Tuomas Palosaari

Langattomien sensoriverkkojen yhteentoimivuus yhdyskäytävätasolla

Tietotekniikan
pro gradu -tutkielma
8. joulukuuta 2019

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Kokkolan yliopistokeskus Chydenius

Tekijä: Tuomas Palosaari
Yhteystiedot: tuomas.palosaari@iki.fi
Puhelinnumero: 050-5827835
Ohjaaja: Ismo Hakala
Työn nimi: Langattomien sensoriverkkojen yhteentoimivuus yhdyskäytävätasolla
Title in English: Sensor data
Työ: Tietotekniikan pro gradu -tutkielma
Sivumäärä: 71+4
Tiivistelmä: Tutkielman tarkoituksena oli perehtyä esineiden internetin ja erityisesti langattomien sensoriverkkojen perusteknologioihin ja eri laitteiden yhteentoimivuuteen liittyviin ongelmiin. Esineiden internetin laitekirjo on valtava eikä vielä ole mitään varsinaista standardia eri järjestelmien keskinäiseen kommunikaatioon. Olisi kuitenkin suotavaa, että järjestelmät käyttäisivät internetissä jo käytettäviä teknologioita, kuten HTTP, HTTPS tai CoAP. Yhteentoimivuuden suhteen ongelmaksi muodostuu käytettävien viestien muoto ja tiedon esitys. Jokaisella laitevalmistajalla on oma tapansa enkoodata ja esittää mittaustieto. Tähän tarjotaan ratkaisuksi niin sanottuja teknologian sovituskerroksia. Helpoiten tämän roolin hoitaa eri järjestelmien välille rakennettu yhdyskäytäväpalvelu.

Tutkielman teoriaosuudessa vertailtiin WWW-sovelluspalveluteknologioita langattomien sensoriverkkojen näkökulmasta ja esiteltiin lyhyesti langattomien sensoriverkkojen käyttämiä tiedonsiirtoprotokollia, tiedon esitys- ja varastointitapoja sekä eri standardointiorganisaatioiden ja projektien tuottamia rajapintamääritelmiä ja tiedon esitystapoja. Tämän lisäksi perehdyttiin tarkemmin TTY:n kehittämään WSN OpenAPI -rajapintamääritelmään.

Tutkielman tuloksena syntyi Kokkolan yliopistokeskuksen informaatioteknologian yksikön käyttöön WSN OpenAPI -määritelmää mukaileva REST-rajapinnan tarjoava palvelu sekä kevyt web-asiakasohjelma, joka hyödyntää tätä rajapintaa.

Tällaisilla yhdyskäytävillä on mahdollista yhdistää useita eri standardeja käyttäviä sensoriverkkoja toisiinsa, mutta tämä edellyttää jokaiselle erilaiselle verkolle omaa yhdyskäytävää. Tämä voi olla haaste usean eri laitevalmistajan laitteita käytettäessä.
Avainsanat: WWW-sovelluspalvelu, REST, esineiden internet, sensoriverkot, yhdyskäytävät, API
Abstract: The purpose of this thesis was to get acquainted with the basic technologies of the Internet of Things and more specifically the wireless sensor networks and problems related to interoperability of different devices. The IoT device spectrum is enormous and at the time of writing there is no one standard for communication

between different systems. It would be desirable for systems to use technologies that are already in use on the Internet such as HTTP/HTTPS and CoAP. The form and representation of the messages and information are the problems with interoperability. Each device manufacturer has their own way of encoding and representing the measurement data. So-called technology adaptation layers are offered as a solution. These layers are most easily handled by building gateway services between different systems.

The theory part of the thesis consists of comparing web service technologies from the point of view of wireless sensor networks, the different ways of presenting and storing the data and the interface specifications produced by different standardization organizations and projects. In addition, the WSN OpenAPI interface developed by TUT was further explored.

A service providing a REST API conforming to the WSN OpenAPI specification and a lightweight web client utilizing the API was produced as a result of this work for the Information Technology unit of the Kokkola University Consortium Chydenius.

With gateways such as the one created it is possible to connect sensor networks using different standards for communication, but this requires the implementation of a new gateway service for each different network. This can prove to be a challenge when using devices from several different manufacturers.
Keywords: Web Service, REST, Internet of things, sensor networks, gateways, API

Sanasto

CoAP	Constrained Application Protocol
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
IDL	Interface Description Language
IoT	Internet of Things
IP	Internet Protocol
JSON	Javascript Object Notation
JWT	JSON Web Token
LPWAN	Low Power Wide Area Network
M2M	Machine to machine
MQTT	Message Queue Telemetry Transport
MQTT-S	MQTT for Sensors
REST	Representational State Transfer
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
XML	Extensible Markup Language
WADL	Web Application Description Language
WPAN	Wireless Personal Area Network
WSDL	Web Services Description Language

Sisältö

1 Johdanto

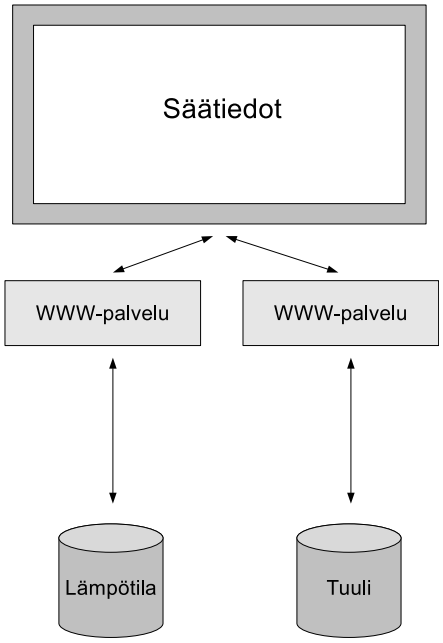
Esineiden internet (Internet of things) laajenee jatkuvasti ja sen tuottama tieto on monipuolista sekä esitystavat monenkirjavia. Verkkoon yhdistettyjen laitteiden ja niiden tuottaman tiedon määrä kasvaa myös koko ajan. Esineiden internetiin liittyvät myös langattomat sensoriverkot. Langattomien sensoriverkkojen käyttötarkoitukset vaihtelevat teollisuudesta ja maataloudesta älykoteihin. Näillä laitteilla on käytettävistä teknologioista ja laitteiden luonteesta johtuvat ongelmat esineiden internetin haasteiden lisäksi.

Eräs mahdollinen käyttötarkoitus langattomalle sensoriverkolle voisi olla esimerkiksi erilaisten mittauslaitteiden tulosten tallennus ja haku. Nykyaikana esimerkiksi tietoa kaupungin ilmanlaadusta ja säätiedoista voi olla monessa eri järjestelmässä, jolloin tiedon kerääminen yhteen paikkaan on työlästä. Kuvassa 1.1 näkyy miten useassa eri tietokannassa olevat säähän liittyvät tiedot voidaan koota yhteen sovellukseen tai näkymään. Tarvittava tieto voi sijaita useassa eri järjestelmässä ja asiakkaalle esitetään niistä kooste. Eri säätietojärjestelmissä on pääasiassa samankaltaista tietoa, mutta eri muodossa tallennettuna. On tärkeää saada nämä eri järjestelmissä olevat tiedot yhdistettyä ja otettua käyttöön.

Eri järjestelmien yhteentoimivuus onkin muodostunut ongelmaksi kun jokaisen laitevalmistajan järjestelmä voi puhua, jos ei eri kieltä, niin vähintäänkin eri murretta. Laitevalmistajien tapa esittää tieto ja tarjota se toisten käyttöön vaihtelee paljon. Tällöin eri laitekantojen ja verkkojen yhteentoimivuudesta tulee haastavaa. Yksi ongelma onkin niin sanottu siiloutuminen. Siiloutumisella tarkoitetaan tässä yhteydessä järjestelmän vertikaalista eristäytymistä toisista järjestelmistä. Eri järjestelmien yhteentoimivuuden mahdollistavat esimerkiksi WWW-sovelluspalveluteknologiat.

Tutkielman tavoitteena on selvittää minkälaisten rajapintojen kautta langattoman sensoriverkon tuottama mittaustieto voidaan tarjota muiden järjestelmien käyttöön ja miten se voidaan muuttaa hierarkisesti esitettyyn ja standardoituun muotoon. Tämän lisäksi on myös selvitettävä käytettävissä olevien WWW-sovelluspalveluteknologioiden soveltuvuus tähän tarkoitukseen. Tavoitteena on myös selvittää mitä tiedonsiirtoprotokollia on käytettävissä ja mitkä niistä soveltuvat parhaiten langattomien sensoriverkkojen käyttöön tiedon käytön näkökulmasta.

Tutkielman luvussa kaksi käydään läpi langattomat sensoriverkot ja niiden toi-



Kuva 1.1: WWW-sovelluspalvelut säätietojärjestelmässä

minta. Luvussa perehdytään myös langattomuuden aiheuttamiin haasteisiin. Luvussa kolme käydään läpi WWW-sovelluspalveluiden yleisiä periaatteita ja teknologiat. Luku neljä käsittelee langatonta sensoriverkkoa sovelluspalveluiden näkökulmasta. Luvussa käydään läpi käytettyjä protokollia sekä haasteita. Luvussa viisi käydään läpi WSN OpenAPI -määritelmä ja sen tarjoamat eri rajapinnat. Luvussa kuusi esitellään lyhyesti käytössä oleva toimintaympäristö, määritellään tiedon esitys ja esitellään määritelmää mukailevan WWW-sovelluspalvelun toteutus sekä asiakasohjelma, joka käyttää tätä WWW-sovelluspalvelua. Luvussa seitsemän on yhteenveto ja pohdinta.

2 Langattomien sensoriverkkojen ominaisuuksista

Tässä luvussa käydään läpi langattoman sensoriverkon määritelmä ja lyhyesti tällä hetkellä käytössä olevia verkkoprotokollia. Tämän lisäksi perehdytään sensoriverkkojen ominaisuuksiin, rajoituksiin ja niihin liittyviin haasteisiin.

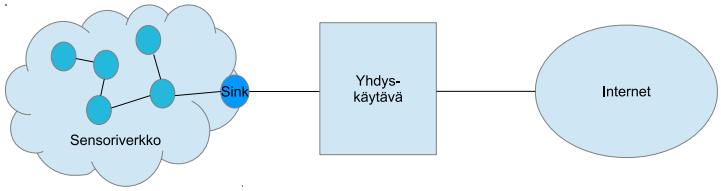
2.1 Määritelmä

Kirjassaan [57] Vasseur ja Dunkels määrittelevät älykkään laitteen (Smart Object) seuraavasti: esine, joka on varustettu jonkinlaisella anturilla tai toimilaitteella, prosessorilla, kommunikointilaitteella ja virtalähteellä. Anturi ja toimilaite mahdollistavat kanssakäymisen fyysisen maailman kanssa. Prosessorin avulla laite käsittelee anturien antamaa tietoa. Kommunikointilaite mahdollistaa tiedon lähettämisen ulkomaailmaan tai toisille laitteille. Sen avulla myös voidaan vastaanottaa viestejä toisiltä laitteilta.

Rawat et al määrittelevät artikkelissaan [45] langattoman sensoriverkon pienten anturinoodiksi kutsuttujen laitteiden verkoksi, joka on levitetty johonkin tilaan ja jonka osat välittävät yhteistyössä alueelta keräämänsä informaatiota langattomien linkkien kautta. Kerätty tieto lähetetään sink-noodille, joka joko välittää sen eteenpäin yhdyskäytävän kautta tai käsittelee sitä paikallisesti. Langattoman sensoriverkon rakenne näkyy kuvassa 2.1. Verkon anturinoodi on laite, jossa on prosessori, radiolähetin, muistia, virtalähde ja yksi tai useampi mittalaite. Mittalaite voi mitata esimerkiksi lämpötilaa, ilman kosteutta, valoisuutta tai äänenvoimakkuutta. Laitteiden pieni koko rajoittaa niiden suorituskykyä monelta osin; muistia ei ole määrättömästi, prosessori ei ole tehokas, radion kantama ei ole pitkä eikä akunkesto ole pitkä. Tämä rajoittaa myös mittausten tiheyttä ja lähetettävää tietoa.

Langattoman sensoriverkon noodi on siis älylaitteiden erikoistapaus. Yleisen älylaitteen määritelmän mukaan kommunikointitapaa ei ole rajoitettu, vaan se voi kulkea joko kaapelia pitkin tai langattomasti.

Langaton sensoriverkko voidaan sijoittaa joko satunnaisesti tai suunnitellusti. Edellinen tapa toimii paremmin jos verkon on tarkoitus kattaa iso alue ja jättää se suorittamaan mittauksia ilman valvontaa. Jälkimmäisen etuna on hallinnan helppous kun sijoitettavia noodeja on vähemmän ja ne laitetaan tiettyihin ennalta määrättyihin paikkoihin. [45]



Kuva 2.1: Langaton sensoriverkko

2.2 Rajoitteet ja haasteet

Sensoriverkkoihin ja niiden käyttöön liittyy erilaisia rajoitteita ja haasteita johtuen niiden luonteesta ja fyysisistä ominaisuuksista. Akyildiz et al ovat keränneet artikkelissaan[2] langattoman sensoriverkon suunnitteluun vaikuttajia tekijöitä.

- Viansieto
- Skaalautuvuus
- Tuotantokulut
- Toimintaympäristö
- Verkon rakenne
- Raudan rajoitteet
- Viestin välitys
- Virrankulutus

Viansiedolla tarkoitetaan verkon kykyä jatkaa toimintaa yhden tai useamman noodin pettämisestä huolimatta. Verkon käyttämiä algoritmeja ja protokollia suunnitellessa ja valitessa tulee ottaa käyttötarkoitus ja toimintaympäristö huomioon. Esimerkiksi rakennuksen lämpötilamittauksissa viansiedon ei tarvitse olla korkea, koska tällaisessa käytössä noodit vaurioituvat harvoin.

Verkon noodien määrän skaalautuvuus on myös otettava huomioon. Noodien määrä voi kasvaa suureksi mittauskohteen luonteesta riippuen. Myös noodien tiheys pitää huomioida.

Langattoman sensoriverkon yksittäisen noodin hinnan tulee olla tarpeeksi alhainen, jotta teknologian käytössä olisi järkeä verrattuna perinteisiin mittalaitteisiin. Tämä korostuu kun sensoriverkon noodien määrä kasvaa.

Sensoriverkon toimintaympäristöt asettavat myös rajoituksensa noodeille. Ne voivat joutua kestämään äärimmäisiä olosuhteita tai niiden tarvitsema verkkoliikenne voi tulla häirityksi.

Verkon topologia ja sen ylläpito ovat oma haasteensa. Riippuen verkon koosta ja tiheydestä, verkon sijoituksessa ja ylläpidossa tulee ottaa huomioon asennuksen kustannukset, joustavuus, noodien tavoitettavuus ja mahdollinen noodien hajoaminen.

Tärkeimpiä noodien rakennetta rajoittavia tekijöitä ovat koko ja virrankäyttö. Käyttötarkoituksesta riippuen noodi voi olla tulitikkuaskin kokoinen tai pienempi. Pieneen laitteeseen ei mahdu isoa akkua mukaan ja tämä rajoittaa komponenttien valintaa. Tästä syystä myös laitteen muisti on rajallinen.

Yleisin viestinvälitystapa on radio. Sopivan taajuuden valinta, mahdollinen häiriö muista lähetyksistä ja lähetysteho vaikuttavat radiolähettimen valintaan. Muita kommunikointitapoja voivat olla esimerkiksi infrapuna tai laser-diodi, mutta nämä vaativat näköetäisyyden toisiin solmuihin.

Virrankulutus on merkittävä tekijä langattoman sensoriverkon suunnittelussa ja se voidaan jakaa kolmeen osaan: anturien, kommunikoinnin ja tiedonkäsittelyn virrankulutus. Anturien virrankäyttö riippuu paljon mittaustiheydestä, mitattavan ilmiön luonteesta ja kohinan määrästä. Suurin osa virrankulutuksesta syntyy kommunikoinnista. Viestien lähettämiseen ja vastaanottamiseen kuluu suunnilleen yhtä paljon virtaa.

Myös Rawat et al esittävät edellä mainittujen lisäksi artikkelissaan[45] oman näkemyksensä langattomien sensoriverkkojen kohtaamista haasteista sekä miten niihin voitaisiin vastata.

- Resurssirajoitukset
- Ympäristötekijät
- Tiedon päällekkäisyys
- Viestinnän epäluotettavuus
- Globaalin identifioinnin puute
- Noodin hajoaminen
- Sijoituksen skaala

Resurssirajoituksilla tarkoitetaan käytettävissä olevaa virtaa, muistin määrää sekä laskennallista kapasiteettia. Käytettävissä olevan virran tehokkaaseen hyödyntämiseen on pyritty esimerkiksi virtaa säästävällä reitityksellä tai MAC-kerroksen virransäästötilalla. Sensoriverkon sijoitusympäristö voi johtaa noodien hajoamiseen tai viestinnän heikkenemiseen. Tämän takia verkon anturien pitää pystyä esimerkiksi käyttämään eri viestintäprotokollia tai signaalinkäsittelyalgoritmeja. Myös yksittäisen noodin hajoamiseen pitää voida varautua reitittämällä viestit uudestaan toisen noodin kautta. Tätä voidaan ennakoida asentamalla suuri määrä noodeja, mutta se tuo mukanaan muita ongelmia. Noodien läheisyys moninkertaistaa välitetyn mittaustiedon määrän, joten pitää osata suodattaa pois samaa ilmiötä useaan kertaan ilmaiseva tieto. Tämän lisäksi pitää pystyä identifioimaan jokainen erillinen noodi ja verkon tulee pystyä tukemaan suuria määriä noodeja. Kun puhutaan tuhansien noodien verkoista, tämä vaatii noodeilta kykyä organisoida tiedonvälitys ja reititys itse.

