solid-systeem
  - RDF ontologie: Thijs & Flor.
  - LwM2M → RDF vertaalapplicatie: Thijs.
  - Visualisatietool voor historische sensordata: Flor.

De planning wordt in de vorm van een Gantt schema in bijlage 8.1 meegegeven.



Figuur 1: Visualisatie van het demonstratienetwerk.

4 Implementatie

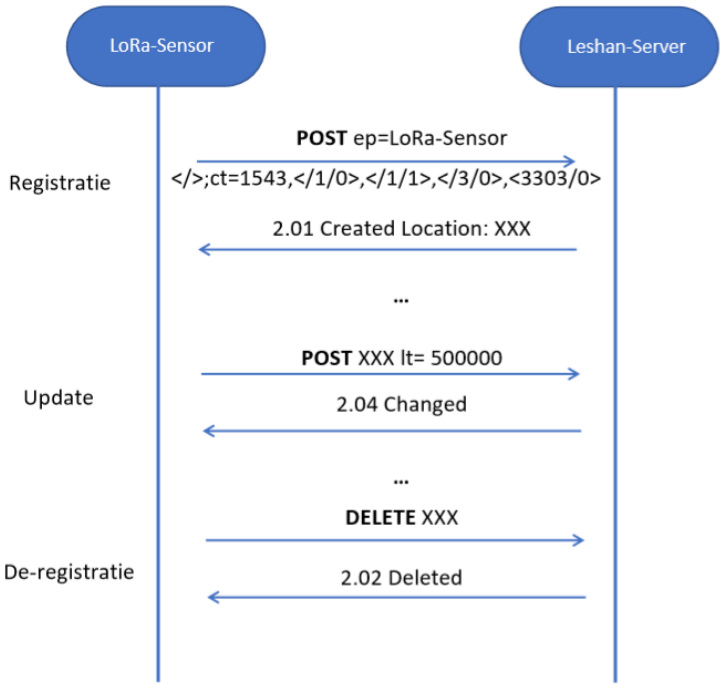
De volgende secties van dit verslag gaan dieper in op de details van zowel werking als implementatie van de verschillende componenten, gebruik makende van figuur 1 als aanknopingspunt om de context te duiden.

4.1 LwM2M-netwerk

Het LwM2M-netwerk bestaat uit een combinatie van sensoren en verbindingen met een LwM2M-server. De verzamelde data worden verstuurd via het LwM2M-protocol naar een LwM2M-server, die deze data dan verder kan sturen naar het Solid systeem.

4.1.1 Het LwM2M-protocol

Om het LwM2M-netwerk te kunnen implementeren is het belangrijk dat we de werking van het onderliggende protocol goed begrijpen. Daarom bespreken we eerst hoe de communicatie tussen de sensoren en de LwM2M-server verloopt. Dit doen we aan de hand van de LoRa-sensor die in het volgende deel verder wordt besproken.



Figuur 2: Communicatie LoRa-sensor met LwM2M-server.
Gebaseerd op: [9].

Figuur 2 toont hoe de LoRa-sensor uit het volgende deel zich registreert, ververst en uiteindelijk deregistreert op de LwM2M-server. De registratie gebeurt met een POST bericht via CoAP ('Constrained Application Protocol'). Dit protocol vervult een zeer gelijkaardige functie als HTTP, maar is meer geschikt voor apparaten met weinig middelen ter beschikking [10].

Dit POST-bericht bevat verschillende elementen:

- ep=LoRa-Sensor: de naam van de node waarmee de sensor zich registreert.
- $< / >; ct = 1543$: het bestandsformaat waarmee gegevens verstuurd worden, i.c. JSON.
- $< /1/0 >, < /1/1 >, < /3/0 >, < 3303/0 >$: lijst met beschikbare objecten en instanties ervan. In dit voorbeeld zijn het de verplichte objecten 1 (server), 3 (device) en het temperatuurobject 3303. Zie ook sectie 4.2.2 voor een meer uitgebreide uitleg over de datastructuur binnen LwM2M.

Na ontvangst van dit bericht stuurt de server een bevestiging van de registratie met de locatie waar de sensor geregistreerd staat. Deze locatie heeft de sensor nodig om later de gegevens op de server te verversen en om zich te deregistreren. Het opvragen van gegevens door de server gebeurt op gelijkaardige wijze via de OBSERVE, GET, DELETE, etc. instructies.

Meer gedetailleerde informatie omtrent het LwM2M-protocol is terug te vinden in de specificaties [9].

4.1.2 LoRa-netwerk

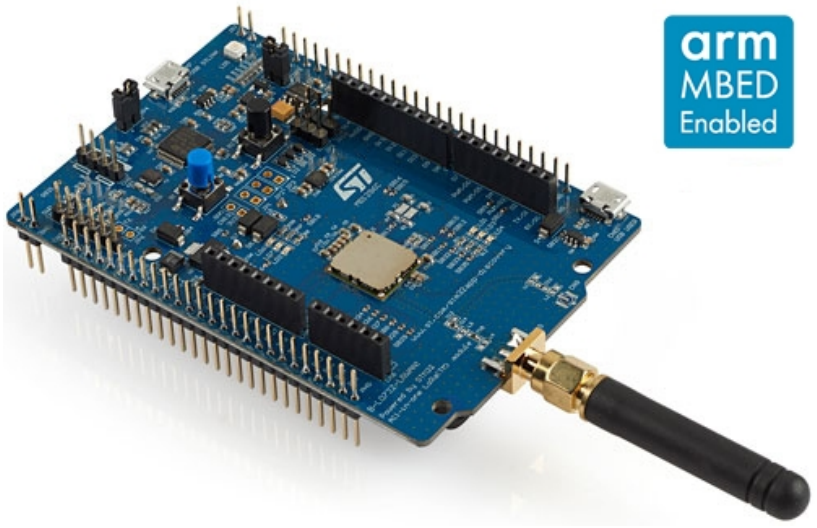
Een van de manieren waarmee gegevens kunnen verzameld worden is via LoRa ('Long Range communication'). LoRa is een gepatenteerde draadloze communicatietechnologie van Semtech waarbij het open LoRaWAN ('Long Range Wide Area Network') protocol wordt gebruikt om berichten uit te wisselen. LoRa kan gesitueerd worden op de fysieke laag, terwijl LoRaWAN op de data link laag werkt (MAC ('Media Access Control')).

Deze vorm van communicatie is zeer geschikt voor toestellen met een laag energieverbruik, dus zeker voor M2M communicatie.

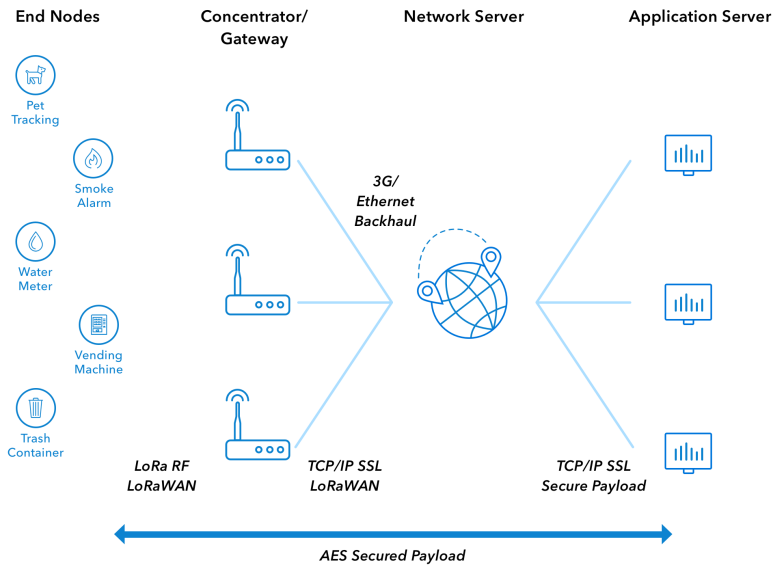
Het gebruik van LoRaWAN laat communicatie van IoT toestellen toe op lange afstand (meerdere kilometers). Deze communicatie werkt via transmissies, op een bepaalde frequentie afhankelijk van werelddeel. Deze berichten worden dus in elke richting uitgezonden. In Europa is deze frequentie 868 MHz. Als er een LoRa ontvanger in de buurt is zal dit bericht opgevangen worden en verder verwerkt worden. Meer informatie hierover bij TheThingsNetwork (zie 4.1.2.2).

4.1.2.1 LoRa-sensor De eerste component van het LoRa-netwerk die we bespreken is de LoRa-sensor. Deze is verantwoordelijk voor het verzamelen van gegevens. Als ontwikkelingsomgeving maken we gebruik van de *b_l072z_lrwan1-LoRa discovery kit* van STMicroelectronics (figuur 3) in combinatie met het Zephyr besturingssysteem, ontwikkeld in C/C++ [11][12].

De voorbeeldcode voor de LwM2M-client die met Zephyr meegeleverd wordt is voor onze toepassing onbruikbaar gezien het RAM-gebruik hiervan te hoog ligt. Zo heeft deze implementatie 40kB werkgeheugen nodig terwijl slechts 20kB beschikbaar is. Met behulp van de Zephyr-modules voor LwM2M is het mogelijk een eigen implementatie te ontwikkelen.



Figuur 3: De LoRa Sensor (b_l072z_lrwan1).
Bron: [11].



Figuur 4: LoRaWAN Architectuur.
Bron: [13].

Om zich te kunnen registreren bij de server, moet de sensor het adres ervan kennen. Via de LwM2M-module worden de objecten en instanties aangemaakt samen met callbackfuncties. Deze worden opgeroepen bij vraag naar de instantie of na een bepaalde tijd. Zo zal de temperatuur-callbackfunctie de temperatuur meten en de waarden van de instantie updaten wanneer de server hierom verzoekt.

De LwM2M-berichten worden verstuurd via LoRaWAN, ook hiervoor bevat het Zephyrpakket een module. De berichten worden op een bepaalde frequentie verstuurd met daarbij een ID en sleutel waarmee het geregistreerd staat bij TheThingsNetwork. Meer uitleg hierover in onderstaande sectie.

4.1.2.2 TheThingsNetwork Het volgende deel van de LoRa-netwerkketting is TheThingsNetwork (TTN). Dit is een organisatie die open tools en een globaal open netwerk aan LoRa-ontvangers ter beschikking stelt voor IoT toepassingen. Dit netwerk werkt via LoRaWAN, een specificatie voor M2M-communicatie op lange afstand, en het internet zelf.

TTN maakt voornamelijk gebruik van twee soorten berichten:

- Uplink: een bericht dat verstuurd wordt door een end node naar de TTN server.
- Downlink: een bericht dat verstuurd wordt door de TTN server naar een end node.

Andere types berichten dienen ter ondersteuning zoals de acknowledgement (ACK) en error (ERR) berichten.

De sensor doet een uitzending (uplink) van een LwM2M-bericht via zijn antenne op de frequentie 868 MHz (Europa). Dit bericht wordt door alle LoRa-gateways in de omgeving ($\pm 10km$ bereik) ontvangen en doorgestuurd naar servers van TTN. Onderweg wordt er ook extra informatie aan het bericht toegevoegd: welke gateway het bericht ontvangt, tijdstip van ontvangst, etc.

Het ontvangen bericht wordt gesorteerd via de volgende waarden die meegegeven worden in de LoRa uitzendingen:

- Device EUI: een 8 byte code die het apparaat identificeert.
- Application EUI: een 8 byte code die de applicatie van het apparaat identificeert. Deze wijst bij ons naar "*iot_and_solid*", de naam van de applicatie op TTN.
- App Key: een 16 byte code die als sleutel dient voor beveiliging.

Bij het verzenden van een bericht door de sensor kan het gebeuren dat deze door meerdere gateways wordt ontvangen. TTN selecteert dan de dichtstbijzijnde gateway om later berichten zo efficiënt mogelijk te kunnen terugsturen (downlink). Aangezien de randapparaten verbonden met TTN meestal maar voor een kleine periode actief zijn, zal TTN alle downlink berichten in een wachtrij zetten en pas doorsturen als de sensor terug online komt.

De bespreking hierboven behandelt de normale werking van de sensor zoals initieel voorzien was, wegens de COVID-19-maatregelen verschilt de uiteindelijke implementatie echter. Gezien op de thuislocatie geen TTN-ontvangers in de omgeving beschikbaar waren, werd een virtuele sensor geëmuleerd m.b.v. QEMU [14]. Om de reële werking zo goed als mogelijk te simuleren, worden de uitgezonden berichten door een Python script onderschept en over internet aan TTN afgeleverd. Zie ook bijlage 8.2 voor meer informatie over de werking hiervan.