Yllä esitetyistä haasteista ja rajoituksista keskeisimmiksi nousevat virrankäyttö, noodin kustannukset ja toimintaympäristöstä riippuva toiminnan luotettavuus.

2.3 Verkkoprotokollat

Yksi tapa vastata näihin haasteisiin on valita sopiva verkkoprotokolla. IoT-käyttöä varten on tällä hetkellä tarjolla useita eri protokollia. Pääasiassa jako voidaan tehdä kantaman perusteella. Lyhyen kantaman protokollista käytetään nimitystä Wireless Personal Area Network (WPAN) ja pitemmän kantaman protokollista käytetään Centenaron et al[14] mukaan yleensä nimitystä Low Power Wide Area Network (LPWAN).

ZigBee[3] ja 6LoWPAN[39] ovat pääasiassa lyhyen kantaman tiedonsiirtoprotokollia. Kummatkin käyttävät IEEE 802.15.4 -standardia ja tämän lisäksi 6LowPAN tukee myös Wi-Fiä. Kummatkin ovat kantamaltaan muutamia kymmeniä metrejä. Protokollat voivat käyttää kolmea eri taajuutta tiedonsiirtoon, 868 MHz, 915 MHz ja 2,4 GHz. Tiedonsiirtonopeus on vastaavasti 20 kb/s, 40 kb/s tai 250 kb/s ja viestin koko korkeintaan 127 tavua[20].

Uudempia pitemmän kantaman verkkoprotokollia ovat muun muassa Sigfox, LoRaWan ja NB-IoT. Mekki et al ovat vertailleet artikkelissaan[38] LPWAN -teknologioista edellä mainittuja. LPWAN -teknologiat mahdollistavat kilometrien kantaman. Jokaisella näistä kolmesta on kuitenkin rajoituksensa ja etunsa. Sigfox tarjoaa pisimmän kantaman eli 10 kilometriä kaupunkialueilla, 40 kilometriä kaupunkialueiden ulkopuolella mutta päivittäisten viestien määrä ja hyötykuorma on pie-

ni, vain kaksitoista tavua lähetettäessä tai kahdeksan tavua vastaanottaessa. LoRaWAN ei rajoita viestien määrää ja tarjoaa 243 tavun hyötykuorman. Kantamaksi LoRaWAN lupaa 5 kilometriä kaupungissa ja 20 kilometriä kaupunkien ulkopuolella. NB-IoT on kantamaltaan pienin, 1 kilometri kaupungissa ja 10 kilometriä kaupunkien ulkopuolella. NB-IoT:n hyötykuorma on kuitenkin LoraWAN:ia suurempi, 1600 tavua. NB-IoT tarjoaa myös paremman palvelulaadun, pienen latenssin ja suurimman määrän yhteyksiä tukiasemaa kohti, mutta se on näistä kolmesta kallein hyödyntää.

3 WWW-sovelluspalvelut

Tässä tutkielmassa palvelulla tarkoitetaan itsensä kuvaavaa, alustariippumatonta sovellusta tai järjestelmää. Palvelu on hyvin määritelty ja itsenäinen eikä se ole riippuvainen muista palveluista. WWW-sovelluspalvelu puolestaan on teknologia, jonka avulla palvelut keskustelevat keskenään. WWW-sovelluspalveluiden avulla toteutetun järjestelmän arkkitehtuurin sanotaan olevan palvelukeskeinen.

Palvelukeskeisen tietojenkäsittelyn kannalta on tärkeä ensimmäisenä määrittää mitä tarkoitetaan kun puhutaan palvelusta. Papazogloun ja Georgakopoulusin artikkelissaan [41] esittämän määritelmän mukaan palvelut ovat itsensä kuvaavia, avoimia komponentteja, jotka tukevat nopeaa ja edullista hajautettujen sovellusten kehittämistä. Palvelun tarjoajan vastuulla on tarjota palvelun kuvaus, toteutus ja tuki. Palvelun asiakas voi olla toinen sovellus yrityksen sisällä tai ulkopuolella. Papazogloun artikkelissa[40] todetaan, että tämän takia palvelun tulisi olla teknologianeutraali, löyhästi kytketty eli palvelu ei ole riippuvainen muiden palveluiden tai komponenttien määritelmistä ja toiminnoista sekä sijainniltaan läpinäkyvä. Palvelu voi olla yksinkertainen palvelu, eli sen tarjoama tieto ja toiminnallisuus on saatavilla yhdestä paikasta, tai yhdistelmäpalvelu, joka yhdistää tietoa ja toiminnallisuutta usean eri palveluntarjoajan palveluista. Myös Barry [6, s. 19] painottaa riippumattomuutta muiden palveluiden tilasta tai asiayhteydestä. Palvelulta voidaan vaatia määrittelyt muun muassa seuraavissa kategorioissa, mutta jokaiselle palvelulle ei tarvitse määritellä jokaista osa-aluetta, vaan nämä päätetään käyttötarkoituksen mukaan [25]:

- suorituskyky
- kapasiteetti
- liiketoimintaorganisaatio
- riskit ja kiistakysymykset
- omistus
- luotettavuus
- turvallisuus
- vaikutus liiketoimintaan
- sietokyky
- palvelusopimus
- riippuvuudet

Tässä luvussa käydään läpi WWW-sovelluspalvelun määritelmä sekä tutustutaan kahteen tapaan toteuttaa WWW-sovelluspalveluita, WS-* ja REST.

3.1 Palvelusta WWW-sovelluspalveluun

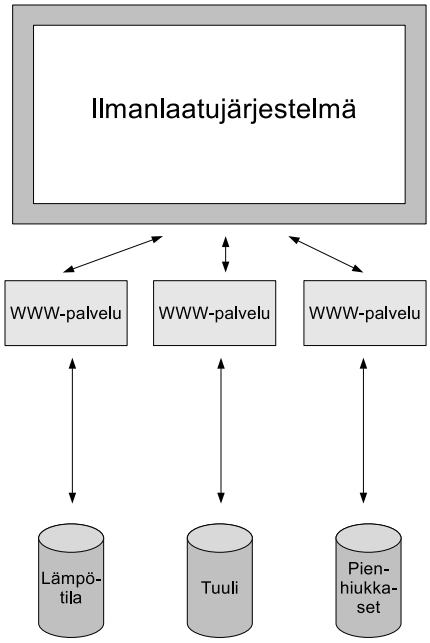
Aluksi WWW-sovelluspalveluita käytettiin verkkosivuilla tai osana olemassaolevia järjestelmiä. Useita WWW-sovelluspalveluita on nykyään jo saatavilla, mutta käyttäjien kannalta hyödyllinen tieto sijaitsee usein eri paikoissa erilaisten rajapintojen takana. WWW-sovelluspalveluteknologian avulla palveluita yhdistämällä saadaan koottua tietoa ja toiminnallisuutta käyttäjien saataville. WWW-sovellus-palveluita yhdistämällä voidaan rakentaa koko sivusto, esimerkiksi voidaan tarjota tietoa kaupungin ilmanlaadusta, joka kootaan usean eri WWW-sovelluspalvelun kautta. Vanhemmissa järjestelmissä oleva tieto saadaan myös WWW-sovelluspalve-lukäyttöön sovittimien avulla [6, s. 64].

WWW-sovelluspalveluiden toteuttamisen lisäksi yksi keskeinen asia on niiden julkaiseminen. Tätä varten voidaan esimerkiksi toteuttaa hakemisto, mistä asiakkaat voivat hakea niiden määritelmät ja käyttöohjeet. Tämän jälkeen sovelluksen toteutus riippuu tarjotusta rajapinnasta ja näin ollen muutokset tarjottuun rajapintaan vaativat myös muutoksia sovellukseen, joka käyttää sitä. WWW-sovelluspalvelut voidaan myös tarjota aina samanlaisen yksinkertaisen rajapinnan kautta. Tämä riippuu paljon tarjotun toiminnallisuuden tai resurssin luonteesta sekä halutun sovelluksen suoriutumiskyky- tai turvallisuusvaatimuksista.

Jotta WWW-sovelluspalvelusta voidaan puhua, sen tulee täyttää tietyt kriteerit. Jokainen WWW-sovelluspalvelu tulee olla tunnistettavissa URI:lla (Uniform Resource Identifier) eli merkkijonolla, jolla kerrotaan tietyn resurssin yksikäsitteinen sijainti. Sen tulee paljastaa ominaisuutensa internetiin standardoitujen protokollien kautta. WWW-sovelluspalvelu tulee voida toteuttaa itsekuvaavan, avoimiin internet-standardeihin perustuvan rajapinnan kautta [40].

WWW-sovelluspalvelut poikkeavat perinteisestä hajautetusta tietojenkäsittelystä monella tavalla. Ne voivat käyttää jo olemassa olevaa protokollaa kuten HTTP. WWW-sovellus-palveluiden tulee myös olla ohjelmointikielten suhteen läpinäkyviä eli ne toimivat yhdessä toteutuskielestä huolimatta. WWW-sovelluspalveluiden täytyy olla rakenteeltaan modulaarisia, eli uusia järjestelmiä voidaan rakentaa yhdistämällä olemassa olevia [28, s.2-3].

Nykyaikaiset ohjelmistojärjestelmät on toteutettu eri kielillä ja ne sijaitsevat erilaisilla alustoilla. Vanhoja ohjelmistoja on edelleen käytössä ja joidenkin hyödyntäminen voi olla yrityksille elintärkeää. Näiden järjestelmien uudelleenkirjoittaminen on useimpien toimijoiden resurssien ulottumattomissa. Jos esimerkiksi vanha COBOL -kielellä toteutettu järjestelmä tarjotaan WWW-sovelluspalveluna, se on muilla kielillä kirjoitettujen järjestelmien kanssa yhteentoimiva. [28] Kuvassa 3.1 näkyy mi-



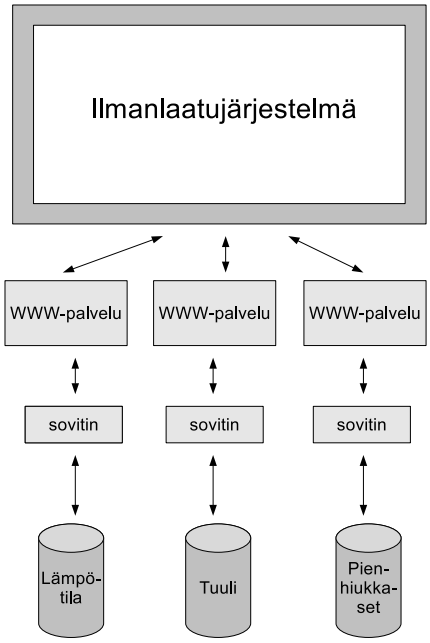
Kuva 3.1: WWW-sovelluspalvelut ilmanlaatujärjestelmässä

ten erilaiset mittaustiedot on yhdistetty isommaksi ilmanlaatujärjestelmäksi. Jos järjestelmät on kehitetty ennen WWW-sovelluspalveluteknologian käyttöönottoa, eri järjestelmiin tarvitaan WWW-sovelluspalvelusovittimet. Sovittimien sijainti arkkitehtuurissa näkyy kuvassa 3.2.

Snellin, Tidwellin ja Kulchenkon [53] mukaan WWW-sovelluspalvelun voidaan ajatella olevan verkon kautta saatavilla oleva rajapinta sovelluksen toiminnallisuuteen. Toteutettu WWW-sovelluspalvelu ja sen sovittimet sijaitsevat sovelluksen ja sen käyttäjän välissä.

WWW-sovelluspalvelun määritelmiä on käytössä useita ja useimmat niistä ovat melko laajoja. W3C:n (World Wide Web Consortium) määritelmää [8] mukaillen tässä tutkielmassa WWW-sovelluspalvelulla tarkoitetaan tietokoneiden verkon yli tapahtuvaa yhteentoimivuutta tukemaan suunniteltua ohjelmistojärjestelmää. Tällaisella järjestelmällä on tarkasti määritelty rajapinta, joka on kuvattu esimerkiksi tietokoneen ymmärtämässä WSDL -muodossa (Web Service Definition Language, määritelty W3C:n toimesta [15]). Muut järjestelmät ovat vuorovaikutuksessa WWW-sovelluspalvelun kanssa kuvauksen mukaisesti esimerkiksi SOAP -viestein (aikaisemmin Simple Object Access Protocol), jotka tyypillisesti välitetään HTTP-protokollan avulla XML -muodossa, yhdessä muiden web-standardien kanssa.

WSDL on muodostunut käytännön standardiksi sekä SOAP:ia käyttäville palveluille että yleisille verkkopalveluille. [25] Suosiossa SOAP-pohjaisten WWW-sovel-



Kuva 3.2: WWW-sovelluspalvelut ilmanlaatujärjestelmässä sovittimien kanssa

luspalvelui-den rinnalle ja osittain niiden ohi on noussut REST [46] eli Representational State Transfer [19]. REST:n suosion nousua selittänee se seikka, että se koetaan kevyemmäksi ja helpommaksi oppia kuin perinteiset SOAP -pohjaiset ratkaisut, kuten Guinardin, Ionin ja Mayerin tutkimuksesta [23] selviää.

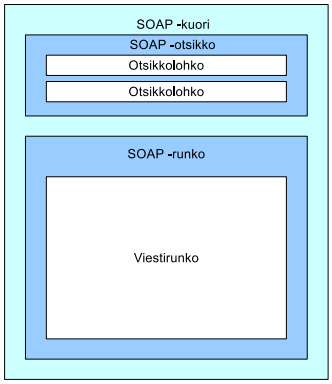
3.2 WWW-sovelluspalveluteknologiat

Tässä alaluvussa perehdytään syvällisemmin SOAP- ja REST-ratkaisuihin. Ensin käydään lyhyesti läpi SOAP sekä siihen liittyvä WSDL -kuvauskielet. Tämän jälkeen esitellään REST, käydään läpi URI- ja resurssisuunnitteluun liittyviä periaatteita sekä HTTP:n tarjoama rajapinta.

3.2.1 SOAP, WSDL ja UDDI

SOAP, aikaisemmin Simple Object Access Protocol, nykyään vain SOAP [21] [6], on XML -kieltä. SOAP käsittelee palvelua proseduurien kautta, toisin sanottuna jokainen pyyntö suorittaa jonkin operaation [46].

SOAP -viestien rakenne on määritelty XML Schemalla [22]. Snell et al esittelevät SOAP -viestin rakenteen kirjassaan [53] ja rakenne näkyy kuvassa 3.3. XML -viestien etuna on sovellus-, käyttöjärjestelmä- ja ohjelmointikieliriippumattomuus.



Kuva 3.3: SOAP -viestin rakenne

SOAP -viesti koostuu Envelope -elementistä, eli kuoresta, jonka sisällä on valinnaiset otsikkotiedot ja pakollinen viestirunko. Otsikkoelementti (Header) sisältää viestin käsittelyä ohjaavat tiedot. Viestirunko (Body) sisältää toimitettavan ja käsiteltävän viestin. Viestin sisältö voi olla mitä tahansa, mikä voidaan esittää XML -muodossa. SOAP -kuoressa tulee olla vain yksi Body -elementti. Jos kuori sisältää otsikkotiedot, niiden tulee olla ensimmäisena kuoren sisällä, ennen viestirunkoa. Myös otsikkotietojen tulee olla XML -muodossa. [53]

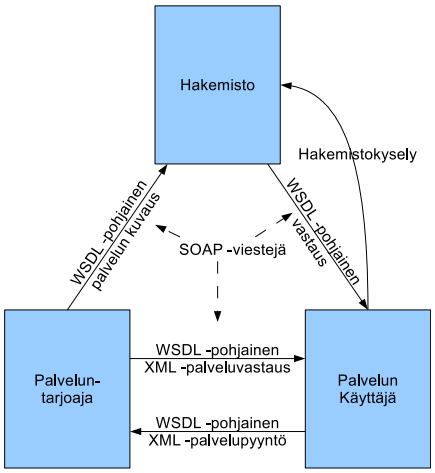
Jos palvelinpäässä tapahtuu virhe pyyntöä suorittaessa, myös virheilmoitus toimitetaan SOAP-viestinä. Viestin sisältöön kuuluu virhekoodi, virheen kuvaus, virhetoiminnon suorittanut osa sekä virheen yksityiskohdat. Virhekoodityyppejä ovat VersionMismatch, MustUnderstand, Server ja Client. Näiden lisäksi voidaan käyttää sovelluskohtaisia virhekoodeja. [53] Käytännössä SOAP:n virheilmoituksia käytetään ohjelmointipoikkeusten lähettämiseen palvelun ja sen käyttäjän välillä [58].

SOAP -pohjainen palvelu vaatii kuvauksen. Koska SOAP perustuu RPC-tyyliin, rajapintakuvausta varten voidaan käyttää rajapinnan kuvauskieltä eli IDL:a (Interface Description Language) [33][36]. SOAP itsessään ei ota kantaa palvelun kuvaukseen ja yleisesti palvelun rajapinnan kuvaukseen käytetään WSDL -kieltä. Kuvaukseen kuuluu mitä palvelu tekee ja miten palvelua voidaan käyttää. [53] Curbera et al [16] tarkentavat kuvauksen kattamaan sovelluksen abstraktin rajapinnan ja täsmälliset protokollariippuvaiset seikat. Tämä erottelu mahdollistaa samankaltaisen sovellustoiminnallisuuden tarjoamisen eri päätepisteissä.

WSDL:n käytön takia kytkentä on yleensä tiukka. Järjestelmän osat ovat hyvin

riippuvaisia toisten osien toteutuksesta. Rajapintakuvauskielten ansiosta voidaan tuottaa valmista koodia automaattisesti, mutta jos rajapinnan kuvausta muutetaan myöhemmin, siitä riippuvaiset ohjelmat eivät välttämättä enää toimi. [58]

Palveluntarjoaja toimittaa kuvauksen palvelusta palveluhakemistoon. Palvelun asiakkaat voivat hakea palvelukuvauksen, jonka perusteella tietävät mitä operaatioita palvelu tarjoaa. Kuvassa 3.4 esitetään WSDL-viestiliikenne [6].



Kuva 3.4: Palvelukeskeisen arkkitehtuurin viestiliikenne

SOAP -pohjaisten WWW-sovelluspalveluiden löytäminen tapahtuu UDDI:n (Universal Description, Discovery and Integration [6]) avulla. Curberan et al artikkelin [16] mukaan UDDI:n mukaisen palvelurekisterin pitää tarjota:

- jokaisesta palvelusta tarjottava tieto ja miten se koodataan
- kysely- ja päivitysrajapinta rekisterille, joka kuvaa miten tietoa päivitetään ja miten siihen päästään käsiksi

Samassa artikkelissa[16] jaetaan UDDI:n tallentama tieto kolmeen eri tyyppiin:

- “valkoiset sivut” eli palvelun nimi ja yhteystiedot
- “keltaiset sivut” eli palvelu- ja liiketoimintapohjainen luokittelu
- “vihreät sivut” eli palvelun tekniset tiedot

Tarkemmin jaoteltuna UDDI -rekisterimerkintä sisältää businessEntity -osan ja businessServices -osan. BusinessEntity sisältää tiedot WWW-sovelluspalvelun tarjoajasta ja voivat sisältää mm. yhteystiedot, toiminta-alan ja listan tarjotuista palveluista. Business services sisältää WWW-sovelluspalvelun kytkentätiedot sekä sovelluspalvelun tyypin ja lajittelutiedot. Jokainen businessEntity ja businessServices sisältää yksilöllisen UUID -tunnisteen (Universally Unique Identifier). [53]

Binding template -osio sisältää palvelun tekniset kuvaukset ja vastaa WWW-sovelluspalvelun varsinaista toteutusta. Palvelulla voi olla useampi eri toteutus esimerkiksi eri protokollalla tai eri osoitteessa, tällöin tarjotaan jokaiselle toteutukselle oma binding template. [53] [16] Lisäksi rekisteriin tallennetaan tModel-tietona tarvittavat konseptit omina tietotyyppeinään, joihin voidaan sitten viitata palveluista.

3.2.2 REST

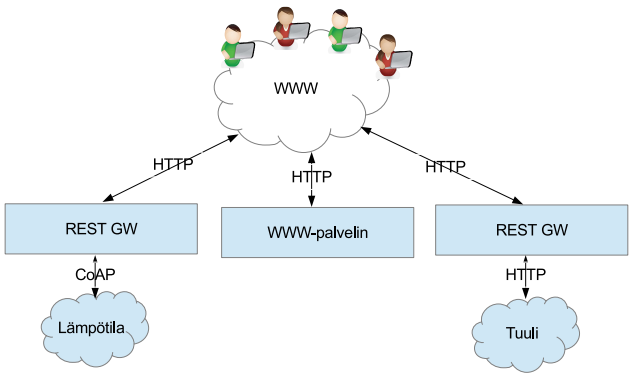
REST eli Representational State Transfer on arkkitehtuurityyli, joka on suunniteltu hajautetuille hypermediajärjestelmille. Se on johdettu useasta eri verkkopohjaisesta arkkitehtuurityylistä ja siihen on yhdistetty rajoitteita, jotka määrittelevät yhdenmuotoisen rajapinnan. [19] Tässä alaluvussa käydään läpi REST -arkkitehtuurityylissä käytettäviä teknologioita, resurssien määrittelyä, hyviä URI-suunnittelutapoja sekä REST-arkkitehtuurityylin rajapinta.

Fieldingin [19, s. 76] määrittelemä REST-arkkitehtuurityyli on johdettu kuuden rajoitteen avulla, joita ovat asiakas-palvelin-suhde, tilattomuus, välimuistin käyttö, yhdenmuotoinen rajapinta, kerrostettu järjestelmä ja code-on-demand, eli koodin tarjoaminen tarvittaessa.

REST-arkkitehtuurityylin yksi rajoitteista on yksinkertainen, samanmuotoinen rajapinta joka palvelulla [19]. Kyseessä ei ole etäproseduurikutsuihin perustuva järjestelmä [32], mutta REST-arkkitehtuurityylin mukaiset palvelut voivat tarjota vastaavan toiminnallisuuden HTTP-objektien eli resurssien ja näihin kohdistuvien operaatioiden kautta [46]. Vaikka useimmiten REST:n yhteydessä puhutaan HTTP:stä, arkkitehtuurityyli ei kuitenkaan itsessään määrittele mitä protokollaa pitää käyttää. [18][46].

Zeng, Guo ja Cheng esittävät artikkelissaan[60] miten erilaisia järjestelmiä voidaan kytkeä webiin. Järjestelmät voidaan kytkeä suoraan tai epäsuoraan riippuen niiden ominaisuuksista. Suora yhdistäminen vaatii laitteelta ainakin IP-osoitteen sekä sovellustason yhdistettävyyttä. Epäsuoraa yhteyttä tarvitaan, kun liitettävät laitteet joko eivät ole tarpeeksi tehokkaita siihen tai ei ole tarvetta yhdistää jokaista laitetta kuten esimerkiksi sensoriverkkojen tapauksessa. Tällöin käytetään väli-

tyspalvelinta tai yhdyskäytävää. Kuvassa 3.5 näkyy miten eri protokollia käyttävät REST-tyyliset palvelut yhdistyvät WWW:iin.



Kuva 3.5: REST ja WWW

Toisin kuin SOAP-pohjaiset ratkaisut, REST-arkkitehtuurityyli ei määrittele palvelurekisteriä. Lanthalerin ja Gütlin [33] mukaan olisi mahdollista kerätä hakukoneisiin tietoa palveluista, jos ne olisi kuvattu käyttäen esimerkiksi hRESTS -teknologiaa (HTML for RESTful Services). Tätä voitaisiin tukea myös semanttisilla kuvauksilla, kuten SA-REST tai MicroWSMO. Lanthaler ja Gütl esittävät vaihtoehdoksi myös WADL:n (Web Application Description Language) käyttöä palvelukuvauksiin. Tällä hetkellä mitään hyväksyttyä standardia ei kuitenkaan ole käytössä.

REST-palvelun resurssisuunnittelun apuna voidaan käyttää Richardsonin ja Rubyn kirjassaan [46, luvut 5 ja 6] esittämää tekniikkaa. Resurssin voidaan ajatella olevan substantiivi ja REST-arkkitehtuurityylin mukaisesti käsitellään resurssin esityksiä. Resurssiksi kelpaa mikä tahansa, joka voidaan kokea niin tärkeäksi, että siihen halutaan viitata tai sitä halutaan käsitellä. Joka resurssilla tulee olla ainakin yksi URI. Resurssilla voi ainakin hetkellisesti olla useampi URI.[46]

Richardson ja Ruby [46] esittävät resurssien suunnitteluun yksinkertaisen menetelmän:

1. Selvitetään käsiteltävä tieto
2. Jaetaan tieto resursseiksi
3. Nimetään resurssit URI:lla
4. Paljastetaan jokin yhdenmukaisen rajapinnan osajoukko

5. Suunnitellaan asiakkaalta hyväksyttyjen resurssien esitykset
6. Suunnitellaan asiakkaalle tarjottujen resurssien esitykset
7. Yhdistä resurssit olemassa oleviin resursseihin hyperlinkeillä
8. Mieti mitä tyypillisessä tapahtumaketjussa tulee tapahtua
9. Mieti mikä voi mennä pieleen

Sovelluksessa käsiteltävä tieto voi olla jotain jo olemassa olevaa, joka halutaan paljastaa palvelun kautta tai se määritellään palvelua suunnitellessa. WWW-sovelluspalvelut tarjoavat yleensä kolmenlaisia resursseja, etukäteen määritellyt, sovellukselle tärkeät resurssit; palvelun paljastamien objektien resurssit ja käsitellylle tietojoukolle suoritettujen algoritmien tulokset.

URI:en nimeämiseen on kolme kokemuspohjaista sääntöä. Ne voidaan nimetä siten, että ne osoittavat resurssien hierarkian, kuten /vanhempi/lapsi. Jos hierarkiaa ei ole, resurssit voidaan esittää välimerkeillä eroteltuina. Kolmas tapa on kyselymuuttujien käyttäminen algoritmin syötteisiin viitaten, kuten /etsi?q=kala. URI:en nimissä ei ole soveliasta käyttää operaatioiden nimiä, esimerkiksi deleteUser ei ole hyvä valinta [37].

Seuraavaksi tulee suunnitella esityksen muoto ja tarjotut otsikkotiedot. Asiakkaalta hyväksyttyjen esitysten tulisi mielellään olla samassa muodossa kuin asiakkaalle toimitettu esitys. Asiakkaalle tarjotut resurssin esitykset sisältävät resurssin tilan sekä mahdolliset linkit uusiin resursseihin. Esityksen muoto voi olla pelkkää tekstiä tai jokin rakenteinen muoto, kuten JSON tai XML. Esimerkiksi lämpötilatieto voidaan esittää JSON -muodossa kuten esimerkissä 3.1. Resurssin esityksen muotoa valitessa kannattaa huomioida käyttötarkoitus.

Listing 3.1: Esimerkki lämpötilatiedon esityksestä

```
{
  "SensorValue":
  {
      "SensorId":89,
      "SensorValue":"161",
      "Time":"2015-04-09T18:16:00",
      "Variable":"Celsius"
  }
}
```

Resursseissa tulisi myös mahdollisuuksien mukaan olla linkkejä toisiin resursseihin. Näin voidaan siirtyä palvelun resurssien välillä. Tämä varmistaa, että REST:n tavoitteena oleva hypermedian käyttö itse sovelluksen tilan moottorina toteutuu [19]. Toisin sanottuna sovelluksen saatavilla olevat tilat tunnistetaan tarjolla olevista linkeistä [58, s. 13].

Seuraavana vaiheena tulee miettiä miten toimitaan oikein suoritetun pyynnön kohdalla. Pelkästään resurssin tilan lukevat operaatiot ovat yksinkertaisia. Asiakas saa vastauksena HTTP-koodin 200 eli "OK" [18], otsikkotiedot ja pyytämänsä resurssin esityksen haluamassaan muodossa. PUT-operaation tuloksena voi tulla 201 eli "Created" jos resurssia ei ollut olemassa tai 200, jos tietoja vain päivitetään.

Virhetilanteita on monenlaisia. Todennäköisimmät tilanteet ovat olemattoman resurssin GET-pyyntö, jolloin palautetaan 404. Asiakkaalle voidaan myös palauttaa linkki resurssiin, joka muistuttaa hänen haluaamaansa resurssia. Tämä edellyttää eri HTTP-vastauskoodia, joka tässä tapauksessa on 303. Väärää resurssia haettaessa voidaan myös käyttää virhekoodia 400 eli "Bad Request". Virhekoodin lisäksi voidaan tarjota linkki johonkin olemassa olevaan resurssiin tai esimerkiksi palvelun juureen.

3.2.3 Rajapinta

Yleisimmin käytetyt HTTP-metodit ovat GET ja POST. Näiden lisäksi usein käytettyjä ovat PUT ja DELETE. Webberin[58] mukaan näitä metodeja voi verrata CRUD-eli Create, Read, Update, Delete -metodeihin. Yksinkertaistetusti GET hakee resurssin tiedot, POST luo uuden resurssin, PUT päivittää resurssin tiedot ja DELETE poistaa kyseessä olevan resurssin. [46][58].

POST-metodi mahtuu REST-arkkitehtuurityylin rajoituksiin, mutta sillä on myös laajempi toimintatarkoitus [46]. Dokumentissa RFC2616 [18] mainitaan, että POST -metodi sallii muun muassa tietojen lisäämisen olemassa olevaan resurssiin, viestin lähettämisen uutisryhmään tai postituslistaan, lomakkeelta saadun tiedon tarjoamisen tietoa käsittelevälle prosessille tai tietokannan laajentamisen. POST-metodin todellinen toiminta on kuitenkin palvelimen päätettävissä [46].

Näiden lisäksi käytettävissä ovat myös OPTIONS ja HEAD. OPTIONS kertoo mitä HTTP-metodeja käytetty resurssi tukee. Riippuen pyynnön mukana lähetetyistä tiedoista, esimerkiksi autentikointitiedot, resurssi saattaa antaa eri tuloksen OPTIONS-pyynnölle. HEAD palauttaa resurssin metadataesityksen. Tätä voi käyttää esimerkiksi tarkistaakseen onko haluttu resurssi olemassa hakematta mahdollisesti suurikokoista resurssia. [46]

GET ja HEAD ovat oikein käytettyinä ja toteutettuina turvallisia operaatioita. Asiakas voi pyytää moneen kertaan resurssin esityksen tai sen metatiedot eikä itse resurssin tilan pitäisi muuttua mitenkään. Palvelimen puolella muutoksia saattaa tapahtua, esimerkiksi käyntilaskuri voi kasvaa, mutta tällaiset muutokset eivät ole asiakkaan kannalta merkittäviä. [46]

GET, HEAD, PUT ja DELETE tuottavat saman tuloksen joka kerta, kun ne suoritetaan eli kyseessä olevat metodit ovat idempotenttisia. Resurssin pyytäminen palvelimelta moneen kertaan ei muuta sitä mitenkään. Resurssin päivittäminen samoilla tiedoilla ei myöskään tuota erilaista lopputulosta. [46] Vastauksena asiakas saa resurssin esityksen, joka voi olla esimerkiksi HTML-, JSON- tai JPEG -tiedosto, esityksen metatiedot, jotka kertovat muun muassa kyseessä olevan mediatyypin sekä joskus myös itse resurssiin liittyvää metatietoa. [19]

3.3 Vertailu

Tässä alaluvussa käydään läpi WWW-sovelluspalveluissa käytetyt eri arkkitehtuurit sekä vertaillaan tarkemmin REST- ja SOAP-pohjaisten WWW-sovelluspalveluiden eroja. Tarkemmin tarkastellaan rajapintoja, kytkentää ja monimutkaisuutta. Tämän lisäksi vertaillaan myös kyseessä olevia WWW-sovelluspalveluteknolo-gioita käyttäen kriteereinä kytkennän tiukkuutta, muistinkäyttöä, sovelluskehityksen monimutkaisuutta, viestiliikenteen nopeutta ja tietoturvaa langattomien sensoriverkkojen näkökulmasta käyttäen luvussa yksi havaittuja haasteita ja rajoitteita.

3.3.1 Arkkitehtuurit

Richardsonin ja Rubyn mukaan [46] WWW-sovelluspalveluissa käytetyt arkkitehtuurit voidaan jakaa kolmeen eri tyyppiin: RESTful, RPC ja REST-RPC.