Eens de berichten op het TTN-platform toekomen, kunnen deze op verschillende manieren verwerkt worden. Zo is het mogelijk om de ingebouwde decoder voor de ontvangen bytes aan te passen om deze naar wens te verwerken. TTN biedt verschillende integraties aan om de ontvangen data verder door te sturen, op te slaan, te mappen, etc. Integratie van eigen applicaties is mogelijk door gebruik van een van de aangeboden handler API's, waaronder de MQTT data API die in dit project gebruikt wordt.

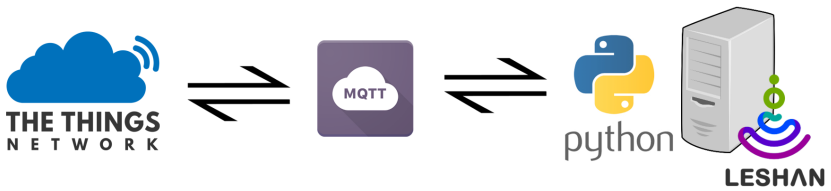
4.1.2.3 MQTT MQTT, staande voor 'Message Queue Telemetry Transport', is een M2M communicatieprotocol dat vooral gebruikt wordt wegens de geringe omvang van de code en het publish-subscribe systeem [15].

Hierbij is er in tegenstelling tot het client-server systeem een derde component (de broker) waarlangs alle verbindingen gaan. Dit levert enkele voordelen voor de client en de server:

1. Ze moeten elkaar niet kennen, enkel de broker (ip + poort).
2. Ze moeten niet op hetzelfde moment actief zijn.
3. Ze kunnen asynchroon werken: moeten niet actief wachten op berichten, waardoor ze ondertussen iets anders kunnen doen.

Hierdoor is het ideaal voor toestellen met een beperkt geheugen en actieve tijd zoals IoT toestellen.

De rol van MQTT in ons project is om de data van TTN door te sturen naar de LwM2M-server. Concreet maken we hiertoe gebruik van een Python script dat m.b.v. een MQTT-bibliotheek verbinding maakt met de MQTT broker inbegrepen in het TTN-platform. Dit proces wordt gevisualiseerd in figuur 5 [16].



Figuur 5: Verbinding tussen TTN en de LwM2M-server via MQTT.

De volgende waarden zijn nodig om een beveiligde verbinding te kunnen opstarten:

1. App ID: "iot_and_solid" De naam van de applicatie op TTN.
2. Access Key: "ttn-account-v2.XXXXXXXXXXXXX..." De sleutel gebruikt om toegang te krijgen tot de data van de applicatie van TTN. Je kan op de TTN website zelf de bevoegdheden van de sleutel instellen.
3. Device ID: "the_blue_board" De naam van de end node (sensor) die data naar TTN verstuurd. Deze waarde is optioneel, zeker wanneer je maar één node aan je applicatie hebt gekoppeld.

Eens de MQTT-client is gekoppeld aan TTN, moeten er nog callback-functies ingesteld worden zodat de handler weet wat hij moet doen in het geval dat de connectie gestart wordt, TTN data van een sensor ontvangt (uplink) en de connectie afgesloten wordt.

In deze situatie moest de uplink callback-functie ingesteld worden zodat de data ontvangen van TTN eerst gedecodeerd worden naar het originele CoAP bericht en dat dit bericht naar de LwM2M-server wordt doorgestuurd. Deze LwM2M-server bevindt zich in hetzelfde netwerk als het apparaat waarop het Python script draait (dit kan hetzelfde apparaat zijn als de LwM2M-server).

Om berichten terug te sturen van de LwM2M-server naar de sensor kan er gebruikgemaakt worden van de MQTT downlink functie.

Het probleem bij MQTT is de schaalbaarheid. De Leshan server heeft per sensor een apart IP-adres + poort nodig. Deze eis zorgt ervoor dat we voor elke sensor uit het LoRa-netwerk een afzonderlijke verbinding moeten opstellen met de LwM2M-server.

Een voorlopige oplossing voor dit probleem zou zijn om per sensor gebruik te maken van een unieke poort, hetzelfde principe als een NAT. Dit zorgt voor een groter, maar nog steeds beperkt, aantal mogelijke sensoren.

Deze eis van de Leshan server is in het algemeen niet vereist bij LwM2M-servers, er moet gewoon een uniek label zijn per client.

Een andere oplossing is het gebruik van SCHC [17] ('Static Context Header Compression'). In dit protocol zijn er twee extra lagen tussen de IPv6 en LPWAN laag: SCHC Compressie en SCHC fragmentatie. Het gemaakte SCHC pakket bestaat uit een compressie header en de payload. Aan de hand van deze header zouden we de berichten met hetzelfde IP-adres toch nog kunnen verdelen over verschillende clients waardoor er maar één verbinding met MQTT nodig is.

4.1.3 IEEE 802.15.4 netwerk

Naast LoRa wordt in dit project ook gebruikgemaakt van een tweede technologie voor draadloze communicatie. De sensoren en server communiceren in dit geval via de IEEE 802.15.4 standaard.

4.1.3.1 IEEE 802.15.4 IEEE ('Institute of Electrical and Electronics Engineers') 802.15.4 is een in 2003 vastgelegde standaard voor snelle communicatie tussen netwerkelementen met een laag vermogenverbruik (bv. sensoren).

Er is een gigantische verscheidenheid aan apparaten met draadloze communicatie op de markt. IEEE 802.15.4 wil hierbinnen een basis vormen voor hoe deze systemen op elkaar afgestemd worden en op een simpele manier met elkaar kunnen communiceren. De standaard is ontwikkeld om zeer goede prestaties af te leveren bij het versturen van simpele, kleine datapakketten. Ze zorgt voor een hoge accuraatheid, weinig vertraging en een goede duurzaamheid binnen een netwerk van laagverbruikende toestellen.

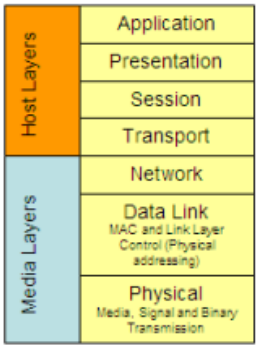
De basis van IEEE 802.15.4 is het vastleggen van de fysieke laag uit het OSI model. Elke node heeft een PSK ('Phase Shift Keying') zendontvanger die transmissiesnelheden kan halen tot 250 kbps en opereert binnen vastgelegde frequentiebanden.

Communicatie gebeurt via het zenden en ontvangen van frames die allemaal voldoen aan gestandaardiseerde vormen. Daarnaast zijn er ook voorzieningen binnen de MAC-laag ('Medium Acces Control').

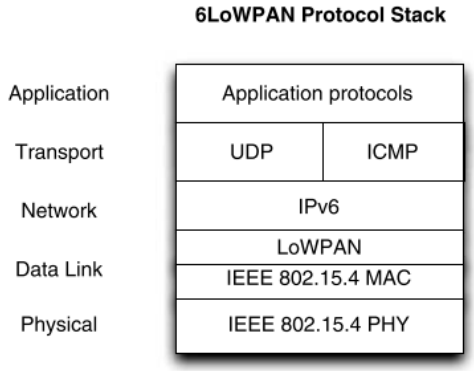
Dit biedt de mogelijkheid aan apparaten om een knoop binnen een netwerk te vormen en te synchroniseren met andere knopen. Apparaten kunnen 4 verschillende soorten frames uitsturen, waarvan een dataframe er slechts 1 is. Zo zijn er ook bijvoorbeeld 'acknowledgment' frames, waarmee voor een betrouwbare connectie tussen gesynchroniseerde knopen gezorgd wordt, en 'beacon' frames, waarmee een apparaat zichzelf kan outen als knoop en toetreden tot het netwerk.

Er zijn verschillende bedrijven die chips produceren die werken binnen deze IEEE 802.15.4 standaard. Figuur 6 geeft een visueel overzicht van de leegtes die IEEE 802.15.4 invult: het voorziet een fysisch medium (chip) waarvan de fysische werking volgens de standaard vastgelegd is (frequentie, zendontvanger,...) en binnen MAC laag is vastgelegd hoe data tussen apparaten rondgestuurd wordt zoals de opbouw van een data frame, de tijdssloten waarbinnen data en acknowledgements verwacht worden toe te komen en het toetreden van een apparaat tot de groep knopen.

Hogere lagen in de OSI-netwerkstapel zijn nog door de gebruiker zelf in te vullen door gebruik te maken van bijkomende standaarden.



(a) Het OSI model, IEEE 802.15.14 legt de onderste 2 lagen ondubbelzinnig vast.
Bron: [18].



(b) Functionaliteiten vastgelegd door IEEE 802.15.4. 6LoWPAN zorgt voor de link naar de hogergelegen netwerklaag.
Bron: [19].

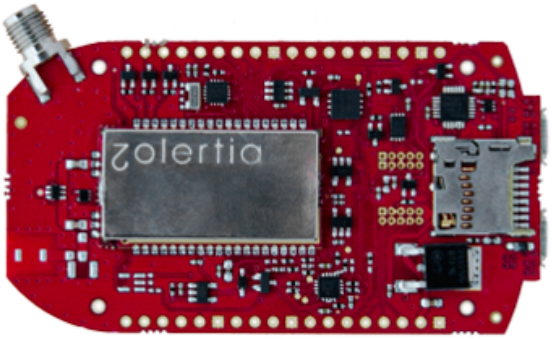
Figuur 6: Situering van IEEE 802.15.4 in de OSI netwerkstapel.

Het is belangrijk om vast te stellen dat IEEE 802.15.4 de netwerklaag niet vastlegt. Het biedt de mogelijkheid aan apparaten om met elkaar te verbinden en peer to peer data in milliseconden uit te wisselen, maar er wordt niet aan routing en andere netwerklaag-gerelateerde functionaliteiten gedaan. Er zijn wel netwerktechnieken ontwikkeld die niet tot de IEEE 802.15.4 standaard behoren, maar de netwerklaag van de standaard wel vervolledigen. In ons IoT netwerk zullen we gebruik maken van 6LowPAN ('IPv6 over Low-power Wireless Personal Area Network') dat de netwerklaag door gebruik van IPv6 invult.

Tijdens dit project werd duidelijk dat IEEE 802.15.4 verre van een wondermiddel is voor het opstarten van een draadloos IoT netwerk. Het voorziet eerder robuuste betrouwbare hardware die binnen de opgelegde standaard functioneert en een ontwikkelaar met minstens enige voorkennis van IoT netwerken veel mogelijkheden biedt [20][18][21].

4.1.3.2 6LoWPAN De netwerklaag die door IEEE 802.15.4 nog wordt opengelaten, zullen we opvullen door gebruik te maken van 6LoWPAN. Dit staat voor "IPv6 over Low-Power Wireless Personal Area Network", maar is eigenlijk niet meer dan onze weinig verbruikende sensoren IPv6 berichten naar elkaar laten sturen. 6LoWPAN zorgt voor de adaptie tussen het MAC niveau van IEEE 802.15.4 en IPv6 door encapsulatie in de ene richting en headercompressie in de andere richting, waardoor gegevens gericht van zender naar ontvanger kunnen reizen [19][22][23][24].

4.1.3.3 Zolertia RE-mote De hardware die tijdens dit project gebruikt wordt voor ontwikkeling van de IEEE 802.15.4 sensor is de Zolertia RE-mote (figuur 7). Dit is een prototypebord waarop een sensor aangesloten kan worden en dat ontworpen is om van 6LoWPAN gebruik te maken.



Figuur 7: De Zolertia RE-mote.
Bron: [25].

De Zolertia RE-mote wordt geprogrameerd via het Contiki OS besturingssysteem. Dit is open-source en gratis software die specifiek ontwikkeld is voor het programmeren van geheugenbeperkte IoT apparaten met laag vermogenverbruik. Contiki maakt het mogelijk om toepassingen zoals LwM2M te laten lopen op onze prototypeborden.

Via een IPv6 communicatiestapel zijn apparaten verbonden met het internet en Contiki voorziet een zeer grote verscheidenheid aan programma's (in C) en bijhorende makefiles waarmee de Zolertia RE-mote geprogrammeerd kan worden om bepaald gedrag te realiseren. Dit kan gaan van een MQTT client (zie sectie 4.1.2.3) tot het zuiver doorsturen van inkomende pakketten naar een server (router gedrag).

Binnen dit project wordt gebruikgemaakt van een virtuele Linux desktop genaamd Instant Contiki, die door Contiki zelf is ontwikkeld en reeds een basis voor alle nodige programma's bevat [26][27][28].