RESTful -termiä käytetään REST-arkkitehtuurityylin mukaisesta järjestelmästä. Tämä tarkoittaa sitä, että metoditieto on HTTP-metodissa ja rajaustiedot URI:ssa.[46, s. 13] Jotta palvelua voidaan sanoa REST-tyyliseksi, on tärkeää, että HTTP-metodien alkuperäisiä määritelmiä kunnioitetaan ja käytetään oikein.[28, s. 123]

RPC-tyylinen WWW-sovelluspalvelu toimii pääasiassa niin, että se vastaanottaa asiakkaalta kuoren täynnä tietoa ja lähettää vastauksena samanlaisen kuoren takaisin [46, s. 14]. SOAP on yksi käytettävissä oleva kuori. SOAP-dokumentin lähettäminen HTTP:n yli käärii SOAP -kuoren ja sen sisältämän tiedon HTTP -kuoreen ja siirtää sen käyttäen POST-metodia [50]. Puhdasta XML:ää voidaan myös käyttää kuorena [28]. Jokainen RPC-tyylinen WWW-sovelluspalvelu määrittelee oman raja-

pintansa, toisin kuin REST-tyyliset palvelut, jotka käyttävät HTTP:n rajapintaa [46]. RPC-tyylinen palvelu paljastaa tyypillisesti yhden URL:n ja tukee vain yhtä HTTP:n metodia, eli POST-metodia. [46, s. 15]

REST-RPC on RPC-tyylinen palvelu, joka käyttää HTTP:aa viestiensä kuoriformaattina, mutta esimerkiksi käytetty metodi ja rajaustieto sijaitsevat URL:ssa. Tällainen palvelu on jossain RPC:n ja RESTful -palvelun välissä.[46, s. 16] Tämäntyylisistä palveluista voidaan myös käyttää nimitystä HTTP + Plain Old XML tai Service Trampled REST [46, s. 21].

3.3.2 SOAP vs. REST

Tyypillisesti SOAP+WSDL paljastaa RPC-tyylisesti sisäisiä algoritmeja monimutkaisen, joka palvelulle erilaisen rajapinnan kautta [46, s. 19]. REST-tyyliset palvelut puolestaan paljastavat tietoa yksinkertaisen ja aina samanlaisen rajapinnan kautta [32].

Pohjimmiltaan SOAP on hyvin yksinkertainen ja kuvaa vain XML-kuoren ja prosessimallin, jonka mukaan viestit kulkevat verkossa [58, s. 376]. Kalinin mukaan [28, kpl. 7] SOAP:iin liittyvillä standardointipyrkimyksillä on monimutkaistettu asiaa. Kuitenkin SOAP yksinkertaistaa montaa seikkaa, muun muassa koska asiakas ja palvelu vaihtavat keskenään XML-dokumentteja, joiden käsittelyyn löytyy valmiita kirjastoja. Guinardin et al tutkimuksen mukaan [23] REST koetaan helpoksi oppia ja käyttää, koska se perustuu tunnettuihin teknologioihin kuten HTTP. Vastaavasti SOAP koetaan jopa liiallisen monimutkaiseksi, mutta samaan aikaan sitä pidetään monipuolisempana ja turvallisempana.

Pautasson ja Wilden artikkelissa [42] on vertailtu palveluiden kytkentää usean eri osa-alueen kautta, kuten palveluiden löytäminen, tunnistautuminen, koodin generointi ja niin edelleen. Jokainen osa-alue arvioidaan kytkennän kannalta joko tiukkana, suunnittelukohtaisena tai löysänä kytkentänä. Artikkelin perusteella REST-pohjaiset palvelut ovat pääasiassa löysästi kytkettyjä ja RPC-pohjaisilla on taipumusta tiukkaan tai suunniteluperustaiseen kytkentään.

Artikkelissaan Priyantha et al [44] tarkastelevat WWW-sovelluspalveluiden viestikokoja. Heidän mukaansa SOAP 1.1 ja 1.2 kapseloivat viestit isommiksi kuin normaali HTTP, jota voidaan käyttää REST-arkkitehtuurityylin mukaisissa järjestelmissä.

Artikkelissaan zur Muehlen, Nickerson ja Swenson [61] vertaavat SOAP- ja REST-pohjaisia järjestelmiä. Heidän mukaansa SOAP:n etuna on tiukka kytkentä ja tästä seuraava mahdollisuus debuggaukseen ja testaukseen. REST-pohjaisten järjestel-

mien etuna on skaalattavuus sekä yhdenmukaisesta rajapinnasta ja pienestä operaatiomäärästä johtuva operaatioiden käytön keveys.

3.3.3 Soveltuvuus

WWW-sovelluspalveluteknologian valinta riippuu hyvin pitkälle käyttötarkoituksesta. Tämän työn tarkoitus on selvittää muun muassa tarjolla olevien teknologioiden soveltuvuutta langattomien sensoriverkkojen käyttöön, joten vertailtaviksi kriteereiksi nostetaan viestien käsittelyyn liittyvä virran- ja muistinkäyttö, operaatioiden raskaus sekä tiedonsiirtoon käytettävä kaista.

Pautasson et al[43] sekä Landren ja Wesenbergin [32] mukaan WS-* -palvelut sopivat Enterprise application integration -käyttöön, jossa esimerkiksi palvelun tasolta voidaan vaatia enemmän. Pautasson [43] mukaan taasen REST-pohjaiset palvelut sopivat ad hoc -integraatioon WWW:n yli.

Aihkisalon ja Paason [1] mukaan REST on yksinkertaisempi, kevyempi ja täten tehokkaampi, mutta ei välttämättä kykene tarjoamaan riittävää ratkaisua toiminnallisesti monimutkaisiin järjestelmiin. Sulautetuilla laitteilla REST-arkkitehtuurityylin eduksi voidaan Shelbyn [50] mukaan laskea myös pienemmät kustannukset, yksinkertaisempi parsiminen, tilattomuus ja tiukempi integraatio HTTP:n kanssa.

REST-pohjaisen sovelluksen nopeuteen vaikuttaa ehdollinen HTTP:n GET -operaatio, koska jos haluttu resurssin esitys ei ole muuttunut, sitä ei tarvitse hakea uudestaan. [59] Toisessa tutkimuksessa [1] ilmenee selvästi, että hitammillaankin REST on useita kertoja SOAP-pohjaista toteutusta nopeampi. Nopeuteen vaikuttaa paljon lähetetyn paketin formaatti, mutta se ei selitä kaikkea. SOAP:n hitautta ei selitä pelkkä XML:n käyttö, koska REST-XML ei ollut merkittävästi hitaampi kuin muut REST-tyyliä käyttävät formaatit. Kyseisessä tutkimuksessa ei kuitenkaan mitattu viestipaketista riippuvaa tiedonsiirtonopeutta, vaan järjestelmien sisäistä viivettä viestien käsittelyssä.

Yazarin ja Dunkelsin tutkimuksen [59] mukaan REST-tyyliset ratkaisut kuluttavat myös vähemmän virtaa. Myös tämä on ehdollisen HTTP GET -toiminnon ansiota, koska välimuistissa olevaa tietoa ei tarvitse siirtää joka kerta, kun sitä tarvitaan. Toisin kuin REST, SOAP ei tarjoa välimuistia, koska yleensä käytetään HTTP POST -viestejä. [58, s. 377]Yazarin ja Dunkelsin [59] tutkimuksen mukaan SOAP-pohjainen toteutus myös vaatii enemmän muistia käyttöönsä, kuin REST. Tämä johtuu pääasiassa SOAP:n tarvitsemasta XML-parserista.

Näiden tulosten valossa näyttää siltä, että REST on sopivampi valinta langattomien sensoriverkkojen käyttöön. SOAP on kuitenkin käyttökelpoinen valinta, var-

sinkin jos www-sovelluspalvelu toteutetaan tehokkaammalla alustalla, jossa ei tarvitse huolehtia virrankulutuksesta, laskentatehosta tai muistinkäytöstä.

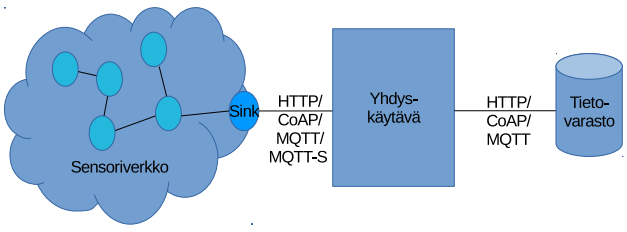
4 WSN WWW-sovelluspalveluna

Tässä kappaleessa esitellään kuinka langattomia sensoriverkkoja voidaan yhdistää toisiin järjestelmiin ja internetiin. Tämän työn painopiste on WWW-sovelluspalveluteknologioissa, joten eri teknologioita tarkastellaan WWW-sovelluspalveluiden näkökulmasta. Jotta sensoriverkkojen tuottama tieto olisi käytettävissä mahdollisimman laajasti, tulee se tarjota helposti käytettävällä tavalla sekä käyttöjärjestelmästä ja ohjelmointikielestä riippumattomassa muodossa. Tämän lisäksi tiedon esitys tulee valita tarkasti resurssien rajallisen määrän takia. Tätä varten on tarjolla muun muassa WSN OpenAPI:n käyttämä määritelmä. Tämän lisäksi myös tiedon esitysformaatin valinta on tärkeä. Yleisimmin käytettyjä ovat XML ja JSON. Myös käytetty tiedonsiirtoprotokolla tulee valita huolella. Tarjolla on HTTP:n lisäksi myös CoAP ja MQTT.

Aiemmin esiteltiin kuvassa 2.1 yleinen sensoriverkon arkkitehtuuri. Tässä luvussa keskitytään tarkemmin tällaisen verkon yhdyskäytävään ja siihen liittyviin asioihin.

4.1 Sovellusprotokollat

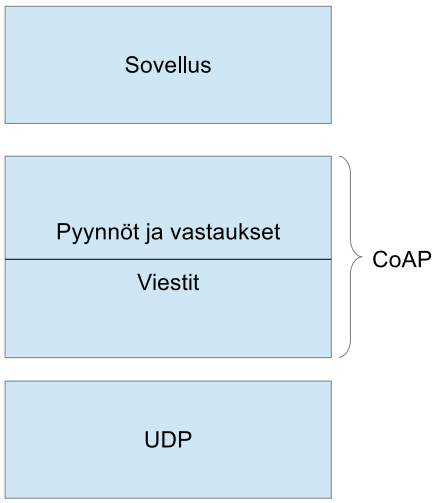
Koska HTTP on käsitelty REST-arkkitehtuurityylin yhteydessä, tässä alaluvussa keskitytään CoAP- ja MQTT-protokolliin. Kuten kuvassa 4.1 näkyy, eri tiedonsiirtoprokollia voidaan käytää sensoriverkon mittaustietojen siirtämiseen arkkitehtuurin eri osien välillä. Tietovarastona voi toimia esimerkiksi relaatio- tai NoSQL -tietokanta.



Kuva 4.1: Sensoriverkko, yhdyskäytävä ja protokollat

4.1.1 CoAP

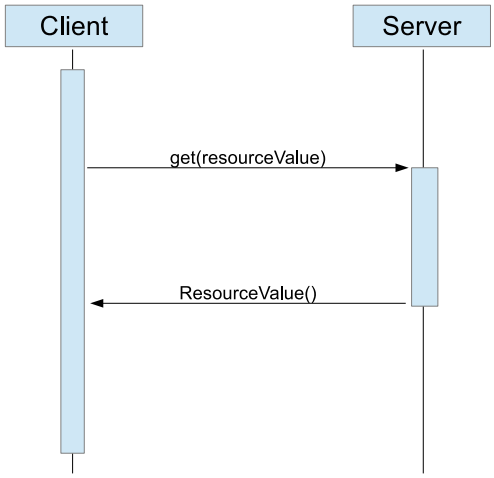
CoAP eli Constrained Application Protocol on kehitetty resurssirajoitteisia laitteita varten. Koska HTTP on suhteellisen tilaa vievä ja raskas verkonkäytöltään, se ei kaikilta osin ole sopiva resurssirajoitteisten laitteiden kanssa käytettäväksi. Tämän takia Borman et al esittivät sille tähän käyttötarkoitukseen korvaajaksi CoAP:n[9]. CoAP on suunniteltu geneeriseksi web-prokotollaksi rajoitettuihin ympäristöihin. Sen tarkoituksena on toteuttaa M2M -sovelluksille optimoitu REST:n osajoukko. Tämän lisäksi CoAP mahdollistaa myös sisäänrakennetun palveluiden ja resurssien löytämisen sekä tarjoaa tuen ryhmälähetyksille ja asynkroniselle viestinnälle. Kuvassa 4.2 näkyy mihin kohtaan CoAP sijoittuu suhteessa sovellus- ja tiedonsiirtoprotokollaan.[5



Kuva 4.2: CoAP:n kerrokset

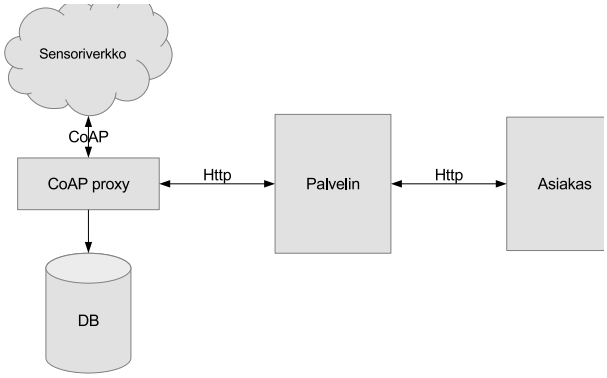
CoAP tarjoaa request-response -mallin mukaisen toiminnallisuuden. Malli näkyy kuvassa 4.3 ja tarjoaa HTTP:n GET-, PUT-, POST- ja DELETE-metodit sekä samanlaiset vastauskoodit. CoAP käytti alunperin tiedonsiirtoprotokollana UDP:tä TCP:n sijaan, joka toisin kuin TCP, ei automaattisesti lähetä isompaa hyötykuormaa useana palasena. Tämän takia isompia hyötykuormia varten CoAP tarjoaa block-toiminnallisuuden, jossa resurssin esitys siirretään useammassa pyyntö-vastaus -parissa[9].

IoT-käytössä tulee huomioida myös virrankäyttö. Tämä otetaan CoAP:ssa huomioon Observe-toiminnolla. Tällöin asiakasohjelma saa ilmoituksen kun resurssin arvo päivittyy eikä sen tarvitse kysellä palvelimelta jatkuvasti viimeisintä arvoa. Kuten Thangavel et al artikkelissaan[56] toteavat, tämä on publish/subscribe -mallin mukainen, mutta tilauksen kohteena on jokin tietty resurssi. Langattoman sensori-



Kuva 4.3: Request-response -malli

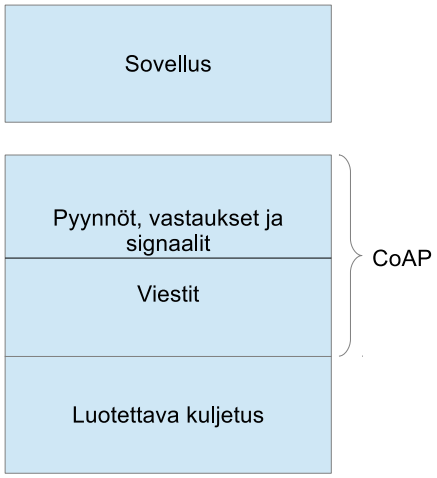
verkon ja CoAP:n yhteydet näkyvät kuvassa 4.4. Määritelmänsä mukaan [51] CoAP tukee myös palveluiden ja resurssien löytämistä /.well-known/core ja /.well-known-/scheme URI-osoitteiden kautta.



Kuva 4.4: REST, CoAP ja sensoriverkko

Koska UDP:tä ei pidetä tarpeeksi luotettavana, CoAP tarjoaa oman mekanismin luotettavuuden takaamista varten. Viestit voidaan luokitella varmistettaviksi (confirmable) tai ei-varmistettaviksi (non-confirmable). Varmistettava viesti vaatii kuittauksen (acknowledgement) vastaanottajalta[51]. Myöhemmin CoAP laajennettiin

tukemaan myös TCP:tä, TLS:ää ja Websockets -teknologiaa[10]. TCP:tä käytettäessä CoAP:n viestikerroksen ei tarvitse tukea viestien kuittausta tai havaita mahdollisia kaksoiskappaleita viesteistä. Tällöin viestin tyyppi ja viestin tunniste voidaan jättää pois. TCP:n, TLS:n ja Websocketsin käyttöönoton myötä CoAP:n protokollapino[10] voidaan esittää yleisesti kuten kuvassa 4.5.



Kuva 4.5: CoAP:n kerrokset luotettavalla kuljetuskerroksella

4.1.2 MQTT

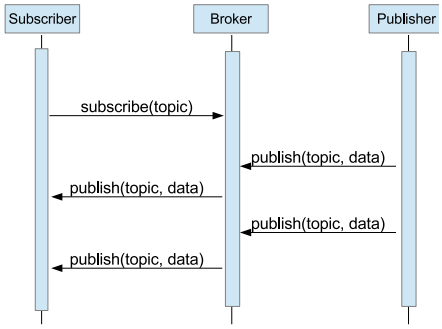
Toinen tarkasteltu protokolla on nimeltään MQTT eli Message Queue Telemetry Transport, joka on myös suunniteltu resurssiköyhät ympäristöt mielessä. Määritelmän[5] mukaan MQTT on niin sanottu *publish/subscribe* -mallin mukainen tiedonsiirtoprotokolla. Mallin toiminta näkyy kuvassa 4.6. MQTT:n tapauksessa asiakas voi toimia sekä publisher- että subscriber -rooleissa.

MQTT edellyttää tiedonsiirtoprotokollan tarjoavan järjestetyn ja häviöttömän datavirran asiakkaalta palvelimelle ja toisinpäin, MQTT voi siis käyttää protokollana muun muassa TCP/IP:tä, TLS:ää tai WebSocketia. UDP hylättiin, koska se ei täytä vaadittuja kriteereitä, koska se voi toimittaa tiedon väärässä järjestyksessä tai kadottaa osan siitä.