4.1.3.4 Realisatie van het netwerk Zoals afgebeeld in figuur 1, bestaat het IEEE 802.15.4 netwerk dat gerealiseerd moet worden uit 2 verschillende bouwstenen. De sensor moet meetwaarden uitsturen die een border-router moet ontvangen en op zijn beurt correct moet doorsturen naar de LwM2M-server. Beide knopen van het netwerk worden afzonderlijk gerealiseerd door een Zolertia RE-mote, zij het telkens op een andere manier geprogrammeerd. De border-router zorgt voor de connectie tussen het lokale sensornetwerk (6LoWPAN) en het publieke internet.

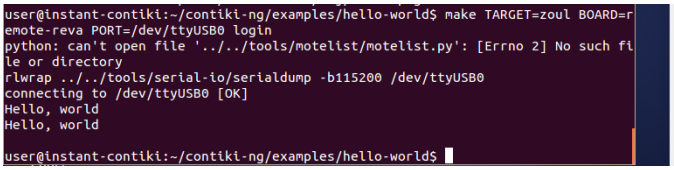
Tijdens dit project werd duidelijk dat het programmeren van een Zolertia RE-mote allesbehalve een sinecure is en dat dit een proces met vallen en opstaan is. Na eerst een wat naïeve poging te ondernemen door de Contiki bestanden op een Windows desktop te installeren werd overgeschakeld naar de virtuele Linux desktop Instant Contiki. Een nieuwe uitdaging hier is het correct bereiken van de USB-poorten waarop de Zolertia RE-mote bordjes aangesloten waren. Hiervoor moet de virtuele Linux lopen binnen een applicatie die USB-filtering toelaat. Indien de USB-poort waarop het device is aangesloten geïdentificeerd is kan men data naar dit device schrijven via:

```
make TARGET=zoul BOARD=remote-reva
PORT=<naam USB-poort> <locatie make file>.upload
```

We kunnen tevens inloggen op de RE-mote via:

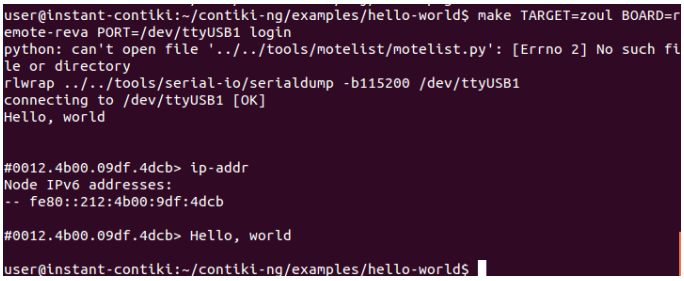
```
make TARGET=zoul BOARD=remote-reva PORT=<naam USB-poort> login
```

Gedurende lange tijd lukte het schrijven van data naar de RE-mote niet. Uiteindelijk is dit opgelost door een veel oudere versie van een ARM-compiler te installeren. Deze compiler vertaalt de C-programma's uit de Contiki bestanden naar assembly code, die de processor van de RE-mote kan begrijpen. De ARM-compiler die oorspronkelijk geïnstalleerd werd op de Linux desktop was een recente versie uit 2019. Instant Contiki beschikt echter over het verouderde Ubuntu 12.04 en de ARM-compiler uit 2019 werkt niet voor deze versie. Na het installeren van de juiste compiler, uploaden van data en inloggen op de RE-mote werd een simpel hello-world programma succesvol geflasht (zie figuur 8).



Figuur 8: De hello-world boodschap uitgestuurd door de RE-mote.

Indien de standaard makefile van het Contiki pakket licht aangepast wordt door een shell module toe te voegen, kan er na inloggen op de RE-mote ook bepaalde informatie opgevraagd worden aan het device zoals het IPv6 adres en mogelijke IPv6 buren (zie figuur 9).



Figuur 9: Het IP-adres opvragen van de RE-mote.

Één van de twee RE-motes werd als border-router geprogrammeerd door het rpl-border-router programma op het device te uploaden. Dit C-bestand is te vinden onder *contiki-ng/examples/rpl-border-router*. Eerst wordt de code gecompileerd via

```
make TARGET=zoul BOARD=remote-reva PORT=<naam USB-poort> border-router
```

en dan geflashed naar het device via

```
make TARGET=zoul BOARD=remote-reva PORT=<naam USB-poort> border-router.upload
```

De Zolertia RE-mote is nu als border-router geprogrammeerd.

De border-router wordt uit de PC gehaald en de tweede RE-mote wordt ingeplugd om deze eveneens op een correcte manier te programmeren. De tweede RE-mote wordt als LwM2M-object geprogrammeerd waardoor deze node sensordata van ingeplugde sensoren op de RE-mote kan doorsturen. Het nodige C-programma is te vinden onder *contiki-ng/examples/lwm2m-ipso-objects/example-ipso-objects*.

Opdat de sensordata de juiste LwM2M-server zouden bereiken, moet het standaard `fd00::1` adres voor de LwM2M-server veranderd worden in het IPv6-adres van de LwM2M-server.

Analoog als bij het vorige device moet ook hier de code eerst gecompileerd worden alvorens deze te flashen naar de RE-mote.

Indien de border-router en vervolgens het LwM2M-object nu opnieuw in de pc ingeplugd worden, detecteert de border-router het LwM2M-object als client binnen zijn 6LoWPAN-netwerk en zorgt ervoor dat de paketten naar de LwM2M-server doorgestuurd worden.

De praktische werking van dit netwerk is niet getest kunnen worden door de uitzonderlijke omstandigheden van dit semester. Het verkrijgen van een sensor die op een RE-mote geïnstalleerd kan worden bleek niet mogelijk. Tevens bleek een van de twee RE-mote geen correcte werking meer te hebben, waardoor een opstelling met een gesimuleerde sensor eveneens niet mogelijk was. De werkende RE-mote werd echter succesvol eens als border-router en eens als LwM2M-object geprogrammeerd, waardoor correcte werking enkel werd verhinderd door gebrek aan toegang tot materiaal wegens de COVID-19 lockdown omstandigheden.

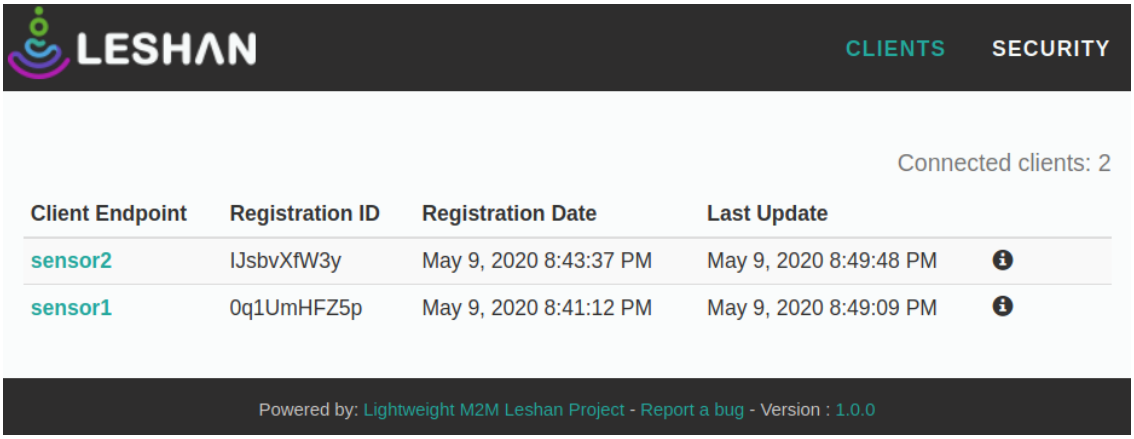
4.1.4 LwM2M-server

De LwM2M-server implementatie die voor dit project wordt gebruikt is een open-source project genaamd Leshan[29]. De projectomschrijving is als volgt: 'Leshan voorziet in bibliotheken opdat mensen hun eigen LwM2M-server kunnen schrijven'.

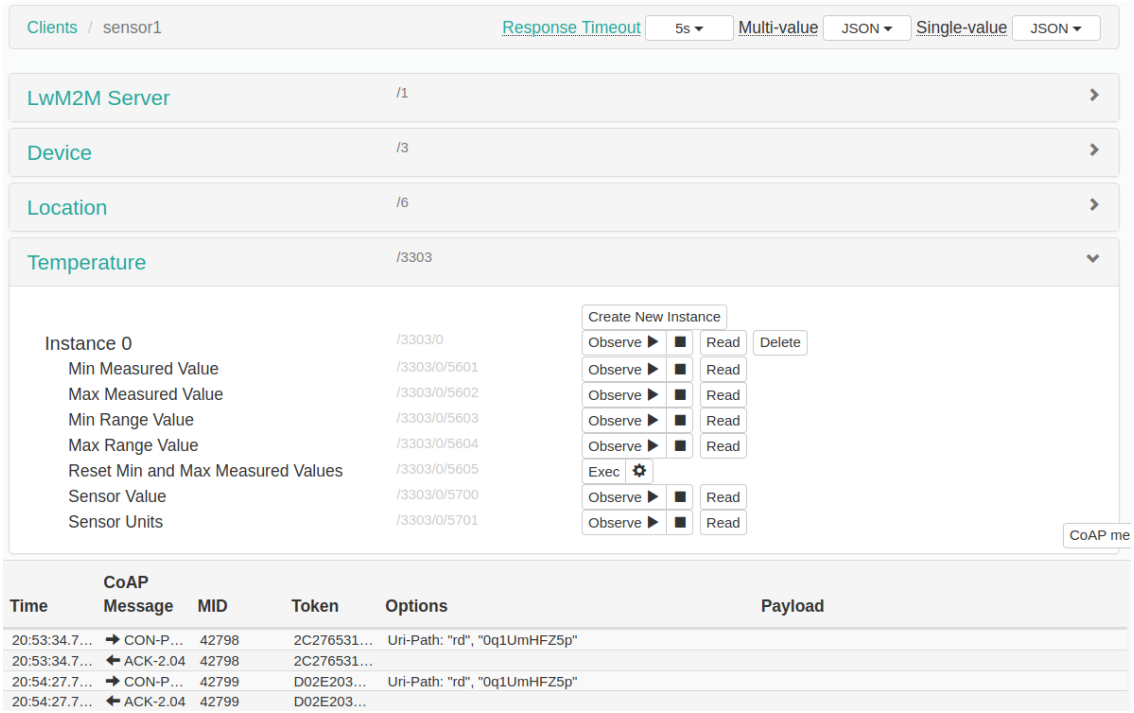
Het project voorziet echter ook een demoserver, die als basis dient voor de server ontwikkeld in dit project. Er is ook een democlient voorzien, die zeer nuttig is gebleken tijdens het ontwikkelen en testen van ons netwerk (zie 5).

De Leshan server is een Java applicatie dat als centraal deel in het demonstratienetwerk de communicatie verzorgt met de IoT sensoren via het LwM2M-protocol en de gebruiker. De communicatie met de gebruiker gebeurt (voor de demoserver) via de gebruikersinterface: een webpagina (HTTP) beschikbaar op poort 8080 tijdens het ontwikkelingsproces. De communicatie met de IoT sensoren gebeurt via CoAP op poort 5683 of voor CoAPS op poort 5684.

Interface De gebruikersinterface is zichtbaar in figuur 10. In 10a zien we de lijst van geregistreerde sensoren op onze server. Als we daarvan één aanklikken, komen we in 10b terecht. Hier zien we de LwM2M-objecten aanwezig in het apparaat, namelijk van boven naar onder: het verplichte server object, het verplichte device object, een locatie object en als laatste een temperatuur object. Dit laatste is opengevouwen, met daarin zichtbaar dat we één instantie hebben met enkele resources, zoals 'Sensor Value'. Helemaal onderaan de pagina hebben we nog een surplus, namelijk een log van de verzonden en ontvangen CoAP berichten.



(a) Lijst van geregistreerde sensoren.



(b) Interface van de sensor.

Figuur 10: Leshan (demo) server interface.

Bij gebrek aan documentatie zat er, om de werking van Leshan te achterhalen, niets anders op dan in de codebase te duiken. Aanvankelijk leek het erop dat een module voor het doorsturen van data naar externe applicaties via een API geschreven zou moeten worden, maar al snel bleek dat weinig aanpassingen nodig waren om de gewenste functionaliteit te verkrijgen.

Zie bijlage 8.3 voor een beknopte uitleg van de werking van de Leshan demoserver.

Het was dus mogelijk de demoserver te hergebruiken, mits een aanpassing van een tiental lijnen code [30]. Ondertussen is deze aanpassing ook opgenomen in de officiële demoserver.

Deze server kan op eender welke hardware draaien met behulp van Java; voor het demonstratienetwerk werd een Raspberry Pi gekozen. Daarenboven kunnen de functionaliteiten die onmiddellijk aansluiten aan de Leshan server in het demonstratienetwerk

(figuur 1), meer bepaald de MQTT-ontvanger en de vertaalapplicatie, ook op hetzelfde apparaat uitgevoerd worden.