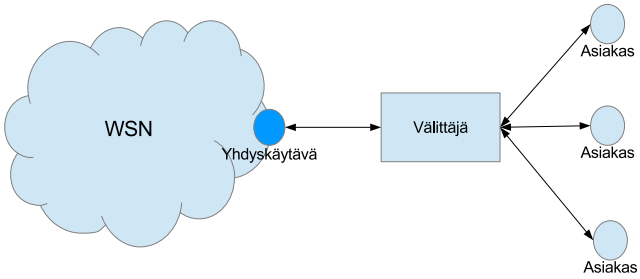
MQTT- protokolla käyttää aiheisiin perustuvaa *publish/subscribe*- arkkitehtuuria. Kun asiakas julkaisee viestin V, joka liittyy aiheeseen A, jokainen aiheen A tilannut asiakas saa viestin V. *Publish/subscribe* -mallissa niin sanottu välittäjä (broker) huolehtii siitä, että asiakkaiden tieto päätyy oikeille tilaajille. Välittäjä ylläpitää tilaustietoa ja välittää julkaistut tiedot aiheiden mukaan niiden tilaajille. Välittäjä on

siis arkkitehtuurissa asiakkaiden ja mittausverkon välissä. Kuvassa 4.7 näkyy mihin välittäjä sijoittuu WSN-arkkitehtuurissa.

MQTT käsittelee tilauksia aiheiden nimien ja niin sanottujen jokerikorttien avulla. Jokerikorttien avulla voidaan tilata samankaltaisen aiheiden päivitykset yhdellä kertaa. Esimerkiksi tilaus voi olla aiheeseen *urheilu/jalkapallo/pelaaja1/#*. Tällöin tilaaja saisi päivitykset aiheista *urheilu/jalkapallo/pelaaja1*, *urheilu/jalkapallo/pelaaja1/pisteet ja urheilu/jalkapallo/pelaaja1/rangaistukset*. Myös muita jokerimerkkejä on mahdollista käyttää erilaisten aihekokonaisuuksien tilaamiseen.



Kuva 4.6: Publish-subscribe -malli



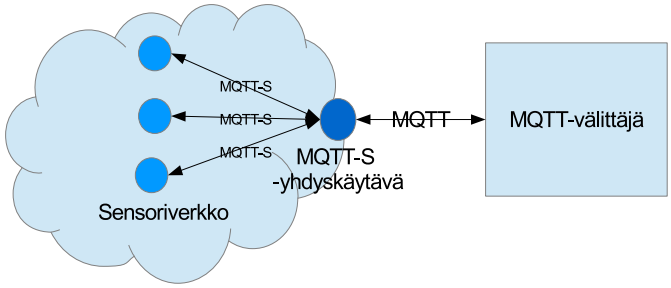
Kuva 4.7: MQTT:n käyttämä arkkitehtuuri

MQTT tukee kolmea palvelulaadun tasoa[5, s. 52]: viesti toimitetaan korkeintaan kerran eikä odoteta kuittausta, viesti toimitetaan vähintään kerran ja odotetaan kuittausta tai viesti toimitetaan vain kerran käyttäen nelivaiheista varmistusta.

Eri palvelulaatua voidaan käyttää saman sovelluksen sisällä. Esimerkiksi normaali mittaustieto voidaan toimittaa alhaisimmalla palvelutasolla, mutta raja-arvoista poikkeava ja hälytyksen aiheuttava tieto voidaan toimittaa korotetulla palvelutasolla

MQTT-S suunniteltiin MQTT:n laajennukseksi langattomia sensoriverkkoja varten ja sen on tarkoitus olla mahdollisimman samankaltainen MQTT:n kanssa, jotta sensoriverkot voitaisiin helposti yhdistää olemassa olevaan järjestelmään. Hunkeler ja Stanford-Clark esittelevät MQTT-S -protokollan artikkelissaan[24].

MQTT-S on tarkoitettu käytettäväksi sensoriverkon sisällä MQTT-S -yhdyskäytävälle asti. Tästä eteenpäin liikenne voidaan hoitaa MQTT:n avulla tai MQTT-S -yhdyskäytävä voi olla suorana yhteydessä välittäjään. Kuvassa 4.8 näkyy miten MQTT-S yhdistyy MQTT:n avulla MQTT-välittäjään. Yhdyskäytävän tehtävä on siis tulkata MQTT:n ja MQTT-S:n väliset viestit.

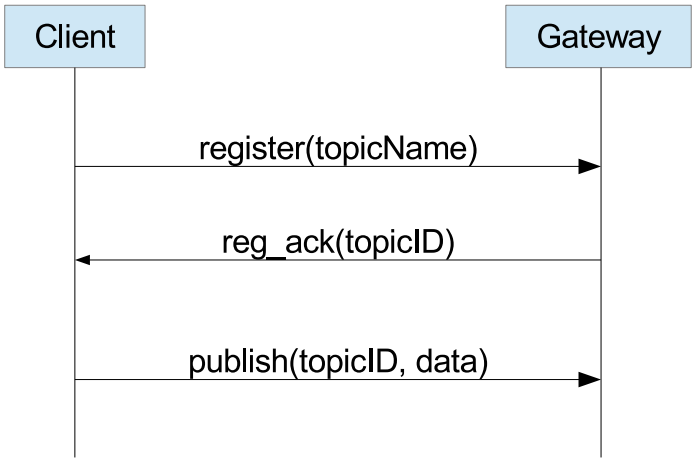


Kuva 4.8: MQTT-S -arkkitehtuuri

Yhdyskäytävä voi olla luonteeltaan joko läpinäkyvä (transparent) tai keräävä (aggregate). Läpinäkyvä yhdyskäytävä ylläpitää jokaista MQTT-S -asiakasta varten yhteyden MQTT -välittäjään. Tällöin voidaan eritellä jokainen asiakas toisistaan ja tarjota välittäjäpalvelimen koko toiminnallisuus niille. Keräävä yhdyskäytävä käyttää vain yhtä yhteyttä välittäjään ja päättää mikä osa verkon laitteilta tulevasta informaatiosta siirretään välittäjälle.

MQTT-S tukee myös useampaa yhdyskäytävää. Tällä pyritään varmistamaan laitteiden jatkuva pääsy toimittamaan tietonsa välittäjälle. Jos yhdyskäytävä katoaa verkosta, laite etsii uuden yhdyskäytävän. Samalla voidaan myös varmistaa laitteen yhteys oikeaan verkkoon, jos alkuperäinen yhdyskäytävä on suuren liikennemäärän takia tukossa.

MQTT-S -protokollassa on pyritty myös pienentämään viestien kokoa. Kaksi viestityyppiä, CONNECT ja PUBLISH, voivat olla hyötykuormasta riippuen hyvinkin isoja. CONNECT -viestit on jaettu kolmeen eri osaan [24]. PUBLISH -viesteissä päätettiin käyttää aiheen nimen sijaan kaksitavuista topic id -tunnistetta. Asiakkaat rekisteröivät aiheet yhdyskäytävälle jolta ne saavat yksilöllisen tunnisteen, jota käyttävät jatkossa julkaistessaan tietoa. Rekisteröintiprosessi näkyy kuvassa 4.9[24]



Kuva 4.9: MQTT-S - aiheen rekisteröinti

4.2 Semanttiset ratkaisut

Tässä alaluvussa käydään läpi eri tapoja esittää mittauslaitteiden tarjoama tieto ja miten siihen tietoon päästään käsiksi. Tarkemmin perehdytään erilaisiin tapoihin esittää itse tieto sekä tähän liittyvä metatieto. Tätä varten käydään lyhyesti läpi OGC:n SensorThings API [35] ja W3C:n Web of Things -arkkitehtuuri[27]. TTY:n kehittämä WSN OpenAPI käydään tarkemmin läpi kappaleessa viisi.

4.2.1 Tiedon muoto ja metatieto

OGC SensorThings API [35] koostuu kahdesta osasta, mittauksista ja toiminnasta. Mittausosa mahdollistaa tiedolle ja metatiedolle CRUD-operaatiot. Tietomalli koos-

Taulukko 4.1: Datastream-olion ominaisuudet

Nimi	Määritelmä	Tiedon tyyppi	Käyttö ja yhteydet
name	Kuvaileva nimi Datastream -entiteetille	CharacterString	Yksi, pakollinen
description	Kuvaus	CharacterString	Yksi, pakollinen
unitOfMeasurement	Kolme avain-arvo -paria, nimi, symboli ja määritelmä.	JSON-olio	Yksi, pakollinen
observationType	Havainnon tyyppi	OGC 10-004r3 ja ISO 19156:2011 -määritelmien mukaiset mittaustyypit	Yksi, pakollinen
observedArea	Rajattu alue, johon Datastream-olion mittaukset kuuluvat	GeoJSON-monikulmio	Nollasta yhteen
phenomenonTime		Aikaväli, ISO 8601 -standardin mukaan	Nollasta yhteen
resultTime		Aikaväli, ISO 8601 -standardin mukaan	Nollasta yhteen

tuu havainnoista (Observation), jotka tuottavat tuloksia (Result), jotka ovat arvioita mittauksen kohteen (FeatureOfInterest) jostakin ominaisuudesta. Havainnot luokitellaan ajan, kohteen, mittaustoimenpiteen (esimerkiksi Sensor) ja havainnoidun ominaisuuden (ObservedProperty) mukaan. SensorThings määrittelee myös Thing-tyyppisen entiteetin, joka on samalla tavalla määritelty kuin W3C:n Web of Things -määritelmässä. Thing-entiteettien sijaintitietoa pidetään olennaisena ja näiden sijaintihistoria tallennetaan. Havainnoitujen ominaisuuksien ja anturin kokoelmasta käytetään termiä Datastream. Datastream -olion sisältö on koottu taulukkoon 4.1.

Tiedon esityksen määritelmä on saatavilla OGC:n standardissa SWE Common Data Model[47]. SWE Common Data Model -määritelmän tarkoituksena on tarjota tapa kuvata sensoreihin liittyviä tietojoukkoja. Määritelmän mukaan tietojoukkoon kuuluu sensorien mittaukset ja sensoreihin liittyvä tieto, kuten yksittäisen anturin tila. Tietojoukko koostuu tietokomponenteista, joita ovat:

- Esitys
- Tiedon luonne ja semantiikka
- Laatu
- Rakenne
- Enkoodaus

W3C:n Web of Things -aloite [27] pyrkii standardoimaan ja yhdistämään IoT -laitteet ja palvelut. Päätavoitteena on kuvata IoT-rajapinnat formaalisti, jotta eri laitteet ja palvelut voivat kommunikoida keskenään. WoT -aloite tarjoaa tämän lisäksi myös mekanismin laitteiden löytämistä ja käyttöä varten.

WoT:n arkkitehtuuri rakentuu seuraavista elementeistä: Thing, WoT Thing Description, WoT Binding Templates ja WoT Scripting API. Näitä elementtejä voidaan käyttää WoT-arkkitehtuurissa laite-, yhdyskäytävä- ja pilvitasolla.

Thing on fyysisen tai virtuaalisen asian esitys. Tämä asia voi olla mittalaite tai vaikka sijainti. Jokainen Thing tarjoaa rajapinnan vuorovaikutusta varten. Tämä rajapinta (WoT Interface) on formaalisti määritelty.

Jokaisella Thing-entiteetillä pitää myös olla Thing Description (TD). Thing Description on määritelty formaalisti[26] ja sen on tarkoitus toimia "esineiden HTML:nä". TD tarjoaa Thing-entiteetin yleisen metatiedon sekä metatiedot sen vuorovaikutuksista, tietomallista, viestinnästä ja turvallisuusmekanismeista. Thing Description sisältää siis esineen ominaisuudet, toiminnot ja yksilöivät osoitteet mistä näihin pääsee käsiksi.

Internet of Things -teknologian yksi haasteista on suuri laite- ja protokollakirjo. WoT Binding Template on W3C:n yritys selventää tilannetta tarjoamalla Thing Descriptionin yhteydessä metatietoa Thing-entiteetin käyttämistä viestintätavoista. WoT Binding Templates on kokoelma kuvauksia eri laitteiden tavoista olla vuorovaikutuksessa keskenään. Binding Template (kytkentämalli) otetaan käyttöön kun ollaan luomassa Thing Descriptionia. Jokaisella IoT-alustalla on vain yksi kytkentämalli ja tätä käytetään jokaisen alustan laitteen kuvauksessa. Kytkentämallissa oleva metatieto kattaa neljä eri osiota, joita ovat IoT -alusta, tiedonsiirtoprotokolla, mediatyyppi, turvallisuus.

Thing descriptionin ja Binding Templaten avulla muodostuu rajapinta esineen ominaisuuksiin ja toimintoihin. Valittu protokolla määrittelee käytettävät metodit ja Thing Description kertoo mistä yksilöllisistä osoitteista ominaisuudet ja toiminnot ovat käytettävissä sekä missä formaatissa mahdolliset muuttujat ovat.

Web of Things tarjoaa myös Scripting API:n[30], jonka tarkoitus on helpottaa IoT -sovellus-kehitystä. Se helpottaa WoT -verkossa olevien laitteiden ja esineiden löytämistä ja niiden käyttämistä. Scripting API jakaa tämän toiminnallisuuden kolmen eri osion kautta. Peruselementtinä on WoT -olio. Tämä olio tarjoaa metodit Thing-entiteettien löytämistä, käyttämistä ja ominaisuuksien paljastamista varten. ConsumedThing -rajapinnan kautta päästään käsiksi Thing-entiteetin ominaisuuksiin ja toimintoihin sekä voidaan seurata sen ominaisuuksia ja tapahtumia. Kolmantena on ExposedThing -rajapinta. Tämän kautta määritellään Thing-entiteetin pyyntöjen käsittelijät, toiminnot ja tapahtumat. Tämä rajapinta toteuttaa ConsumedThing -rajapinnan. W3C Scripting API -työryhmä ehdottaa myös Observable -luokan rajapintojen käyttöönottoa. Näytä ovat Observer, Subscription ja Observable.

4.2.2 Tiedostoformaatit

Yksinkertaisin tiedostoformaatti on CSV (comma-separated values), joka sisältää tiedot taulukkona tekstimuodossa. Yksi rivi sisältää yhden tietueen. Rivin sisällä eri kentät erotetaan useimmiten pilkulla. CSV -tiedostomuotoa ei kuitenkaan ole missään vaiheessa formaalisti määritelty, mutta IETF:n julkaisussa [49] käyttämän määritelmän mukaan yleisimmältä toteutukselta vaikuttaa seuraava:

1. Jokainen tietue on omalla rivillään. Rivin loppu ilmaistaan rivinvaihtomerkillä.

2. Tiedoston viimeisellä rivillä ei välttämättä ole rivinvaihtoa

3. Tiedoston ensimmäinen rivi voi olla otsikkorivi. Otsikkorivi noudattaa samaa muotoilua kuin muut rivit. Otsikkorivin kentät nimeävät muiden rivien vastaavat kentät.

4. Jokaisella tietuerivillä voi olla yksi tai useampi kenttä, jotka erotellaan toisistaan pilkulla. Joka rivillä tulee olla sama määrä kenttiä. Välilyönnit lasketaan myös kenttien sisällöksi. Viimeisen kentän perään ei tule pilkkua.

5. Jokaisen kentän alussa ja lopussa voi olla lainausmerkit. Jos lainausmerkkejä ei käytetä rajaamaan tietoa, kenttien sisällä ei voi käyttää lainausmerkkejä.

6. Rivinvaihtoja, lainausmerkkejä tai pilkkuja sisältävät kentät tulisi rajata lainausmerkeillä.

7. Jos kenttä rajataan lainausmerkeillä, kentän sisällä oleva lainausmerkki pitää ilmaista käyttämällä kahta lainausmerkkiä.

Ongelma CSV:n käytössä langattomien sensoriverkkojen tiedonsiirrossa on juuri standardoinnin puute. CSV -tiedostojen käsittelyä ja lukemista varten on kuitenkin olemassa useita valmiita työkaluja eri ohjelmointikielille.

XML eli Extensible Markup Language on formaalisti määritelty[12] kieli, joka on SGML:n (Standard Generalized Markup Language) rajoitettu muoto. Jokainen XML-dokumentti omaa loogisen ja fyysisen rakenteen. Fyysisesti XML-dokumentti koostuu yksiköistä ja dokumentti alkaa juuriyksiköstä. Loogisesti dokumentti koostuu esittelyistä, elementeistä, merkkiviittauksista ja käsittelyohjeista. Jokainen XML-elementti määritetään alku- ja loppumerkeillä ja jokaisella elementillä on määritelty tyyppi. Elementti voi olla myös tyhjä. Yksinkertainen XML-dokumentti voi olla kuten esimerkissä 4.1.

Listing 4.1: Esimerkki XML-dokumentista

```
<esimerkkiviesti>
  <vastaanottaja>Tuomas</vastaanottaja>
  <otsikko>Hei!</otsikko>
  <päivämäärä>14.5.2018</päivämäärä>
</esimerkkiviesti>
```

Javascript Object Notation[11] on kehitetty Javascriptin olioiden kuvauksesta. JSON -tarjoaa neljä muuttujatyyppiä ja kaksi rakennetyyppiä. Muuttujia ovat merkkijonot, numerot, totuusarvot ja tyhjä olio. Rakennetta varten on käytössä taulukot ja oliot. Olio on järjestämätön kokoelma nimi/arvo -pareja ja taulukko on järjestetty kokoelma arvoja. Esimerkki yksinkertaisesta JSON-oliosta löytyy listauksesta 4.2.

Listing 4.2: Esimerkki JSON-dokumentista

```
{
    "vastaanottaja": "Tuomas",
    "otsikko": "Hei!",
    "päivämäärä": "14.5.2018"
}
```

4.3 Tiedon varastointi

Tiedon varastointiin käytetään yleensä jonkinlaista tietokantaa. Vaihtoehtoina ovat tavallisimmin perinteiset relaatiotietokannat ja niin sanotut NoSQL -tietokannat. Tässä alaluvussa vertaillaan lyhyesti näitä kahta eri tietokantatyyppiä langattomien sensoriverkkojen näkökulmasta. Tässä työssä puhuttaessa relaatiotietokannoista tarkoitetaan SQL eli Structured Query Language-kyselykieleen perustuvia tietokantoja. SQL on peräisin 1970-luvulta ja on ANSI:n toimesta standardoitu[31]. Relaatiotietokannat perustuvat ajatukseen tiedon tallentamisesta omiin tauluihinsa, joista ne on haettavissa itse taulun, taulun avaintiedon tai taulun kolumnin perusteella. Relaatiotietokannat lupaavat vahvan tuen niin sanotulle ACID -periaatteelle (atomicity, consistency, isolation, durability). Tällä tarkoitetaan sitä, että tiedon eheys ja pysyvyys taataan kun tietokannalle suoritetaan transaktioita.

NoSQL ("Not Only SQL"tai "Not Relational") tarkoittaa Rick Cattellin mukaan[13] tietokantaa, joka on horisontaalisesti skaalattavissa, voi hajauttaa tietonsa usealle eri palvelimelle, tarjoaa yksinkertaisen rajapinnan tai protokollan kutsuja varten, on samanaikaisuusmalliltaan heikompi kuin perinteiset relaatiokannat, hyödyntää

hajautettuja indeksejä tai muistia tiedon varastointiin ja pystyy dynaamisesti lisäämään uusia ominaisuuksia tietuiseiinsa. Toisin kuin relaatiotietokannat, NoSQL -tietokannat eivät tue ACID -periaatetta, vaan tarjoavat niin sanotun BASE -takuun. Tällä tarkoitetaan, että tieto on yleisesti saatavilla (basically available), pehmeässä tilassa (soft state) ja ennen pitkää johdonmukainen (eventually consistent).

4.3.1 Tietokantaratkaisujen vertailu

Artikkelissaan[13] Cattell vertaa relaatiotietokantoja ja NoSQL-tietokantoja skaalautuvuuden suhteen. Hän luokittelee tietokannat vielä tarkemmin avain-arvo -tietokantoihin kuten Redis, dokumenttitietokantoihin kuten MongoDB, extensible record -tietokantoihin kuten Googlen BigTable ja relaatiotietokantoihin, joista esimerkkinä on nykyisin skaalautuva MySQL. Skaalautuvuutta käsitellään samanaikaisuuden, tiedon varastoinnin, replikoinnin ja suoritettujen operaatioiden kautta. Tuloksien perusteella näyttää siltä, että skaalautuvuus on hyvin tuettu ominaisuus useiden perinteisten relaatiotietokantojen ja NoSQL -kantojen osalta.

Suorituskykyä on vertailtu Lin ja Manoharanin artikkelissa[34]. NoSQL -kannoista vertailussa mukana olivat MongoDB, RavenDB, CouchDB, Cassandra, Hypertable, Couchbase ja perinteisistä relaatiotietokannoista Microsoftin SQL Express. Kantojen suorituskykyä vertailtiin viidellä eri operaatiolla, tietokannan luonti, tietojen luku, tietojen kirjoitus, tietojen tuhoaminen ja kaikkien avainten haku. Tietona käytettiin avain-arvo -tietueita. Vaikka NoSQL -tietokannat ovat yleensä hyvin optimoitu tällaisten tietueiden käsittelyä varten, ne eivät silti aina suoriudu paremmin kuin SQL -tietokannat. CouchBase ja MongoDB ovat kuitenkin vertailluista nopeimmat luku-, kirjoitus- ja poisto-operaatioissa.

Näiden kahden vertailun perusteella näyttää siltä, että tietokantaratkaisu pitää tehdä järjestelmän käyttötapaukset ja vaadittu toiminnallisuus mielessä pitäen. Relaatiotietokannat ovat testattuja ja vankkoja järjestelmiä, jotka tukevat samanaikaisia operaatioita paremmin kuin NoSQL -tietokannat. Daniel Bartholomew vertailee artikkelissa[7] lyhyesti eri tietokantaratkaisuja ja hänen mukaansa kyse on siitä, että tarjolla on vaihtoehtoja, joista pitää osata valita työn alla olevaan järjestelmään sen haasteisiin sopiva.

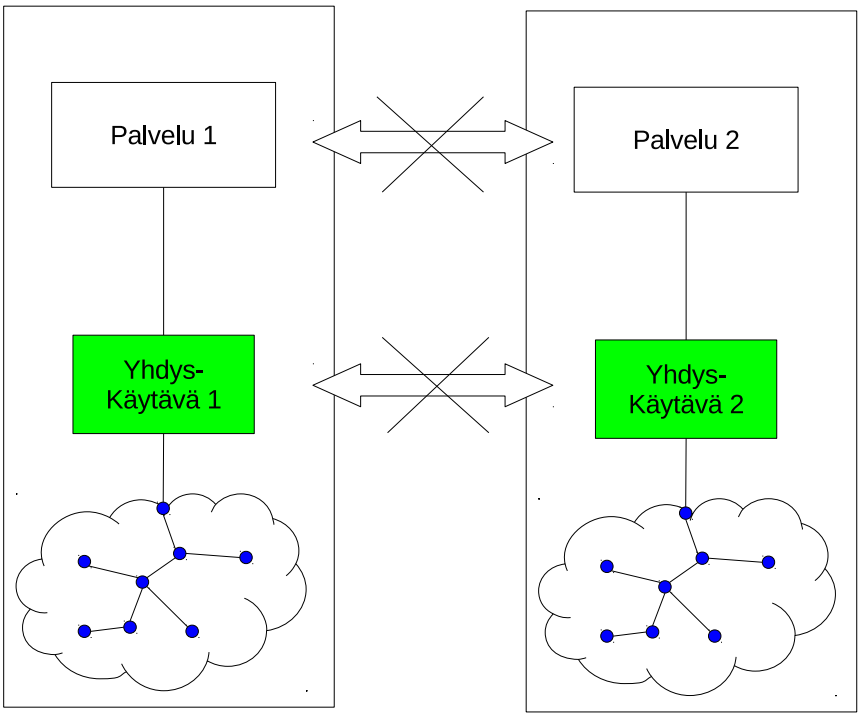
4.4 Yhteentoimivuus

Aiemmissa alaluvuissa esitettyjen ratkaisujen avulla esineiden internet on helpompi toteuttaa. Nämä teknologiat mahdollistavat siis eri järjestelmien ja laitteiden tulee

keskinäisen kommunikoinnin. Atzori, Iera ja Morabito jakavatkin artikkelisssaan[4] esineiden internetin kolmeen eri näkemykseen: itse esineet, internet ja semantiikka. Esinenäkökulmalla tarkoitetaan laitteita ja niiden verkkotason tiedonsiirtoa. Internet-näkökulma keskittyy laitteiden yhdistämiseen ja tunnistamiseen. Semanttinen näkökulma tarkoittaa sitä, että esineiden internetin varastoiman tiedon esitys pitää myös ottaa huomioon.

Desai, Sheth ja Anantharam esittävät[17], että Internet of Things -teknologian yksi iso haaste on yhteentoimivuus. Artikkelissa yhteentoimivuusongelma jaetaan kolmeen eri luokkaan: verkkokerroksen yhteentoimivuus, viestiprotokollien yhteentoimivuus ja mittaustiedon metadatan merkitsemistä vaikeuttavat standardien puutteet.

Koska Internet of Things -sovellukset usein ovat alhaalta ylös -periaattella rakennettuja, Desain et al mukaan[17] niitä voidaan ajatella pystysuorina "siiloina"joiden välillä ei ole horisontaalista kommunikaatiota. Tällöin siis yhdyskäytävälaitteet ja palvelut eivät kommunikoi keskenään, kuten kuvassa 4.10 näkyy. Erilaisten sensoriverkkojen yhteentoimivuus on kuitenkin ratkaistu yleensä yhdyskäytävätasolla, jolloin palveluiden yhteentoimivuus on helpommin toteutettavissa.



Kuva 4.10: IoT-palveluiden siilot

Claire Rowland käsittelee kirjassa[48, luku 10] yhteentoimivuutta tieto- ja verkkotasolla. Tällä tarkoitetaan sitä, että IoT -laitteiden ja palveluiden tulisi pystyä löytämään toisia laitteita ja palveluita sekä kommunikoimaan toistensa kanssa. Laitteet pitäisi myös pystyä vaihtamaan toisen valmistajan laitteisiin ilman häiriöitä järjestelmän toiminnassa. Eri valmistajien laitteiden ja järjestelmien tulisi pystyä kommunikoimaan keskenään riippumatta siitä mikä laitekanta on valittu tai mitä verkko- tai sovellusprotokollaa käytetään.

Rowland esittää[48, s. 394] yhtenä esimerkkinä laitteiden yhteentoimivuudelle Qualcommin AllJoyn -projektin. Se mahdollistaa eri laitteiden keskinäisen kommunikaation, kunhan kaikki laitteet tukevat AllJoyn -järjestelmää. Toisena vaihtoehtona tarjotaan yhdyskäytävän käytön eri laitteiden välillä. Myös eri yhdyskäytävät voisivat välittää tietoa keskenään.

Internet of Things -yhteydessä verkkokerroksen yhteentoimivuusongelmalla tarkoitetaan sitä, että käytössä on useita eri protokollia kuten ZigBee, Wi-Fi ja LoRaWAN. Viestiprotokollien yhteentoimivuusongelma johtuu siitä, että tarjolla on useita eri sovellustason protokollia, kuten HTTP, CoAP, MQTT ja XMPP (Extensible Messaging and Presence Protocol). Desain, Shethin ja Anantharamin mukaan[17] skaalautuvan IoT -arkkitehtuurin pitäisi kuitenkin olla riippumaton eri viestiprotokollista ja samalla kuitenkin tarjota yhteentoimivuutta ja käännökset näiden välillä.

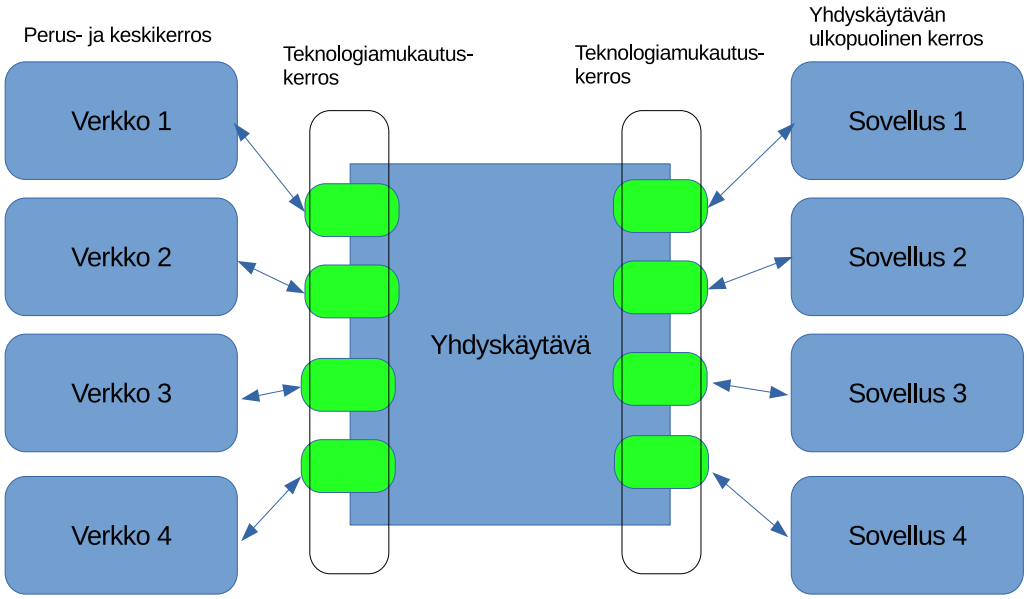
Desainin et al[17] tarjoavat protokollatason ongelmien ratkaisuksi yhdyskäytävää, joka mahdollistaa eri viestiprotokollien käytön yhtäaikaisesti. Heidän mukaansa myös verkkotason yhteentoimivuusongelma voidaan ratkaista käyttämällä montaa eri yhteysprotokollaa tukevia laitteita tai siirtämällä ongelman käsittely sovellustasolle. Jäljelle jää siis tiedon siirto eri laitteiden ja järjestelmien välillä.

Azori et al[4] korostavat artikkelissaan myös esineiden internetin semanttista näkökulmaa. Käsitteen esineiden internet takia painotus vaikuttaa olevan laitteiden ja niiden kommunikoinnissa. Heidän mukaansa yhdistettyjen laitteiden määrän kasvaessa, myös esineiden internetin tuottaman tiedon esittäminen, varastointi, yhdistäminen ja organisointi nousevat haasteiksi.

5 WSN OpenAPI

Yhteentoimivuuden varmistamista edesauttaa selkeästi määritelty tietojen esitystapa. Tampereen teknillisen yliopiston kehittämä WSN OpenAPI[55] tarjoaa oman tapansa käsitellä sensoriverkon mittaustietoa. Artikkelissaan [29] Karvonen et al esittelevät WSN OpenAPI:n arkkitehtuurin. WSN OpenAPI toimii abstraktiokerroksena eri sensoriverkoille ja tarjoaa pääsyn verkon palveluihin määriteltyjen rajapintojen kautta.

WSN OpenAPI:n arkkitehtuuri perustuu hierarkiaan, jossa alimmaisena ovat yksinkertaiset elementit, kuten anturit. Tätä kutsutaan peruskerrokseksi (elementary layer). Ylemmillä tasoilla olevat elementit ovat monimutkaisempia, kalliimpia ja tehokkaampia, mutta kuluttavat myös enemmän virtaa. Peruskerroksen yläpuolella ovat keskikerros (intermediate layer), kehittynyt kerros (advanced layer) ja yhdyskäytävän ulkopuolinen kerros (outer layer).

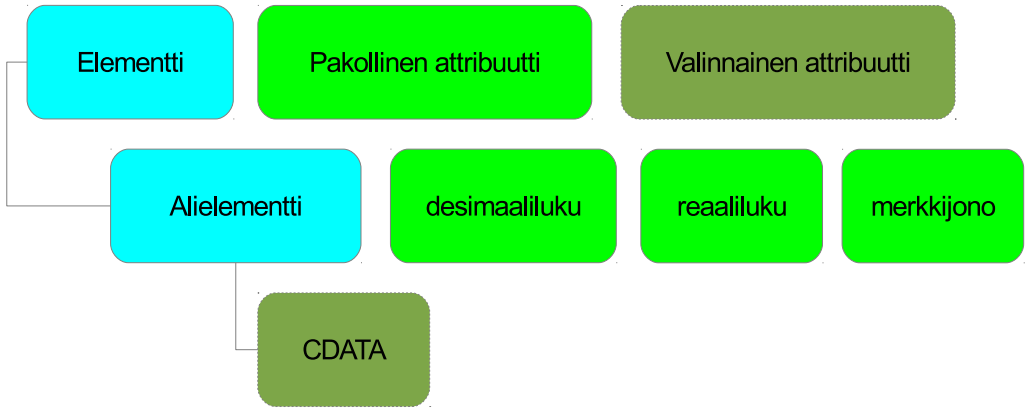


Kuva 5.1: WSN OpenAPI - yhdyskäytäväarkkitehtuuri

Yhdyskäytävä mahdollistaa useiden eri valmistajien laitteiden ja eri teknologioiden yhteiskäytön. Kuvassa 5.1 näkyy WSN OpenAPI:n yhdyskäytäväarkkitehtuuri. Yhdyskäytävän tarkoitus on mahdollistaa eri valmistajien viestien tulkkaamisen yh-

teen muotoon. Ulkopuoliset asiakasohjelmat ja -laitteet voivat täten käyttää eri verkkojen tarjoamaa tietoa. TA (technology adaptation) tarkoittaa teknologiamukautuksesta vastaavia järjestelmän osia.

WSN OpenAPI määrittelee rajapinnat ja tietoformaatit [54], joita hyödynnetään anturien ja toimilaitteiden kanssa. Kuvassa 5.2 näkyy käytettyjen viestien yleinen formaatti.