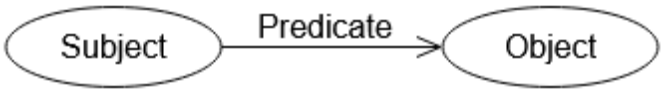
De samenwerking die hierrond nodig was, is nog één van de zaken die door de uitbraak van het Coronavirus bemoeilijkt werden. Een mooie oplossing werd echter ontwikkeld: door een SSH-verbinding met de Raspberry Pi open te stellen voor de medewerkers kon iedereen er op elk moment op werken zonder fysieke toegang nodig te hebben.

4.2 Solid systeem

4.2.1 Linked data en RDF

Solid (Social Linked Data) is gebouwd op de fundamenten van linked data, een beschrijvingsvorm die het mogelijk maakt gedecentraliseerde data aan elkaar te relateren of linken [31].

RDF ('Resource Description Framework') is een gestandaardiseerd formaat voor de beschrijving van linked data, waarin datasets als gerichte grafen worden voorgesteld. De knopen (IRIs, literals of blanco's) worden met elkaar gerelateerd d.m.v. predikaten, die van subject naar object wijzen (figuur 11). De combinatie (subject, predicaat, object) wordt in RDF een triple genoemd [32].



Figuur 11: Onderdelen van een RDF graaf.
Bron: [32, Fig.1].

Om concrete data in RDF te beschrijven maakt men gebruik van ontologieën/vocabularia. Dit zijn een soort lexica die beschrijven welke subjecten en objecten beschikbaar zijn en op welke manier ze door de aangegeven predikaten kunnen worden gerelateerd. Een voorbeeld is de FOAF ('Friend Of A Friend') ontologie, waarmee relaties tussen en eigenschappen van personen kunnen worden beschreven. Hieronder wordt een voorbeeld gegeven van een geldige RDF triple in FOAF [33].

```
@base <https://www.example.com/> .
@prefix foaf: <https://xmlns.com/foaf/0.1/> .
<JohnDoe> foaf:knows <JaneDoe> .
```

In dit voorbeeld is `<JohnDoe>` het subject, `<JaneDoe>` het object en `foaf:knows` het predikaat. De syntax waarin bovenstaande triple uitgedrukt is heet Turtle, wat een compacte en leesbare notatie van RDF-grafen toelaat, maar tal van andere formaten als JSON-LD en RDF/XML bestaan ook en bieden hun eigen voordelen [34].

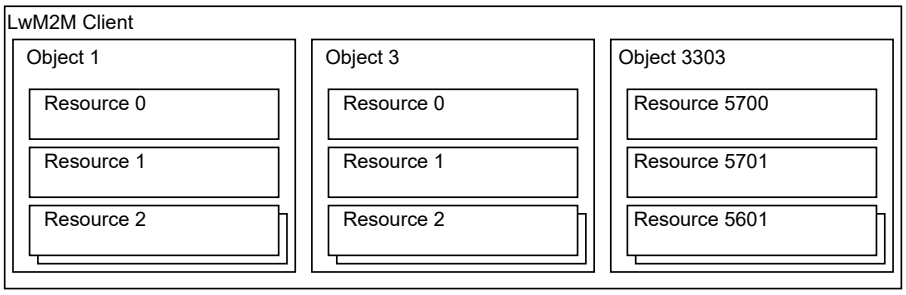
Hoewel de concepten van linked data en RDF op het eerste zicht wat omslachtig kunnen lijken, bieden deze als voordeel dat de opgeslagen data zelfbeschrijvend is en dat meer relaties dan in een hiërarchische boomstructuur toegelaten worden. Dit heeft als resultaat dat linked data door computers kan worden onderzocht zonder van een bepaalde structuur uit te moeten gaan, gezien de context in de data zelf vervat zit.

4.2.2 De LwM2M-ontologie

De LwM2M-specificatie beschrijft o.a. hoe de data binnen de LwM2M-standaard gestructureerd wordt. Zoals weergegeven in figuur 12 bevat een client een aantal objects die op hun beurt uit resources bestaan. Meer specifiek bevat de weergegeven client 3 objects:

- Object 1: Server object (verplicht).
- Object 2: Device object (verplicht).
- Object 3303: Temperatuurobject (vrijblijvend), met resources:
  - Resource 5700: SensorValue.
  - Resource 5701: SensorUnits.
  - Resource 5601: MinMeasuredValue.

De volledige lijst aan beschikbare objects en resources samen met hun IDs en types zijn te vinden in de LwM2M-registry [9][35].



Figuur 12: Datastructuur uit de LwM2M-specificatie.
Aangepast uit: [9, Fig.7.1].

In het kader van een onderzoek voor de Europese commissie werd reeds een basis gelegd aan een ontologie die aan de LwM2M-specificaties voldoet. Hierin bleken echter behoorlijk wat zaken te ontbreken, voornamelijk de predikaten die de relaties tussen de verschillende objecten beschrijven, waardoor deze praktisch onbruikbaar was. [36]

In de afweging tussen het behouden van de scope van het project en het nastreven van zo goed mogelijke overeenkomst met de datastructuur zoals beschreven in LwM2M-specificatie, besloten we de bestaande ontologie zelf uit te breiden met de zaken die voor ons project noodzakelijk waren.

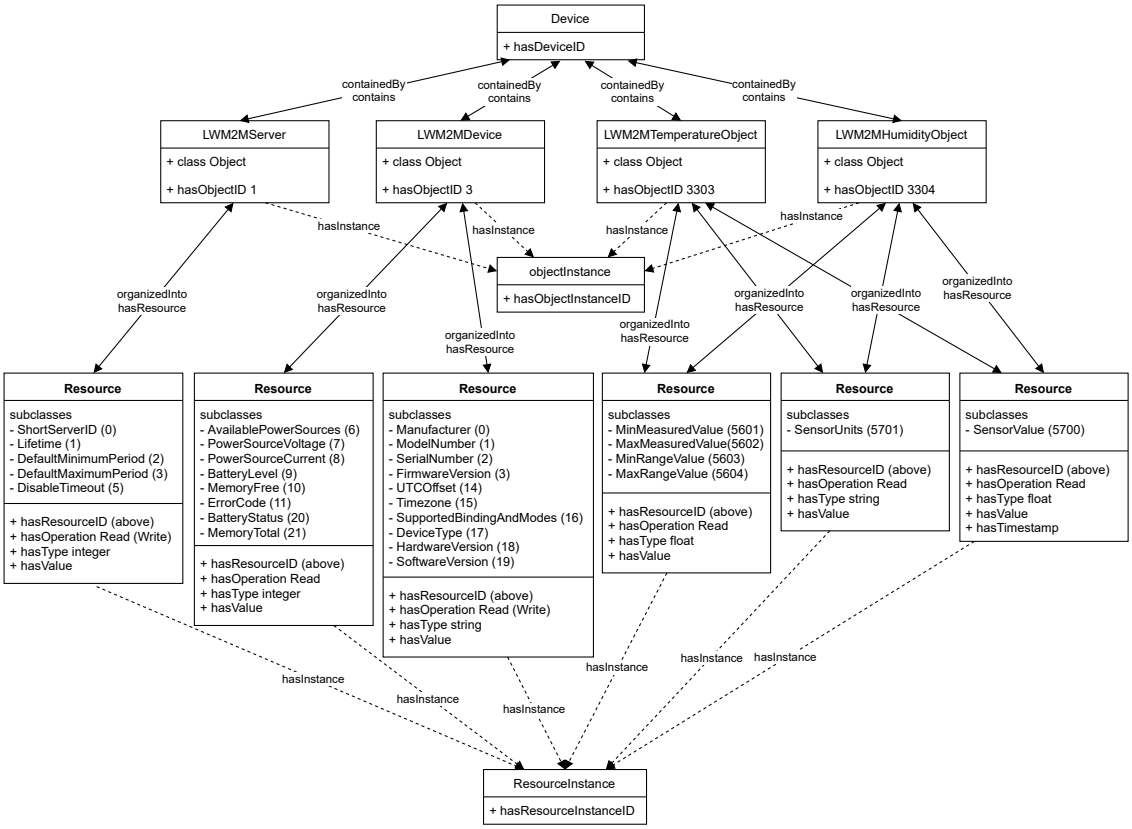
Het voordeel van de gekozen aanpak is dat dit ons de flexibiliteit geeft om elementen naar wens toe te voegen en ook de mogelijkheid biedt om deze in de toekomst uit te breiden en aan te passen om aan de vereisten van andere toepassingen te voldoen.

Gezien het LwM2M-protocol vooral gericht is op device management, communicatie en beveiliging en niet op de gestructureerde opslag van historische data, was het nodig om op twee vlakken de vrijheid te nemen om elementen aan de ontologie toe te voegen die niet strikt in de LwM2M-specificaties beschreven staan.

1. Om de objects waarvan de resources bij eenzelfde LwM2M-client kunnen worden opgevraagd aan elkaar te kunnen relateren, werd een Device subject geïntroduceerd in de ontologie waaraan de objects dan kunnen worden gekoppeld via het predikaat ‘contains’, of ‘containedBy’ in de omgekeerde richting.

2. Om opslag van historische sensor data mogelijk te maken, voegden we aan de resource 'SensorValue' een predikaat 'hasTimestamp' toe waaraan het moment dat de meting uitgevoerd werd, kan worden gekoppeld. Deze oplossing is het meest praktisch binnen de scope van dit project, maar we geven graag mee dat er robuustere oplossingen voor de opslag van historische data in RDF-formaat bestaan [37].

Een visualisatie van de belangrijkste componenten uit de resulterende ontologie wordt gegeven in figuur 13, het volledige resultaat is publiek beschikbaar via de volgende link: https://iotsolidugent.inrupt.net/public/ontologies/omalwm2m.owl.ttl



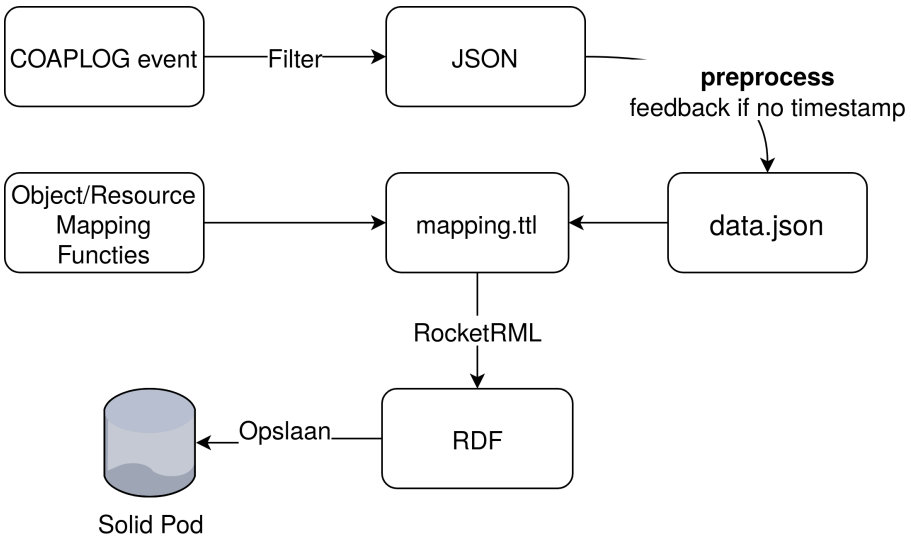
Figuur 13: Visualisatie van de ontologie.

4.2.3 Vertaling en opslag

De vertaallaag is één van de applicaties die zich inschrijft op de Leshan server en de ontvangen data verwerkt en wegschrijft naar de Solid pod. Het vertrekpunt van de vertaallaag is dat deze passief wacht totdat er meetgegevens binnenkomen.

Solid authenticatie & programmeertaal Voor de interactie met de Solid pod wordt er gebruikgemaakt van een derde-partij bibliotheek waarin deze functionaliteit voorzien is. Deze bibliotheek, 'solid-auth-cli' [38], heeft geen browser nodig, in tegenstelling tot de officiële bibliotheek[39] [40]. Het vergde echter wel wat inspanning [41] om dit aan de praat te krijgen.

Het vinden van de juiste interactie-tools met Solid bepaalde ook de programmeertaal waarin de vertaallaag zou worden geschreven, namelijk JavaScript. Een complexer samenspel van JavaScript en andere componenten had mogelijk geweest, maar alle benodigde



Figuur 14: Vertaallaag: schema interne werking.

tools voor de vertaling waren aanwezig in JavaScript bibliotheken, dus werd ervoor gekozen om er één geheel van te maken. Dit geheel is ontwikkeld als een Node.js applicatie [42] en kan dus als server op eender welke omgeving draaien met behulp van Node. Node.js is een runtime-omgeving die gebruikt kan worden om JavaScript-toepassingen buiten de browser te draaien.

De flow gaat als volgt: eerst worden alle nodige voorbereidingen gedaan, pas daarna registreert de vertaallaag zich bij Leshan.

Voorbereiding & gegevensopslaglocatie De voorbereidingen omvatten: inloggen op de Solid pod, het inlezen van de LwM2M-ontologie van het internet en het inlezen van de configuratiebestanden op de Node server zelf. Het inloggen op de solid pod gebeurt dus via de 'solid-auth-cli' bibliotheek, maar eens ingelogd, moet ook nog het juiste bestand gevonden worden om de meetgegevens in op te slaan.

Dit is een vrij belangrijk probleem dat op verschillende manieren aangepakt kan worden. De details worden meegegeven in appendix 8.4, maar het voornaamste is dat de locatie van de meetgegevensopslag beschikbaar is en ernaar geschreven kan worden.

Registratie & events Nadat alle voorbereidingen zijn voltooid, wordt er een registratie naar de Leshan server gestuurd en daarna gewacht totdat er een event binnenkomt. Zoals in 8.3 is uitgelegd, verstuurt Leshan verschillende types events. De vertaallaag luistert enkel naar 'COAPLOG' events, maar omdat deze alle CoAP-pakketten gewoon doorstuurt moet er gefilterd worden op de gewenste pakketten met nuttige meetgegevens. Initiëel werd er geluisterd naar 'NOTIFICATION' events, maar zoals ook in 8.3 is uitgelegd, zijn hier enkele moeilijkheden mee.

Deze meetgegevens worden dan omgezet en opgeslagen. Het overzicht van de bewerkingen op een binnengekomen 'COAPLOG' event wordt op figuur 14 weergegeven. Het omzetten gebeurt als volgt:

Omzetting Het doel is om de ruwe data in JSON formaat (zie bv. listing 2 en 3) om te zetten naar linked data volgens de LwM2M-ontologie.

In de data meegeleverd met het event zit onder andere de payload (zie listing 3), dewelke de meetgegevens bevat in SenML [43]. Dit werd vastgelegd in [44, par 6.4.4 JSON] en handelt over het gebruik van het Media Type 'application/vnd.oma.lwm2m+json' (het enige type dat ondersteund is in ons project).

Voor deze omzetting werden door de begeleiders van dit project enkele aanknopingspunten aangereikt [45] [46] [47] [48].

De technologie die het meest concreet werkbaar leek voor het huidige project was de RDF Mapping Language (RML) [48] [49] [50]. Volgende zaken pleitten in haar voordeel: een implementatie in Java (RMLMapper [48]) die toeliet de eerste testen te doen, duidelijke voorbeelden over het werkingsprincipe van RML en vooral een implementatie in JavaScript (RocketRML [51] [52]).

Werkingsprincipe RML RML is een ontologie die beschrijft hoe data zonder context op RDF moeten worden afgebeeld. Het bestaat uit één of meerdere 'TriplesMap's die volgende predicaten kunnen hebben (zie ook listing 1):

- `rml:logicalSource` om aan te duiden waar de data vandaan komen (exact 1). In het project wijst dit naar het JSON object (onder `DATASOURCE`). Een voorbeeld van dit JSON object is zichtbaar in listing 4.
- `rr:subjectMap` om aan te duiden wat het subject wordt (exact 1). Hier wordt het subject samengesteld op basis van de data meegegeven in het JSON object (onder `DEVICE`).
- `rr:predicateObjectMap` om aan te duiden welke delen van het JSON object op welk RDF-object worden afgebeeld en met welk predicaten (0 of meer). Hier krijgt het device subject een predicaat `lwm2m:hasDeviceID` met als object het ID samengesteld uit de JSON data. Ook krijgt het device subject een predicaat `lwm2m:contains` dat wijst naar een object dat ook gegenereerd wordt door RML (niet getoond in listing 1).

Verder laat een 'FunctionMap' [53] toe om functies mee te geven aan RocketRML en deze te gebruiken om met data uit de JSON aan de slag te gaan en de uitvoer op te nemen in RDF. Hiervan worden er enkele gebruikt, zie ook 8.5.

Omdat de toegekomen data (listing 2) en de data die verwerkt kunnen worden (listing 4) niet overeenkomen, moeten we eerst het toegekomen bericht verrijken met extra gegevens (zoals van welke Leshan server dit bericht komt). Dit gebeurt in de functie `preprocessJSON()`.

Ten slotte wordt het opgestelde mapping.ttl bestand meegegeven aan RocketRML samen met de voorbewerkte JSON data om uiteindelijk de RDF graaf te bekomen.

Enkele verdere uitdagingen bij het ontwikkelen van de vertaallaag worden besproken in 8.5. Verder waren er nog enkele praktische moeilijkheden die opgelost zijn geraakt [54] [55] [56].

```
#### DATASOURCE
:datasource
    rml:source "data.json";
    rml:referenceFormulation ql:JSONPath;
    rml:iterator "$.meas[*]"
    .
#### DEVICE
:DeviceMap a rr:TriplesMap;
  rml:logicalSource :datasource;

  rr:subjectMap [
    rr:template "{^^.leshanServer.protocol}://{^^.leshanServer.domain}/{^^.device.ep}";
    rr:class lwm2m:Device;
  ];

  rr:predicateObjectMap [
    rr:predicate lwm2m:hasDeviceID ;
    rr:objectMap [ rml:reference "^^.device.ep" ; rr:datatype xsd:string ]
  ],
  [
    rr:predicate lwm2m:contains ;
    rr:objectMap [ rr:parentTriplesMap :ObjectInstanceMap ]
  ].
```

Listing 1: Fragment van RDF/turtle bestand gebruikt voor RML mapping.

```
{
 timestamp: 1588281716952,
 incoming: true,
 type: 'NON',
 code: '2.05',
 mId: 16395,
 token: 'C7BD44603DE8DA1A',
 options: 'Content-Format: "application/vnd.oma.lwm2m+json" - Observe: 3691',
 payload: { bn: '/3303/0/5700', e: [ { v: -28.8 } ] },
 ep: 'sensor1'
}
```

Listing 2: JSON data in COAPLOG bericht.

```
{ bn: '/3303/0/',
  e: [{ n: '5601', v: 18 },{ n: '5602', v: 26.2 },
      { n: '5700', v: 18.1 },{ n: '5701', sv: 'cel' }
  ]}
```

Listing 3: alternatieve payload in SenML.

```
{
  leshanServer: { protocol: 'http', domain: 'basisLeshan.com' },
  device: { ep: 'sensor1' },
  meas: [
    {
      object: '3303',
      objectInstance: '0',
      resource: '5700',
      value: -28.8,
      stringvalue: undefined,
      booleanvalue: undefined,
      skolemIRI: '5e549f05-4b7f-4122-8745-e6d20e0ef308'
    }
  ]
}
```

Listing 4: JSON data gebruikt in RocketRML

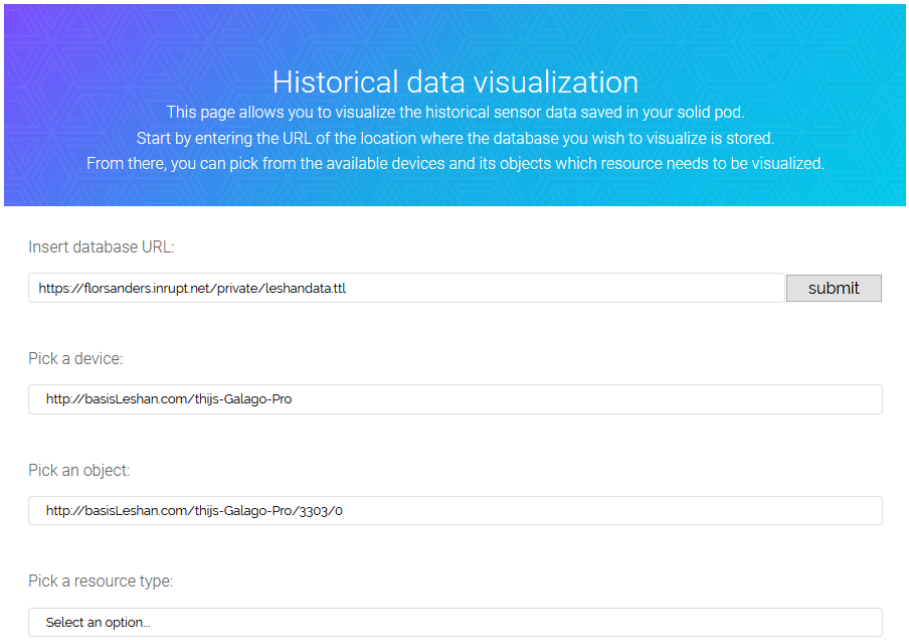
Opslag Als laatste dient de vertaalde data naar de Solid pod te worden geschreven. Dit wordt gedaan met behulp van de updater-tool van rdflib.js [57], die een efficiëntere methode aanbiedt dan telkens het hele document te moeten overschrijven.

4.2.4 Visualisatie

Wat is het nut van data-opslag indien met de opgeslagen data niet aan de slag kan worden gegaan? Met het overduidelijke antwoord op deze retorische vraag in gedachten werkten we een eenvoudige tool uit die toelaat de opgeslagen data op te halen en te visualiseren.

Hiervoor wordt gebruikt gemaakt van de Solid React SDK ('Software Development Kit') en generator ontwikkeld door Inrupt. React is een JavaScript framework om component-gebaseerde gebruikersinterfaces voor webapplicaties te bouwen en wordt ontwikkeld door Facebook. In de SDK werd code ter beschikking gesteld met voorbeelden van enkele eenvoudige solid-toepassingen, waarvan we gebruik konden maken om de visualisatietool uit te werken [58][59].

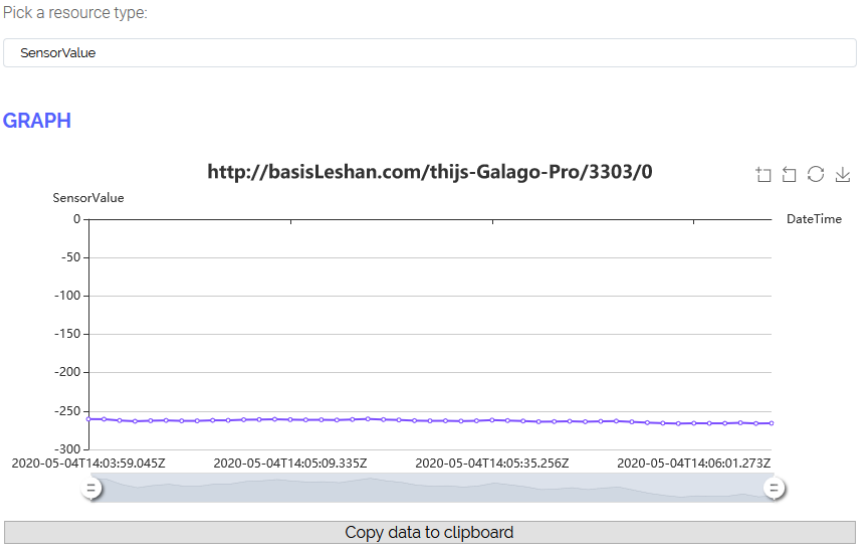
Concreet wordt de werking gerealiseerd door de rdflib.js bibliotheek die de data uit de Solid pod ophaalt, waarna deze stapsgewijs d.m.v. input van de gebruiker (figuur 15) ondervraagd werden op de objects, resources en waarden aanwezig in de pod. [57]



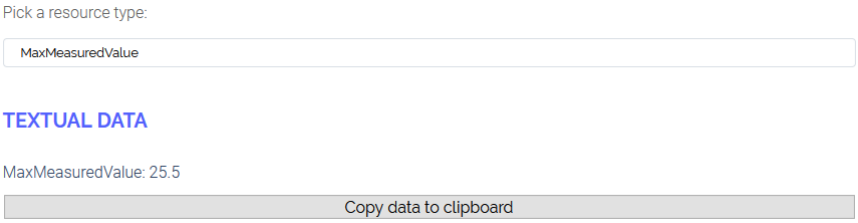
Figuur 15: Selectievelden uit de visualisatietool.

Eens de gebruiker deze selectie heeft gemaakt wordt de eigenlijke data opgevraagd en weergegeven in de gebruikersinterface. Als het gaat om resources waar ook tijdsaanduidingen aan gekoppeld zijn worden deze geplot d.m.v. de ECharts JavaScript-bibliotheek. Ook wordt de optie aangeboden om de pure data in JSON-formaat te kopiëren, zodat deze eventueel in meer geavanceerde programma's kan worden geïmporteerd [60].

Het resultaat van beide types visualisatie wordt weergegeven in figuur 16.