Kuva 5.2: Viestin rakenne

Elementillä tarkoitetaan attribuuttien ja alielementtien kokoelmaa. Elementillä voi olla monta alielementtiä, esimerkiksi 0..1 eli korkeintaan yksi alielementti tai 0..* jolloin alielementtejä voi olla kuinka monta tahansa. Elementin attribuutit ovat nimen ja arvon muodostamia pareja. Arvo voi olla desimaaliluku (d), reaaliluku (f) tai merkkijono (s). Jos arvon tyyppiä ei ole erikseen ilmoitettu, oletetaan käytettävän merkkijonoa. CDATA viittaa elementin arvoon. Tällä syntaksilla kuvataan WSN Open API:n käyttämät viestit ottamatta sen kummemmin kantaa käytettyyn tiedostomuotoon tai enkoodaukseen.

Rajapintoja ovat seuraavat:

- Authentication and Capability Format (ACF)
- Meta-Data Format (MEDF)
- Network Management Format (NMF)
- Node Actuator and Sensor Control (NASC]

- Sensor Archive Data Format (SADF)
- Sensor Information Data Format (SIDF]

ACF tarjoaa autentikoinnin ja listan tarjotuista rajapinnoista. MEDF tarjoaa rajapinnan solmujen ominaisuuksien ja anturien listausta varten. NEMF tarjoaa verkon rakenteen, tilan ja mittausten keräyksen konfiguroinnin. NASC -rajapinta mahdollistaa toimilaitteiden ohjauksen. SADF määrittelee kuinka sensoriverkon tietoarkistoon tallennettuun tietoon päästään käsiksi. SIDF määrittelee verkon reaaliaikaisen mittaustiedon hyödyntämiseen tarvittavat metodit ja tietomuodot.

WSN OpenAPI sisältää siis useita eri rajapintoja, joita voidaan käyttää erikseen. Asiakasohjelman osalta vain ACF eli autentikoinnin ja rajapintalistakyselyn toteuttaminen on pakollista[55].

WSN OpenAPI käyttää verkon solmujen, anturien ja toimilaitteiden esittämiseen hierarkkista rakennetta. Yhteen verkkoon kuuluu solmuja, joihin kuuluu yksi tai useampi anturi tai toimilaite. Tämän lisäksi anturilla voi olla komponentteja. Esimerkiksi kiihtyvyysmittarilla on jokaista akselia kohti komponentti.

Rajapinnat määrittelevät operaatiot pyyntö-vastaus-pareina. Viesteissä on pakollisina attribuutteina versionumero, joka takaa taaksepäin yhteensopivuuden eri määrittelyversioiden välillä. Vastausviestin tulisi käyttä samaa versiota kuin pyyntö. Valinnaisena attribuuttina käytetään viestin tunnistetta messageId. Tämä mahdollistaa vastauksen yhdistämisen aikaisemmin lähetettyyn pyyntöön. Vastausviestissä on tämän lisäksi pakollisena attribuuttina vastauskoodi responseCode, joka kertoo onnistuiko operaatio vai ei. Vastauskoodit ovat HTTP:n vastauskoodien alijoukko. Vaikka WSN OpenAPI käyttää HTTP:n vastauskoodeja, se ei kuitenkaan ota kantaa käytettyyn tiedonsiirtoprotokollaan. Sen käyttämät pyyntö-vastaus-operaatiot soveltuvat lyhytkestoisia yhteyksiä käyttäviin järjestelmiin.

Tiedostomuotoina voidaan käyttää mitä tahansa tekstipohjaista muotoa helpottamaan eri laitearkkitehtuurien välistä kommunikointia. WSN OpenAPI määrittelee kaksi eri tiedostomuotoa, XML ja CSV.

WSN OpenAPI jättää joitakin implementaation yksityiskohtia asiakkaan ja palvelun sovittavaksi. Näihin kuuluvat muun muassa palvelun osoite, portti, käytetty protokolla ja tuetut WSN OpenAPI -rajapinnat. Tuetut rajapinnat voidaan määritellä myös ACF:n kautta.

WSN OpenAPI -määrittelyn mukaan sovellusten pitää tukea yhtä tai useampaa seuraavista autentikointimenetelmistä:

- none: ei mitään

- password: tunnuksella ja salasanalla tapahtuva autentikointi
- challenge: challenge-response -menetelmä

ACF:n mukaisesti järjestelmän pitää pyydettäessä tarjota lista käytettävissä olevista autentikointitavoista. Tätä varten ei edellytetä yhteyden autentikointia. Käytettävät viestit ovat AuthenticationListRequest ja AuthenticationListResponse.

Kun tarjolla olevat autentikointimenetelmät ovat selvillä, autentikointi tapahtuu lähettämällä järjestelmälle AuthenticationRequest -viesti. WSN OpenAPI olettaa, että yhteys on tässä vaiheessa suojattu jotenkin, esimerkiksi Transport Layer Securityn avulla. Vastauksena järjestelmä lähettää AuthenticationResponse -viestin, joka sisältää onnistuneen autentikoinnin jälkeen koodin 200 ja koodin 405, jos autentikointi epäonnistui.

Järjestelmän tukemat rajapinnat ja ominaisuudet selvitetään lähettämällä sille CapabalityRequest. Palvelin lähettää takaisin CapabilityResponse -viestin, jossa on lista tarjotuista rajapinnoista ja niihin liittyvistä parametreistä.

5.1 SIDF

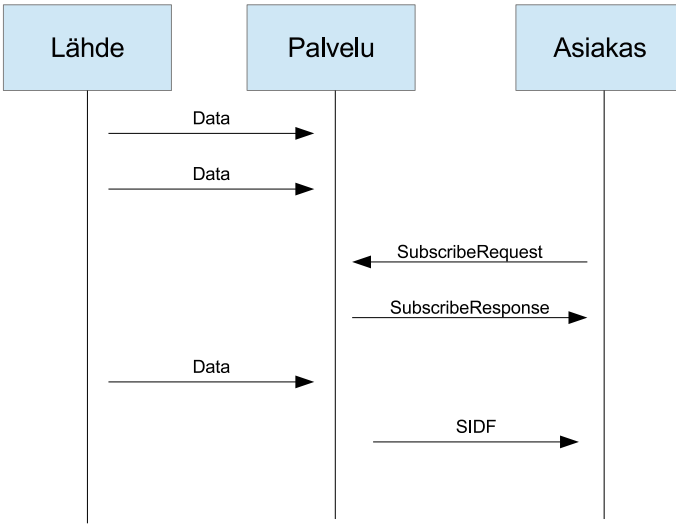
SIDF on tarkoitettu reaaliaikaisen tiedon tarjoamiseen. Asiakkaat tilaavat haluamansa tiedon järjestelmältä ja se toimitetaan heille kun sitä tulee tarjolle. SIDF käyttää siis publish-subscribe -mallia. Kuvassa 5.3 näkyy tämän mallin toiminta.

SubscribeRequest -viestillä pyydetään järjestelmältä tietyn mittauksen, tapahtuman tai kokonaisen verkon tietoa. Haluttu mittaustieto voidaan rajoittaa tiettyyn verkkoon, solmuun tai yksittäiseen anturiin. Tyhjä SubscribeRequest -viesti tilaa kaikki mittaukset. Jos asiakkaalla oli jo tilaus voimassa, uusi SubscribeRequest korvaa aiemman tilauksen.

SubscribeResponse lähetetään vastauksena tilauspyynnölle ja sisältää vastauskoodin. Onnistuneen tilauksen tapauksessa tämä koodi on 200. Jos tilaus epäonnistuu, mahdollinen aiempi tilaus pidetään voimassa.

Tilaus perutaan lähettämällä UnsubscribeRequest palvelimelle. Vastauksena tähän lähetetään UnsubscribeRequest. Vastausviesti sisältää virhekoodin. Tilauksen peruutuksen epäonnistuessa aikaisempi tilaus pidetään voimassa.

Kun tilaus on voimassa, SIDF-tieto lähetetään asiakkaalle Data-viesteinä. Tieto lähetetään kuvan 5.4 mukaisessa muodossa. Jos Data-viestin response-ominaisuuden arvona oli true, viestin vastaanotto kuitataan lähettämällä DataResponse-viesti takaisin. Mittaustietoa voidaan lähettää isompina kokonaisuuksina kerrallaan käyttäen niin sanottua block data -muodossa. Tieto esitetään tällöin CSV-muodossa. En-



Kuva 5.3: Reaaliaikainen mittaustiedon välitys

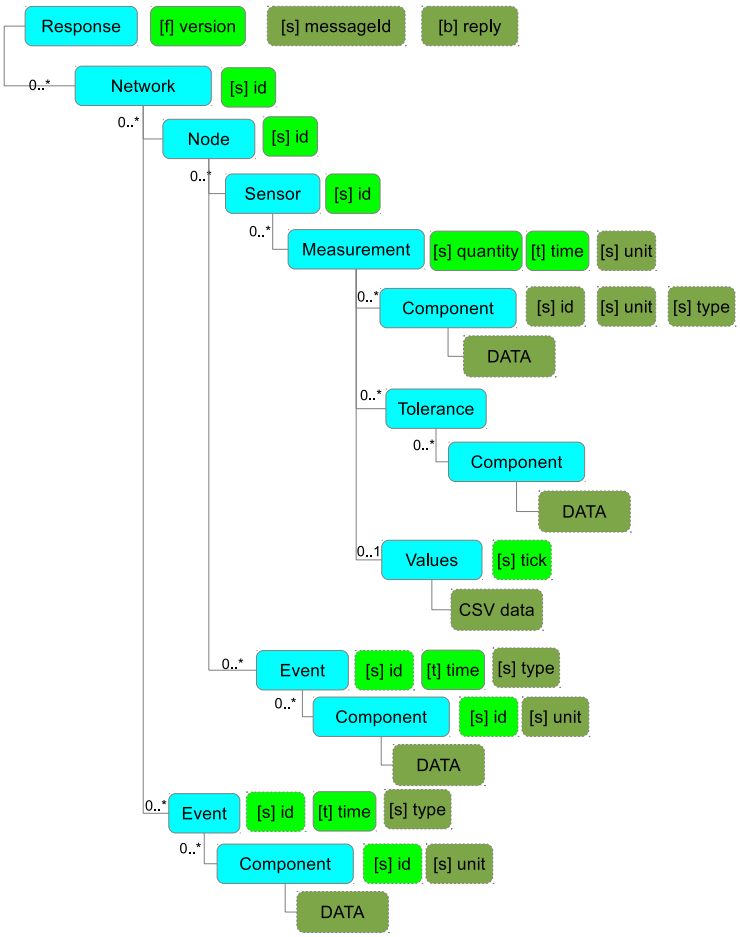
sin määritellään aikasykäysten tyyppi, sekunti tai vastaava. Tämän jälkeen tieto esitetään rivi kerrallaan. Jokaisen rivin ensimmäisenä tietueena on ero mittauksen aikaleimaan aikasykäyksinä.

SIDF -rajapinta sisältää seuraavat ACF:n mukaiset toiminnot:

- acceptPushData
- blockDataSupport
- extDataSupport
- extResourceSchemes

5.2 SADF

SADF -rajapinta tarjoaa sensorien mittaustietoarkiston määrittelyn. SADF toimii pyyntö-vastaus -mallin mukaan. Käytetty operaatio on nimeltään Query. Jos Query -viestillä pyydetään mittaustuloksia sellaisilta elementeiltä, johon asiakkaalla ei ole oikeuksia, ne ohitetaan ilman mitään virheilmoitusta eikä niihin liittyvää tietoa palauteta. Pyynnön yhteydessä voidaan määritellä myös halutaanko tietoa prosessoida jotenkin. Prosessointia varten pitää syöttää aikaväli miltä kerätty tieto käsitellään. Prosessointivaihtoehtoja ovat min (pienin arvo), max (suurin arvo), avg



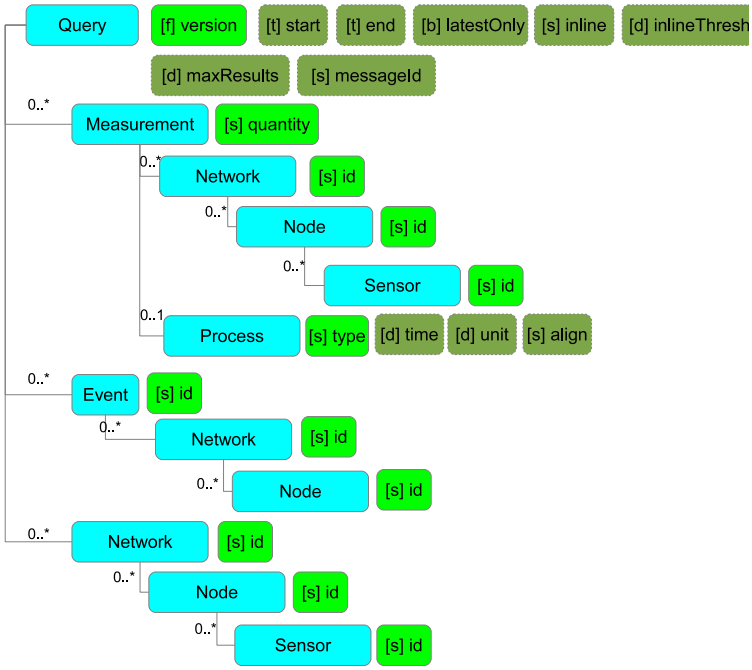
Kuva 5.4: SIDF-formaatti

(keskiarvo), sum (kaikkien arvojen summa) ja count (mittausten määrä aikavälillä). Kuvassa 5.5 näkyy pyyntöviestin hierarkkinen rakenne.

Vastausviesti sisältää vastauskoodin ja siihen liittyvän mittaustiedon. Käytettäviä koodeja ovat 200 onnistuneessa pyynnössä, 206 jos tulosten määrä ylittää pyynnön maxResults -arvon ja 501 jos pyynnön sisältämiä palveluita ei ole toteutettu. Kuvassa 5.6 näkyy vastausviestin hierarkkinen rakenne. Vastaus voi sisältää tulokset joko itsessään tai linkin ulkoiseen resurssiin.

SADF tarjoaa seuraavat ACF:n mukaiset parametrit:

- blockDataSupport
- extDataSupport
- extResourceSchemes



Kuva 5.5: Pyyntöviestin rakenne

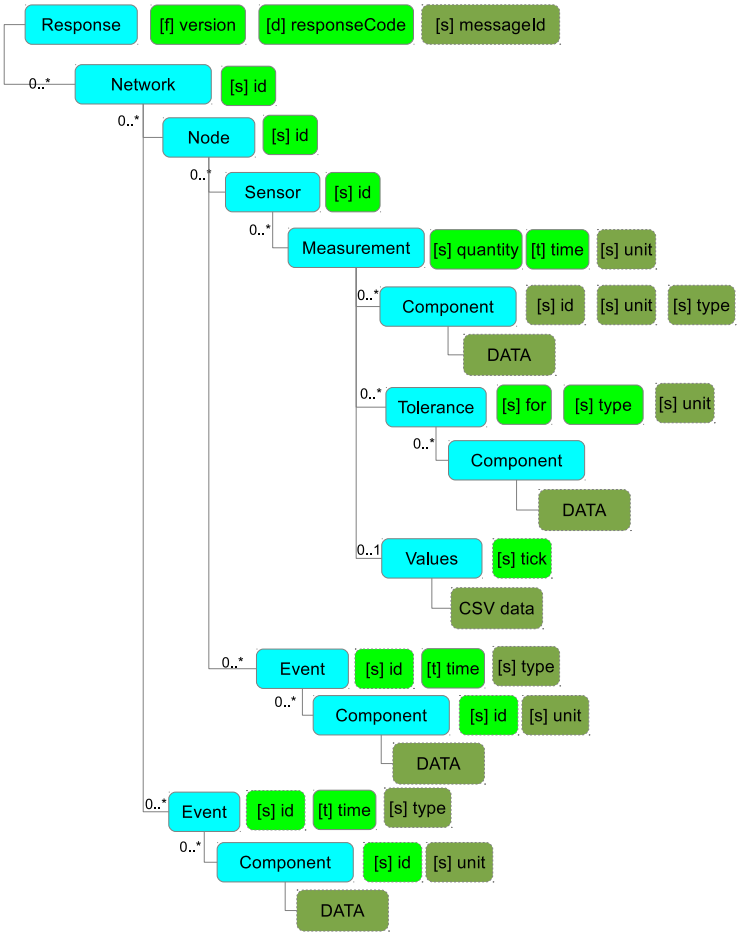
- extDataRetainTime

5.3 NASC

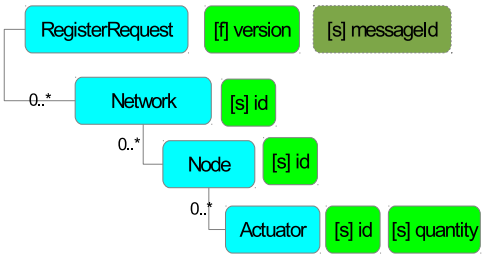
NASC -rajapinta on tarkoitettu toimijoiden kontrollointia varten. Erilaisia toimijoita voivat olla esimerkiksi valojen kytkimet. Rajapintaa ei kuitenkaan ole tarkoitettu vain fyysisten laitteiden ohjaamiseen, vaan sillä voidaan siirtää asetuksiin tai vastaavaan liittyvää tietoa laitteiden välillä.