(a) Grafiek voor data die tijdsaanduidingen bevatten.



(b) Tekstuele weergave voor data die geen tijdsaanduidingen bevatten.

Figuur 16: Visualisatievormen van de RDF-data.

5 Testen

Om ervoor te zorgen dat het project efficiënt verliep, werd er parallel aan de verschillende onderdelen gewerkt. Zo konden de testen voor het LoRa-netwerk, IEEE 802.15.4 netwerk, de Leshan server en de vertaallaag alsook de visualisatie apart uitgevoerd worden. De onderdelen werden op het einde van het project samengevoegd om het complete netwerk te vormen.

Bij de opbouw van het LoRa-netwerk werden er op verschillende punten testen uitgevoerd.

1. De LwM2M-client: deze verbinden met een Leshan demoserver op hetzelfde lokale netwerk.

2. TheThingsNetwork (TTN): de verstuurde/ontvangen berichten inspecteren via de online console van TTN.

3. Python bestanden: met een packet sniffing tool (Wireshark) werd de communicatie tussen MQTT en de Leshan server gecontroleerd.

4. De Leshan server op de Raspberry Pi: de LwM2M-client koppelen met de Leshan server via TTN. Deze test toonde of het volledige LoRa-netwerk werkte.

Om het Solid gedeelte te testen werd er gebruikgemaakt van een Leshan demoserver die al LwM2M-clients met data bevatte om het LoRa-netwerk en het IEEE 802.15.4

netwerk te simuleren. Ook hier werden er testen uitgevoerd op verschillende punten.

1. vertaallaag: door de data beschikbaar op een lokale Leshan server door te sluizen naar de vertaallaag en het resultaat uit te schrijven kon de functionaliteit volledig nagegaan worden. Soms hielp het om visualisatietools te gebruiken.

2. Solid pod: via de webinterface aangeboden bij onze Solid datapod was het mogelijk om bestanden in te laden nog voor één lijn code geschreven werd.

3. visualisatietool: een Python-script werd ontwikkeld om statische sensordata te genereren in RDF-formaat, zodat de visualisatiemogelijkheden tijdens ontwikkeling getest konden worden.

Van zodra alle onderdelen apart waren getest, werden ze samengevoegd voor een laatste test die over het hele netwerk zou gaan. Zo werd er gecontroleerd of de gegevens van de verbonden LwM2M-clients verschenen in de visualisatietool van het Solid systeem.

6 Toekomstige werkzaamheden

Het doel van dit project was om aan te tonen dat het mogelijk is om IoT & Solid te combineren, maar dit is zeker niet het einde. Er zijn nog verschillende verbeteringen en uitbreidingen mogelijk. Een van de aanpassingen die nog moet gebeuren is het aanpassen van de LoRa sensor zodat deze gebruik maakt van TheThingsNetwork op de correcte manier.

Voor de inlogprocedure (zie bijlage 8.4) zou het voordelig kunnen zijn om een aparte Type Registration voor LwM2M-IoT data te onderzoeken, zodanig dat dit niet botst met bijvoorbeeld de 'Notepod' applicatie (zoals nu wel het geval is).

Voor de vertaallaag zou het interessant zijn mocht deze van meerdere Leshan servers kunnen ontvangen en opslaan naar meerdere Solid pods. Meerdere Leshan servers zouden al ondersteund moeten zijn, maar dit moet nog getest worden. Voor meerdere Solid pods moet er gekeken worden naar de mogelijkheden om op meerdere tegelijk aan te melden met 'solid-auth-cli', wat uit een kleine test niet mogelijk leek.

De ontologie zou nog beter kunnen worden afgestemd, onder meer voor het ondersteunen van meerdere instanties van één resource en dat metingen zonder timestamp geen blanco knoop/skolem IRI hoeven te hebben. Dit laatste zorgt er voor dat er geen feedback nodig zou zijn (zie 8.5) en dat alles uniform is tussen meerdere Solid pods.

Het gebruik van RocketRML zou nog efficiënter kunnen worden geïmplementeerd door de aanpassingen en aanvullingen aan de data volledig te beperken tot de `preprocessJSON()`-functie om zo de belangen volledig te scheiden.

Een interessant idee voor de toekomst zou zijn om een soort IoT & Solid pakket op te stellen zodat ook minder gespecialiseerde personen zo'n netwerk eenvoudig kunnen opstellen. Dit zou voor een grotere markt en meer populariteit kunnen zorgen waardoor het doel van Solid een stap dichterbij komt.

7 Slotwoord

De afgelopen weken hebben we, werkende aan dit vakoverschrijdend project, enorm veel bijgeleerd over de domeinen van IoT en linked data door in de details van het Solid-project en de LwM2M-standaard te duiken.

Het doorgronden van deze nieuwe concepten en de implementatie van een oplossing hierrond heeft elk van ons ongetwijfeld frustraties gebracht, maar toen we tegen het einde van het project alle puzzelstukjes op hun plaats zagen vallen was de voldoening groot genoeg om deze te vergeten.

Over de samenwerking en communicatie in onze groep kunnen we alleen concluderen dat iedereen zijn steentje naar verwachting heeft bijgedragen en dat hieromtrent geen noemenswaardige problemen opgedoken zijn.

Het Solid-project ziet er meer dan ooit veelbelovend uit in de context van IoT en dit project vormt slechts een eerste stap in de koppeling van beide technologieën. Uit de vorige sectie blijkt echter dat best nog wat werk nodig is om deze verbinding op een meer schaalbare en commercieel toepasbare manier te verwezenlijken.

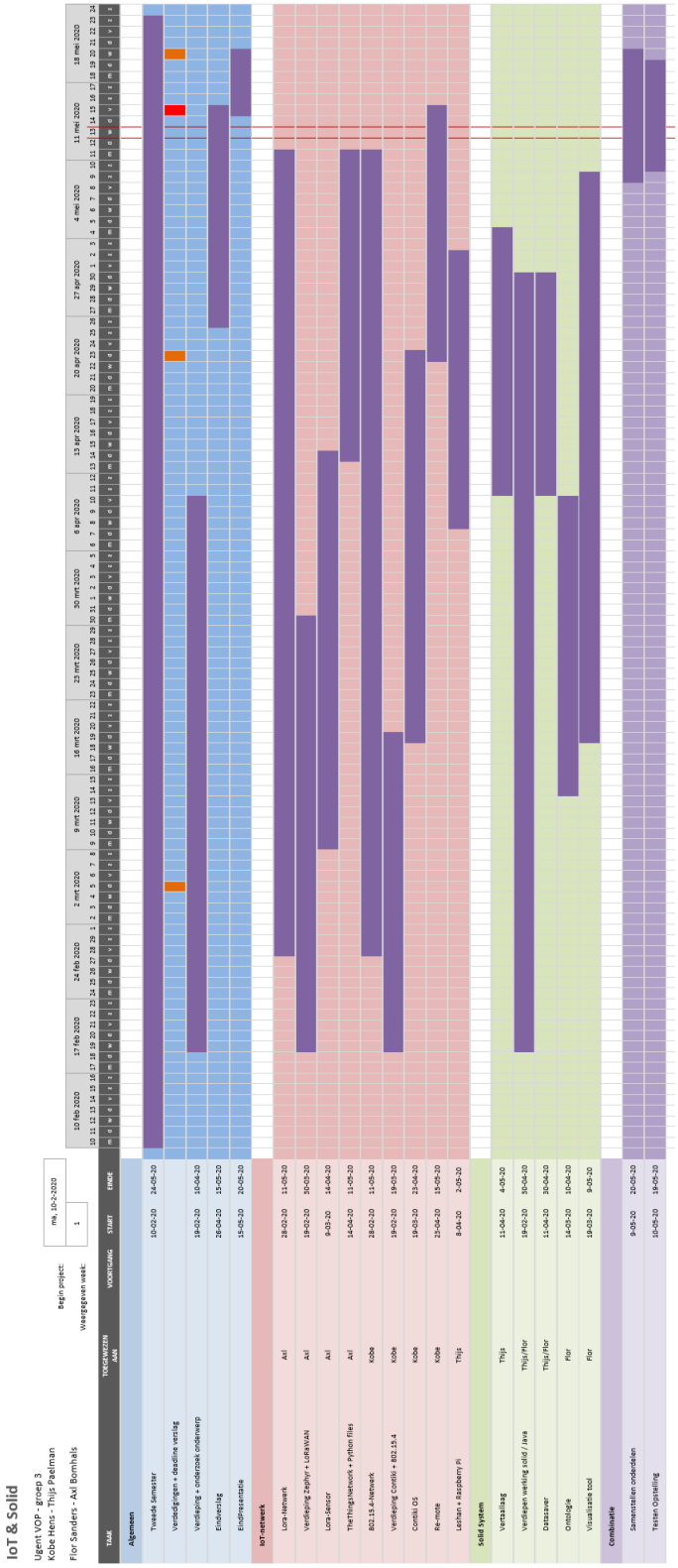
We zijn ervan overtuigd dat wanneer Solid opgroeit, gezien het project momenteel nog grotendeels in zijn kinderschoenen staat, dit de potentie heeft een competitief centraal platform te worden waarrond allerhande IoT-toepassingen zullen worden ontwikkeld.

Om deze redenen zijn wij alvast van mening dat dit een interessant en succesvol project was. Ook willen we graag onze begeleiders Jeroen Hoebeke, Abdulkadir Karaağaç en Bart Moons bedanken voor de hands-on feedback en hulp die ze ons verschaften tijdens het project.

Tot slot geven we als toekomstige referentie ook de Github-pagina mee waar de zaken ontwikkeld in de context van dit project terug te vinden zijn.
`https://github.ugent.be/tpaelman/IoT-Solid-project`

8 Bijlagen

8.1 Gantt schema



Figuur 17: Gantt schema.

8.2 SuperGlue pakket

In deze bijlage wordt extra informatie gegeven over de bestanden die gebruikt zijn om de lora-sensor te verbinden met de Leshan server via TheThingsNetwork (TTN).

Een eerste opmerking hierbij is dat een deel van deze bestanden een verbinding naar TTN simuleren, aangezien het niet mogelijk was om de werkelijke hardware te verbinden door een te grote afstand met de gateways van TTN. Het installeren van onze eigen lora-gateway was niet mogelijk door COVID-19.

Het SuperGlue pakket bevat 5 bestanden:

- *ClientDuctTape.py*
- *ServerDuctTape.py*
- *TTN.py*
- *Lora_Sensor bash script*
- *Lora_Server bash script*

8.2.1 Python files

8.2.1.1 ClientDuctTape.py Dit Python-bestand wordt uitgevoerd op hetzelfde apparaat dat de sensor simuleert. Het gebruikt een UDP verbinding om de CoAP berichten van de sensor op te vangen en als uplink door te sturen naar TTN via het *TTN.py* bestand. Tegelijkertijd zal het ook controleren of er een downlink is van TTN om deze dan via de UDP verbinding terug naar de sensor te sturen.

Bij een correcte werking van de sensor met de gateways van TTN is dit bestand overbodig. De berichten van de sensor zouden dan via LoRaWAN verstuurd worden en automatisch op TTN verschijnen.

8.2.1.2 ServerDuctTape.py Een bestand vergelijkbaar met *ClientDuctTape.py*. Dit bestand wordt samen met de Leshan server uitgevoerd op de Raspberry Pi. Dit bestand zal controleren of er een uplink naar TTN is en deze dan via een UDP verbinding door sturen naar de Leshan server. Tegelijkertijd zal het ook alle berichten ontvangen van de Leshan server als downlink door sturen naar TTN via het *TTN.py* bestand. Normaal gezien zou dit gebeuren via de MQTT Downlink functie.

8.2.1.3 TTN.py Dit bestand dient om JSON objecten te kunnen simuleren op TTN als uplink via een IPv4/IPv6 verbinding. Dit gebeurt door in te loggen op de online TTN console en hier de berichten gecodeerd in hexadecimaal te verzenden via de 'Simulate Uplink' functie.

Ook dit bestand zou bij een correcte werking van de sensor met de lora-gateway overbodig zijn.

8.2.2 Bash scripts

De bash scripts Lora_Sensor en Lora_Server zijn aangemaakt om de opstelling gebruiksvriendelijker op te starten. Het Lora_Sensor script zal uitgevoerd worden om de geëmuleerde sensor en het programma *ClientDuctTape.py* op te starten samen met extra processen die

een virtuele connectie aanmaken en deze na gebruik correct afsluiten. Het Lora_Server script zal uitgevoerd worden om de Leshan server en het *ServerDuctTape.py* bestand op te starten en deze na gebruik correct af te sluiten.

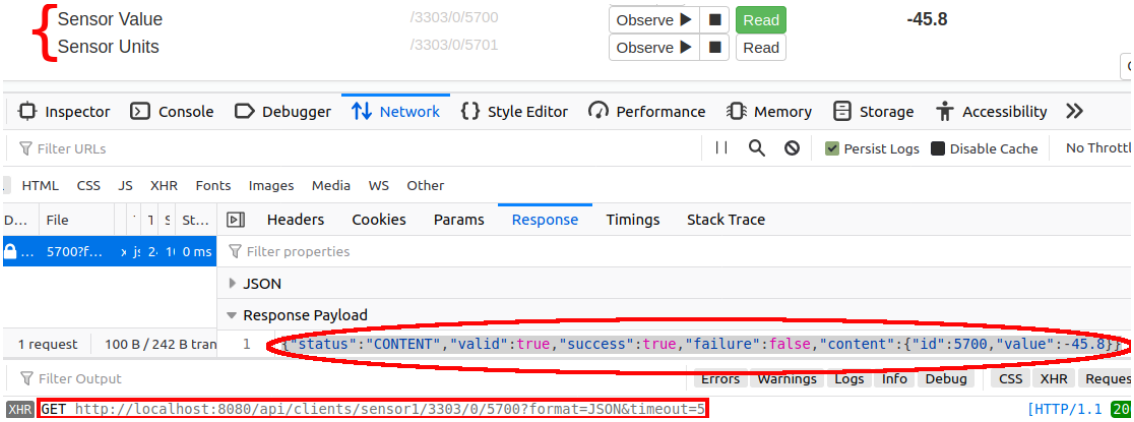
8.3 Leshan werking

Leshan API We doen hier even kort uit de doeken hoe de communicatie tussen de Leshan back-end (Java server) en front-end (Javascript webpagina) - van de demo server - werkt.

Leshan maakt gebruik van een zogeheten RESTful API, het gebruikt namelijk HTTP requests zoals GET, PUT, POST en DELETE om te interageren met de server.

Bij het raadplegen van de server in een webbrowser (bv. `http://localhost:8080/`), wordt er een webpagina teruggegeven die verrijkt is met Javascript. Deze gebruikt in de achtergrond de REST API van Leshan om alle gegevens op te vragen om de pagina op te vullen en om de functionaliteit te voorzien. Zo wordt bijvoorbeeld een GET request gestuurd naar `http://localhost:8080/api/clients` om een lijst in JSON formaat te krijgen van alle aanwezige sensoren.

Wordt de pagina van een apparaat bezocht, dan zal bv. een klik op de 'Read' button van de 'Sensor Value' een GET request versturen naar Leshan. Het antwoord is een JSON object dat de temperatuurwaarde bevat (zie figuur 18 met visualisatie van het netwerkverkeer in firefox developer tools).



Figuur 18: Uitlezen van temperatuur, netwerkverkeer weergegeven - hoofdzaken in rood.

Tot slot zal er met een druk op de 'Observe' knop één POST request verstuurd worden naar Leshan. De gevraagde meetwaarde zal in het antwoord verwerkt zitten. Maar daarna wordt deze waarde blijvend geüpdatet, zonder ogenschijnlijk bijkomstig netwerkverkeer. Dit wordt bewerkstelligd door gebruik te maken in Javascript van `new EventSource();`. Dit is een module die toestaat om in te schrijven op een zogenaamde event-stream. De front-end van onze demo server schrijft zich dus in bij de server om bij elke nieuwe gebeurtenis een melding te ontvangen en deze te verwerken d.m.v. een callback-functie, een functie die wordt opgeroepen telkens er een nieuw event binnenkomt. Dit ziet er zo uit in de code:

```
eventsource.addEventListener('EVENTTYPE', eventCallback);
```

Events Events zijn berichten die vanuit de Leshan server naar de ingeschreven applicaties worden verzonden op zo'n manier dat er niet elke keer vanuit de applicatie moet

worden gepeild of er al dan niet een nieuw bericht is (push i.p.v. pull). Dit zorgt voor een hogere efficiëntie van het gebruik van resources. Er zijn verschillende types events die worden onderscheiden door het `'EVENTTYPE'`. De twee belangrijkste voor het huidige project zijn 'NOTIFICATION' en 'COAPLOG' events.

Eerst ging alle aandacht naar de 'NOTIFICATION' events, als belangrijke spelers bij het updaten van de front-end. Het zijn deze events die er voor zorgen dat na een 'Observe' request de ingeschreven applicaties op de hoogte worden gehouden van veranderingen in de sensoren. Maar het grote probleem daarmee was dat op deze manier nooit de initiële meting verkregen werd, omdat deze via een andere weg wordt gestuurd, namelijk als antwoord op de POST request van de applicatie (bv. front-end). Ook 'Read' requests kunnen zo niet ontvangen worden, omdat die als HTTP antwoord verzonden worden op de GET requests. Het idee was even om de server zo aan te passen dat deze de HTTP antwoorden zou kopiëren naar de event-stream, maar dit was makkelijker gezegd dan gedaan.

Ondertussen werd het echter duidelijk dat er COAPLOG events worden verstuurd waarin *alle* CoAP berichten meegegeven worden. Het enige probleem was dat er in de COAPLOG data geen indicatie was met welke sensor er werd gecommuniceerd (in tegenstelling tot de NOTIFICATION's die dit wel hadden). Uiteindelijk werd voor dit project de demo server zo aangepast dat deze informatie wel meegegeven werd in de COAPLOG berichten [30].

8.4 Solid data acces

Praktische benadering van een gegevensopslagplaats op de Solid pod Zoals vermeld zijn er verschillende manieren om de opslaglocatie van de meetgegevens te benaderen. Eén manier is om gewoon de locatie van de gegevensopslag te hardcoderen en klaar is kees. Dit zorgt er dan wel weer voor dat er problemen kunnen ontstaan bij het hergebruik van deze data over verschillende applicaties heen en dat is allesbehalve wenselijk, want het Solid project gaat juist over het flexibel hergebruiken van data.

Langs de andere kant van het spectrum is er de meest flexibele manier, voorgesteld door prof. Ruben Verborgh in een blogpost [61]. Hier wordt er voorgesteld om gebruik te maken van zogeheten 'Shapes' om op een flexibele manier de juiste data te pakken te krijgen, ongeacht de structuur/opslagplaats van de data.

Aangezien dit op het moment van dit project nog grotendeels theorie is [62] en de locatie bezwaarlijk hardgecodeerd kon worden, werd er gekozen voor een praktisch implementeerbare middenweg voor het datatoegangssysteem in onze vertaallaag. Hiervoor werd inspiratie opgedaan bij een voorbeeldapplicatie 'Notepod' [63], dewelke in een tutorial voor het bouwen van een eerste Solid toepassing aan bod komt [64].

Probleemstelling Als een gebruiker zich inlogt op zijn Solid pod, dan is er eigenlijk maar één stuk informatie beschikbaar dat uiteindelijk moet leiden naar het ontdekken van de structuur van de opslag in Solid. Deze informatie is het *webId* van de persoon. Voor het testprofiel genaamd 'iotsolidugent' bijvoorbeeld is dit `https://iotsolidugent.inrupt.net/profile/card#me`). Dit webId wijst naar een document (alles voor de '#') op de pod die informatie bevat over de gebruiker (deze wordt dan voorgesteld door het webId op het internet - inclusief de '#me').

De volgende stappen zijn onder andere in de tutorial [65] beschreven, maar volledig herschreven in de vertaallaag wegens het gebruik van een andere bibliotheek.

```
@prefix : <#>.
@prefix solid: <http://www.w3.org/ns/solid/terms#>.
@prefix space: <http://www.w3.org/ns/pim/space#>.

:me space:storage ?x .
:me solid:privateTypeIndex ?y .
```

Listing 5: Gezochte triples in </profile/card>

```
[]  a solid:TypeRegistration;
    solid:forClass schem:TextDigitalDocument;
    solid:instance </private/leshandata.ttl> .
```

Listing 6: Type Registration voor onze meetgegevens in de PrivateTypeIndex.

Vanaf nu dient de pod van het testprofiel als de basis voor de rest van de uitleg (zoals in turtle zou gezegd worden: `@base <https://iotsolidugent.inrupt.net>.`), dus wordt de korte versie van het webId </profile/card#me>.

Stappenplan [66] Het document waarnaar het webId verwijst (</profile/card>) wordt opgehaald en daarin wordt gezocht naar 2 triples. Het eerste triple bevat informatie over de locatie van de opslag van deze pod, het tweede bevat informatie over de locatie van de private Type index. Dit is voorgesteld in listing 5. Deze (niet-bestaande) syntax is geïnspireerd door turtle en SPARQL en dient enkel om te duiden hoe de triples matchen, namelijk elk triple dat exact die waarden heeft die voorgesteld worden, waarbij een ? eender welk object mag zijn. Hier stellen ?x en ?y dus de gezochte waarden voor.

Het resultaat kan bv. ?x = </> en ?y = </settings/privateTypeIndex.ttl> zijn.

De Type Index is een document waarin wordt beschreven welke Type Registrations aanwezig zijn. We hebben een publiekelijk zichtbare en een private Type Index.

Type Registrations zijn statements die een bepaald type document koppelt aan een opslaglocatie in de pod. Zo hebben we bv. voor onze dataopslag de Type Registration in listing 6. Als voorbeeld toont listing 6 de Type Registration voor de dataopslag.

De private Type Index wordt ingelezen, waarna gezocht wordt naar exact die registration die voldoet aan listing 7. Op dit moment wordt de opslaglocatie nog geassocieerd met een schem:TextDigitalDocument, omdat dit hetzelfde was als in het geval van de Notepod, maar dit wordt in de toekomst best veranderd naar iets typerender voor onze applicatie.

Als laatste wordt het Object van solid:instance gezocht dat hoort bij het gevonden Subject ?registration en op deze manier is de opslaglocatie gekend (</private/leshandata.ttl>)!

Echter, bij de initiële connectie van de toepassing met een nieuwe Solid pod, moet deze Type Registration nog aangemaakt worden Dat is het moment waarop de gebruiker

```
?registration  a solid:TypeRegistration;
    solid:forClass schem:TextDigitalDocument .
```

Listing 7: Type Registration SPARQL query.

in principe zelf kan kiezen waar hij zijn data gaat opslaan [67], maar momenteel werd enkel de defaultlocatie geïmplementeerd (dezelfde als hier als voorbeeld is gebruikt).

Verder wordt rekening gehouden met mogelijke issues (als de defaultlocatie reeds bestaat, wordt ze niet overschreven enzoverder).

8.5 Uitdagingen RocketRML

Object en resource mapping Zoals in listing 3 te zien is, kan de payload van een CoAP bericht verschillende metingen bevatten, waarvan de LwM2M-objecten en resources numeriek worden voorgesteld.

Maar om deze objecten en resources af te kunnen beelden op hun respectievelijke klasse-namen (bv. object 3303 ↔ `lwm2m:LWM2MTemperatureObject`), moet de LwM2M-ontologie bevraagd worden.

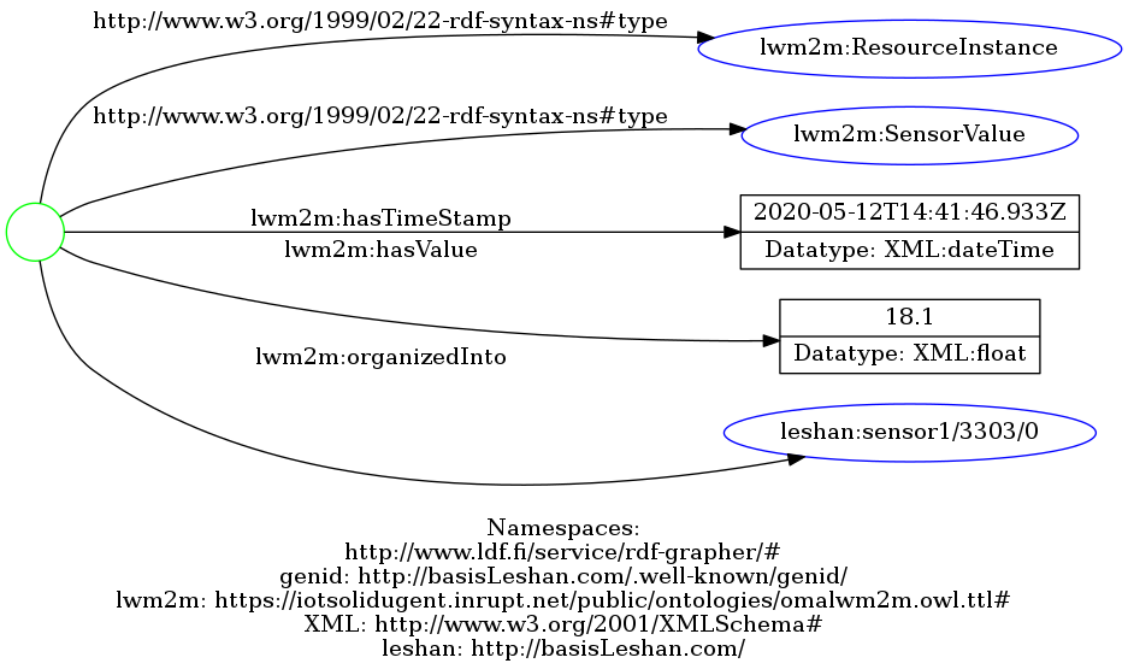
Een hargecodeerde mapping is goed als proof of concept, maar uiteraard niet duurzaam. Om de ontologie te bevragen, werd een andere bibliotheek gebruikt die SPARQL queries toelaat. SPARQL is een taal om RDF grafen te bevragen. De ontologie is dus opgeslagen als een RDF-graaf, waarna deze bevraagd wordt met een SPARQL query die specifiek op zoek gaat naar het element van de ontologie die zegt dat een `lwm2m:LWM2MTemperatureObject` een `lwm2m:hasObjectID` heeft van 3303.

Deze methode is echter afhankelijk van de opslagvorm van de ontologie, waar niet in elk geval vanuit kan worden gegaan. Voor toekomstige referentie stelden we ook reeds de vraag op Stackoverflow [68].

Blanco knopen In figuur 19 is de structuur zichtbaar die gebruikt wordt om de verschillende metingen in RDF op te slaan. Hier wordt van het concept 'blanco knopen' gebruik gemaakt. Blanco knopen zijn anonieme knooppunten in de RDF graaf die enkel worden gebruikt om bepaalde data met elkaar te koppelen, maar zelf niet aanspreekbaar hoeven te zijn. Door een probleem met de gebruikte bibliotheken echter [69], kon niet voortgebouwd worden op dit idee. Daarom zijn de knooppunten nu toch expliciet benoemd, maar op zo'n manier dat de specificaties van RDF zeggen dat het voor alle programma's duidelijk zou moeten zijn dat dit eigenlijk blanco knopen zijn. Dit is het principe van 'skolemization' [70].

Timestamp Een andere uitdaging was dat niet iedere resource in de voorgestelde ontologie een timestamp nodig heeft, maar dat we dit initieel wel voor elke meting deden (ongeacht wat de ontologie zei). Zo konden er bv. meerdere Min Measured Values zijn op verschillende tijdstippen, net zoals bv. verschillende metingen van de Sensor Units, ook al zijn die volgens de ontologie niet tijdsgebonden. Dit had als neveneffect dat nu niet enkel de timestamps genegeerd moesten worden, maar ook dat de vorige meting overschreven moest worden als die al aanwezig was in de Solid pod.

Om dit op te lossen, was er een soort terugkoppeling nodig, zodanig dat eerder aangemaakte metingen zonder timestamp worden overschreven. Het overschrijven wordt dan bewerkstelligd door de Skolem IRI (die normaal volledig random is en een verwaarloosbare kans op reproductie heeft) expliciet te gaan uitlezen van de al aanwezige meting en die mee te geven aan de nieuwe meting. Dit zorgt er wel voor dat de gegenereerde RDF graaf verschilt van pod tot pod. Jammer genoeg werd er zo ook een (niet-dringende) bug geïntroduceerd in het verwachte gedrag [71].



Figuur 19: RDF visualisatie van een resource meting met blanco knoop (linkse cirkel).

Referenties

[1] 80 insightful internet of things statistics. [Online]. Available: https://safeatlast.co/blog/iot-statistics/

[2] Anonymous. Internet of things. [Online]. Available: https://en.wikipedia.org/wiki/Internet_of_things

[3] T. Berners-Lee. One small step for the web.... [Online]. Available: https://inrupt.com/blog/one-small-step-for-the-web

[4] Solid mit website. [Online]. Available: https://solid.mit.edu/

[5] Cloud based iot solar monitoring system. [Online]. Available: https://www.kalki.io/saas-applications/solar-pv-monitoring/

[6] R. Molla. (2019) Your smart devices listening to you, explained. [Online]. Available: https://www.vox.com/recode/2019/9/20/20875755/smart-devices-listening-human-reviewers-portal-alexa-siri-assistant

[7] Facebook-cambridge analytica data scandal. [Online]. Available: https://en.wikipedia.org/wiki/Facebook%E2%80%93Cambridge_Analytica_data_scandal

[8] OMA. Lwm2m v1.1 overview presentation. [Online]. Available: http://www.openmobilealliance.org/release/LightweightM2M/Lightweight_Machine_to_Machine-v1_1-OMASpecworks.pdf

[9] OMA. Lightweight machine to machine technical specification:. [Online]. Available: http://www.openmobilealliance.org/release/LightweightM2M/V1_1_1-20190617-A/OMA-TS-LightweightM2M_Core-V1_1_1-20190617-A.pdf

[10] Practical introduction to coap. [Online]. Available: https://opensourceforu.com/2016/09/coap-get-started-with-iot-protocols/

[11] B-l072z-lrwan1. [Online]. Available: https://www.st.com/en/evaluation-tools/b-l072z-lrwan1.html

[12] Zephyr project. [Online]. Available: https://www.zephyrproject.org/

[13] Lorawan architecture. [Online]. Available: https://www.thethingsnetwork.org/docs/lorawan/architecture.html

[14] Qemu the fast! processor emulator. [Online]. Available: https://www.qemu.org/

[15] T. H. Team. Publish & subscribe - mqtt essentials: Part 2. [Online]. Available: https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/

[16] Mqtt. [Online]. Available: http://mqtt.org/

[17] Lpwan static context header compression (schc) and fragmentation for ipv6 and udp. [Online]. Available: https://tools.ietf.org/id/draft-ietf-lpwan-ipv6-static-context-hc-17.html#rfc.section.5

[18] J. T. Adams, "An introduction to ieee std 802.15.4." [Online]. Available: https://web.sonoma.edu/users/f/farahman/sonoma/courses/cet543/resources/802_intro_01655947.pdf

[19] A. Prisantama, W. Widyawan, and I. Mustika, "Tunneling 6lowpan protocol stack in ipv6 network," vol. 1755, 07 2016, p. 070006.

[20] 2020. [Online]. Available: https://en.wikipedia.org/wiki/IEEE_802.15.4#/media/File:IEEE_802.15.4_protocol_stack.svg

[21] e. notes, "Ieee 802.15.4 standard: Primer electronics notes," 2020. [Online]. Available: https://www.electronics-notes.com/articles/connectivity/ieee-802-15-4-wireless/basics-tutorial-primer.php

[22] 2020. [Online]. Available: https://en.wikipedia.org/wiki/6LoWPAN

[23] e. notes, "What is 6lowpan for iot & m2m - electronics notes," 2020. [Online]. Available: https://www.electronics-notes.com/articles/connectivity/ieee-802-15-4-wireless/6lowpan.php

[24] "What is 6lowpan," 2020. [Online]. Available: https://zolertia.io/6lowpan-iot-protocol/

[25] 2020. [Online]. Available: https://github.com/Zolertia/Resources/wiki/RE-Mote

[26] 2020. [Online]. Available: https://en.wikipedia.org/wiki/Contiki

[27] 2020. [Online]. Available: https://github.com/contiki-os/contiki

[28] 2020. [Online]. Available: https://github.com/Zolertia/Resources/wiki/RE-Mote

[29] Eclipse leshan. [Online]. Available: https://www.eclipse.org/leshan/

[30] Leshan server - solid implementatie. [Online]. Available: https://github.ugent.be/tpaelman/Leshan-server-Solid

[31] W3C. Linked data. [Online]. Available: https://www.w3.org/standards/semanticweb/data

[32] B. M. Graham Klyne, Jeremy J. Carroll. Rdf 1.1 concepts and abstract syntax. [Online]. Available: https://www.w3.org/TR/rdf11-concepts/

[33] L. M. Dan Brickley. Foaf vocabulary specification 0.99. [Online]. Available: http://xmlns.com/foaf/spec/

[34] Anonymous. Resource description framework. [Online]. Available: https://en.wikipedia.org/wiki/Resource_Description_Framework#Serialization_formats

[35] OMA. Oma lightweightm2m (lwm2m) object and resource registry. [Online]. Available: http://www.openmobilealliance.org/wp/omna/lwm2m/lwm2mregistry.html

[36] L. Daniele. Oma lightweight m2m ontology. [Online]. Available: https://sites.google.com/site/smartappliancesproject/ontologies/oma-lightweight_m2m-ontology

[37] Querying history with linked data. [Online]. Available: https://ruben.verborgh.org/blog/2016/06/22/querying-history-with-linked-data/

[38] Browserless persistent solid login. [Online]. Available: https://forum.solidproject.org/t/solid-auth-cli-browserless-persistent-login/1016

[39] Solid authentication client. [Online]. Available: https://www.npmjs.com/package/solid-auth-client

[40] Solid authentication without frontend - question. [Online]. Available: https://forum.solidproject.org/t/solid-authorization-and-data-handling-without-frontend/2874

[41] Solid authentication error. [Online]. Available: https://github.com/solid/solid-cli/issues/15

[42] Node.js. [Online]. Available: https://nodejs.org/en/about/

[43] Sensor measurement lists (senml). [Online]. Available: https://tools.ietf.org/html/rfc8428

[44] Oma technical specification. [Online]. Available: https://www.openmobilealliance.org/release/LightweightM2M/V1_0_2-20180209-A/OMA-TS-LightweightM2M-V1_0_2-20180209-A.pdf

[45] How to map json to rdf. [Online]. Available: https://etl.linkedpipes.com/tutorials/how-to/map_json_to_rdf

[46] Semantic data integration on the web of things. [Online]. Available: https://www.vcharpenay.link/publications/2018-iot.pdf

[47] Designing cross-domain semantic web of things applications. [Online]. Available: https://sensormeasurement.appspot.com/?p=end_to_end_scenario

[48] Github projectpagina rmlmapper. [Online]. Available: https://github.com/RMLio/rmlmapper-java

[49] Rdf mapping language website portal. [Online]. Available: https://rml.io/

[50] Rml specifications. [Online]. Available: https://rml.io/specs/rml/

[51] Rocketrml javascript bibliotheek (beschikbaar via npm). [Online]. Available: https://developer.aliyun.com/mirror/npm/package/rocketrml

[52] Rocketrml javascript bibliotheek - github pagina. [Online]. Available: https://github.com/semantifyit/RocketRML

[53] Rml functionmap. [Online]. Available: https://fno.io/rml/

[54] Rocketrml issue - constant shortcut properties. [Online]. Available: https://github.com/semantifyit/RocketRML/issues/9

[55] Rocketrml issue - supporting javascript async funtions. [Online]. Available: https://github.com/semantifyit/RocketRML/issues/11

[56] Rocketrml issue - iterator syntax. [Online]. Available: https://github.com/semantifyit/RocketRML/issues/14

[57] rdflib.js - javascript rdf library. [Online]. Available: https://github.com/linkeddata/rdflib.js/

[58] Inrupt. Solid react sdk. [Online]. Available: https://inrupt.com/sdk

[59] Facebook. React. [Online]. Available: https://reactjs.org/

[60] Apache echarts. [Online]. Available: https://echarts.apache.org

[61] Using shapes for linked data access. [Online]. Available: https://ruben.verborgh.org/blog/2019/06/17/shaping-linked-data-apps/#top-p-3

[62] Blog post: Shaping linked data apps. [Online]. Available: https://forum.solidproject.org/t/blog-post-shaping-linked-data-apps/1914/7

[63] Notepod application. [Online]. Available: https://gitlab.com/vincenttunru/notepod/

[64] Tutorial: writing a solid app. [Online]. Available: https://solidproject.org/for-developers/apps/first-app

[65] Writing a solid app tutorial - step 4. [Online]. Available: https://vincenttunru.gitlab.io/tripledoc/docs/writing-a-solid-app/4-data-model.html

[66] Solid data discovery. [Online]. Available: https://github.com/solid/solid/blob/master/proposals/data-discovery.md

[67] The right location for a new app's data? [Online]. Available: https://forum.solidproject.org/t/the-right-location-for-a-new-apps-data/662/2

[68] Querying an owl ontology. [Online]. Available: https://stackoverflow.com/questions/61801054/querying-an-owl-ontology

[69] blank node merging issue. [Online]. Available: https://github.com/linkeddata/rdflib.js/issues/405

[70] Skolemizations rdf specification. [Online]. Available: https://www.w3.org/TR/rdf11-concepts/#section-skolemization

[71] Github issue 1 - solid feedback werkt enkel na heropstarten. [Online]. Available: https://github.ugent.be/fpsander/soliddatasaver/issues/1