Rekisteröintitoimintoihin kuuluvat viestit ovat RegisterRequest ja RegisterResponse. Viestien rakenteet näkyvät kuvissa 5.7 ja 5.8. Vastausviesti sisältää vastauskoodin 200 jos rekisteröinti onnistuu. Virhetapauksissa mahdollinen aikaisempi tilaus pysyy voimassa. Rekisteröinnin perumiseen kuuluvat viestit ovat UnregisterRequest ja UnregisterResponse, kuvissa 5.9 ja 5.10.

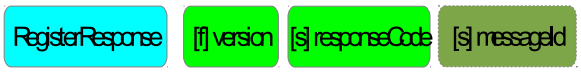
Toimilaitteen tilan voi selvittää lähettämällä StatusRequest -viestin. Jos Network -elementtiä ei pyydetä, vastauksena lähetetään jokaisen verkon jokaisen toimilaitteen tila, vastaavasti myös solmun ja toimilaitteen kohdalla. Vastauksena lähetetään StatusResponse -viesti. StatusRequest- ja StatusResponse -viestien rakenne näkyy kuvissa 5.11 ja 5.12.



Kuva 5.6: Vastausviestin rakenne



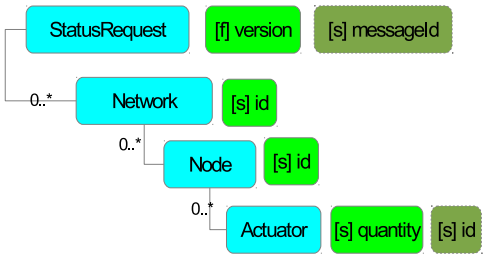
Kuva 5.7: NASC-tilaviestien tilaus



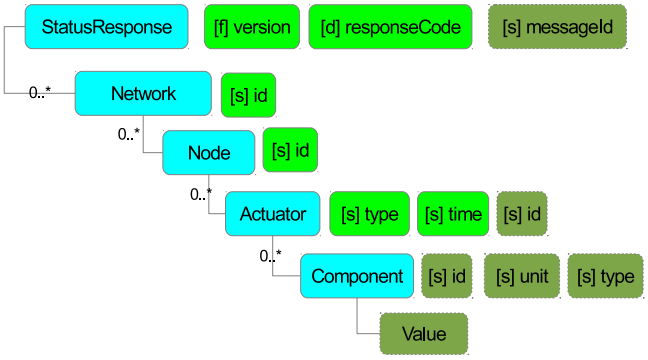
Kuva 5.8: NASC-tilaviestipyynnön vastaus

Kuva 5.9: NASC-tilauksen peruutus

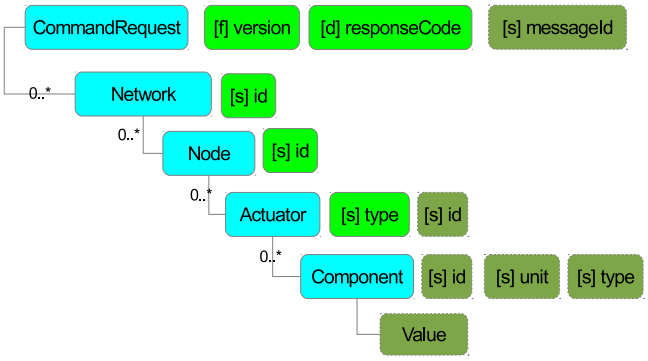
Kuva 5.10: NASC-tilauksen peruutuksen kuittaus



Kuva 5.11: Toimijan tilapyyntöviesti



Kuva 5.12: Toimijan tilapyyntöviestin vastaus

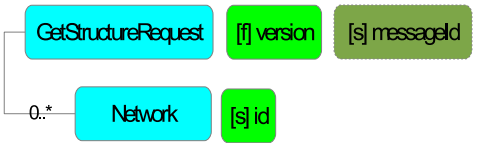


Kuva 5.13: Toimilaitteelle lähetetty komento

Toimilaitteen tilaa muokataan Command-operaatiolla. Se sisältää CommandRequest-, CommandResponse- ja StatusIndication -viestit. CommandRequest -viestin Actuator -elementin tiedolla rajataan lähetetty käsky vain tietylle toimilaitteelle. Jos sitä ei ole määritelty oletetaan, että käsky on tarkoitettu vastaanottavan solmun kaikille toimilaitteille. Viestin rakenne näkyy kuvassa 5.13.

CommandResponse -viesti lähetetään vastauksena CommandRequest -viestille. Viestin ResponseCode -elementti kertoo onnistuiko pyyntö. Koodilla 200 kerrotaan, että käsky suoritettiin onnistuneesti. Koodi 202 tarkoittaa, että käsky hyväksyttiin, mutta sitä ei vielä ole suoritettu. Palvelu vastaa asiakkaalle myöhemmin SIDF -muotoisella vastausviestillä kun toimilaitteen tila on muuttunut. NASC:n Actuator id- ja Actuator quantity -elementit vastaavat SIDF:n Sensor id- ja Sensor quantity -elementtejä. Tällä on haluttu välttää samanlaisen toiminnallisuuden toteutusta eri rajapinnoissa. Kuvassa 5.14 näkyy vastausviestin muoto.

Kuva 5.14: Toimilaitteelle lähetetty komennon vastaus



Kuva 5.15: Verkon rakenteen pyyntöviesti

5.4 NMF

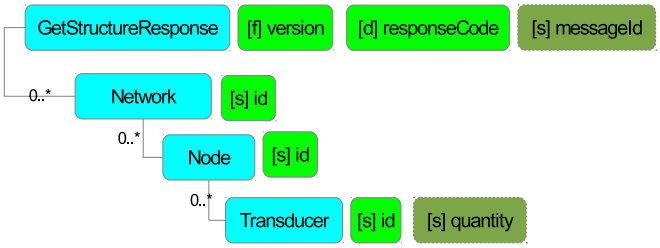
Network Management Format -rajapinta määrittelee millä metodeilla päästään käsiksi verkon rakenteeseen liittyvään tietoon ja millä metodeilla säädellään mittausdatan keräämistä verkosta. Käytettäviä operaatioita ovat GetStructure, GetMeasurements ja SetMeasurements.

GetStructure -operaatio palauttaa asiakkaalle listan järjestelmän halutuista verkoista, solmuista ja mittalaitteista. Jos pyynnössä ei ole erikseen määritelty verkkoa, palautetaan kaikkien niiden verkkojen rakenne, joihin asiakkaalla on pääsy. WSNOpenAPI ei ota kantaa siihen, millä tavalla verkon rakenne on järjestelmän tiedossa. Se voi olla myös välimuistissa oleva kopio eikä reaaliaikainen. GetStructureRequest -viestin rakenne näkyy kuvassa 5.15.

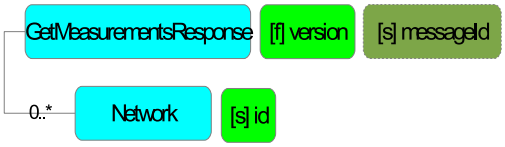
Vastauksena saatava GetStructureResponse sisältää vastauskoodin ja verkon rakenteen. Verkon rakenne voi olla joko täydellinen, jolloin vastauskoodi on 200 tai osittainen, jolloin vastauskoodi on 206. Verkon rakenteen lisäksi vastaus sisältää myös tiedon siitä, milloin solmu on viimeksi keskustellut palvelun kanssa. Vastausviestin rakenne näkyy kuvassa 5.16.

GetMeasurements -operaatiolla pyydetään palvelulta tieto siitä mitä tietoja ja millä aikaväleillä ne kerätään. GetMeasurementsRequest -viestillä pyydetään palvelulta mittausparametrit. Jos viesti ei sisällä network -elementtejä, vastauksena lähetetään jokaisen verkon asetukset. Viestin rakenne näkyy kuvassa 5.17.

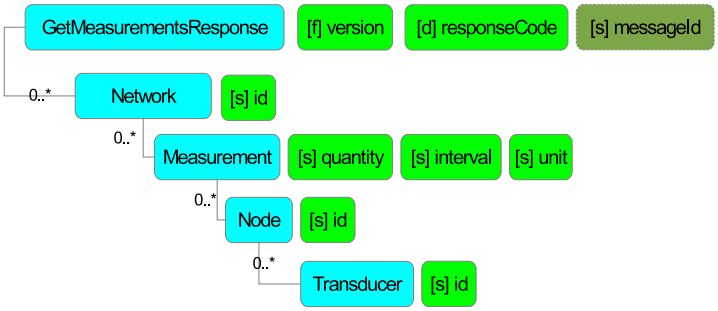
Vastausviesti GetMeasurementsResponse sisältää mittaustyypin, määrän, aikavälin ja yksin solmu- ja mittalaitekohtaisesti. Viestin rakenne näkyy kuvassa 5.18



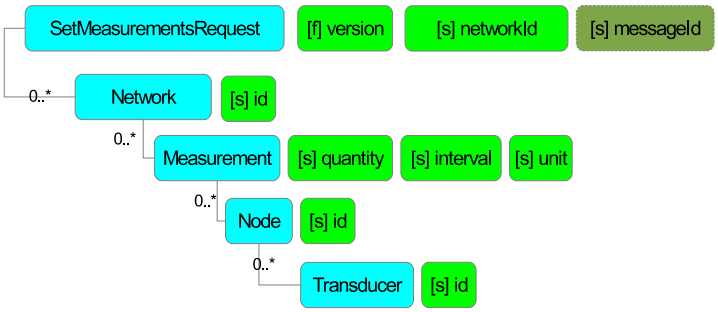
Kuva 5.16: Verkon rakenteen vastausviesti



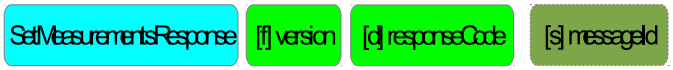
Kuva 5.17: Verkon mittaustietoasetusten pyyntö



Kuva 5.18: Verkon mittaustietoasetusten vastaus



Kuva 5.19: Verkon mittaustietoasetusviesti



Kuva 5.20: Verkon mittaustietoasetusviestin vastaus

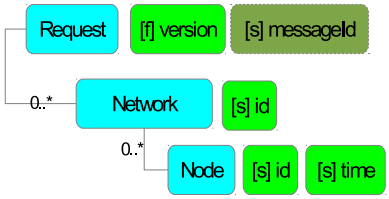
SetMeasurements -operaatiolla asetetaan verkon mittaustietojen keräyksen käytännöt. Käytäntö määrittelee mitä tietoa verkosta kerätään ja kuinka usein. Käytäntö voidaan määritellä verkko-, solmu- ja mittauslaitekohtaisesti. Operaatio sisältää SetMeasurementsRequest- ja SetMeasurementsResponse -viestit. Jos käytetty verkkoteknologia ei tue mittausten keräyskäytäntöjen muuttamista, sitä voidaan emuloida. Jos asetusviestissä ei ole määritelty erikseen solmua, asetus kohdistetaan verkon kaikkiin solmuihin. Viestien rakenne näkyy kuvista 5.19 ja 5.20.

5.5 MEDF

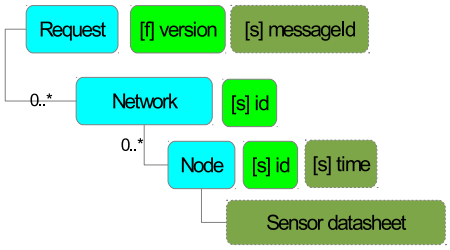
MEDF määrittelee metodit joilla käsitellään antureihin liittyvää metadataa. Metadataa on muun muassa solmun kuvaus, sarjanumero, valmistajat ja muu vastaava tieto.

Metadata pyydetään luettavaksi Request -viestillä. Viestin rakenne näkyy kuvassa 5.21. Vastausviestin rakenne on kuvassa 5.22.

Metadataa voidaan myös lisätä ja poistaa verkon tiedoista. Viestien rakenne noudattaa WSN Open API:n yleistä viestirakennetta. Käytetyt viestit ovat InsertRequest ja InsertResponse metadatan lisäämistä varten sekä RemoveRequest ja RemoveResponse metadatan poistoa varten.



Kuva 5.21: Verkon metadatan pyyntöviesti



Kuva 5.22: Verkon metadatan vastausviesti

5.6 Mitattavat suureet

WSN OpenAPI sisältää myös suositellun käytännön ja sanaston mittalaitteiden ja toimijoiden tarjoamien suureiden käyttöä varten, mutta käytäntöä ei ole pakko noudattaa WSN Open API -toteutuksessa. Koska tarjolla olevien erilaisten mittalaitteiden määrä on suuri, määritelmä esittää vain yleisimmät mittalaitteiden tarjoamat suureet. Mittaustiedolle suositeltu esitysmuoto löytyy listauksesta 5.1.

Listing 5.1: WSN Open API -mittaustiedon esitys

```
NAME = [ PREFIX ":" ] BASENAME
PREFIX = String
BASENAME = String
```

Etuliitettä "openapi"suositellaan käytettäväksi vain WSN Open API -määrittelyn mukaisten suureiden kohdalla. Muita nimeämiskäytäntöjä voidaan myös käyttää näiden tilalla tai rinnalla.

6 Case

Työn empiirisen osuuden tarkoitus oli tuoda Kokkolan yliopistokeskus Chydeniuksen SINNE-hankkeen tuottama mittaustieto REST-rajapinnan kautta Tampereen teknillisen yliopiston WSN OpenAPI:n määritelmän mukaisesti käytettäväksi, mutta XML- tai CSV-formaatin sijaan JSON-muodossa. WSN OpenAPI -rajapinnalle eli niin sanotulle backend -osalle toteuttiin myös asiakasohjelma tai frontend, jonka avulla voidaan hakea mittaustiedot tarjolla olevista verkoista halutulta ajanjaksolta. Asiakasohjelma tullaan tarjoamaan Kokkolan yliopistokeskuksen tietotekniikan maisteriohjelman opiskelijoiden käyttöön.

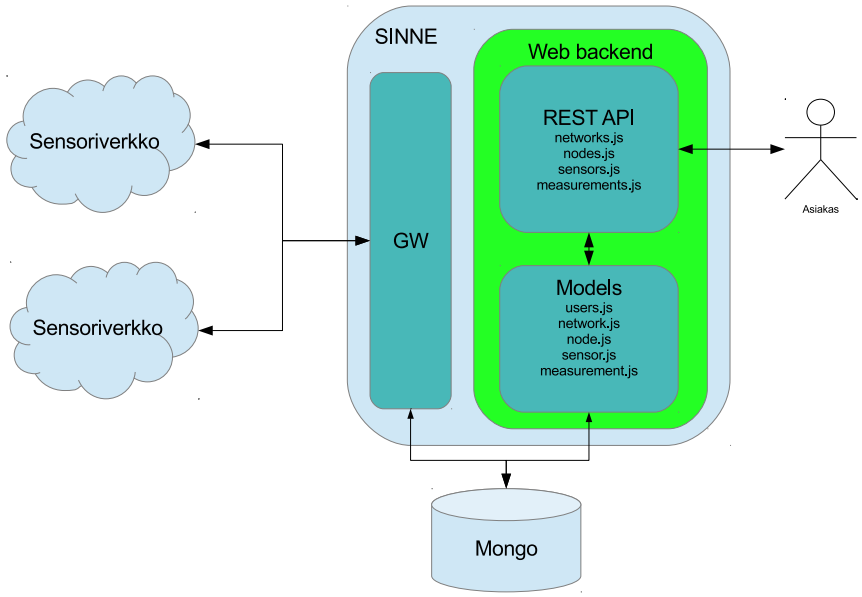
Tietojen esitystä varten tarjolla olevista vaihtoehdoista W3C:n Web of Things[27] oli vielä kirjoitushetkellä keskeneräinen. OGC:n SensorThings API:a[35] tarkasteltiin myös, mutta myös se oli hankkeen aloituksen aikaan vielä keskeneräinen. Tämän lisäksi WSN OpenAPI on rakenteeltaan edellä mainittuja yksinkertaisempi. Järjestelmän muokkaaminen esittämään mittaustiedot myös WSN OpenAPI:n vaatimassa muodossa katsottiin helpommaksi. Näin ollen WSN OpenAPI vastasi parhaiten SINNE-hankkeen vaatimuksiin.

Olemassa olevan järjestelmän tiedon tallennus- ja esitysmuodot poikkeavat WSN OpenAPI:n määritelmästä merkittävästi joten työn keskeinen osa oli tietokantaan tallennetun datan etsiminen, parsiminen ja sen muokkaaminen haluttuun muotoon.

6.1 Ympäristön kuvaus

SINNE[52] on Kokkolan yliopistokeskus Chydeniuksen hanke, jossa kehitetään skaalautuvia ilmanlaatua ja melua mittaavia langattomia verkkoja. Itse SINNE-järjestelmä koostuu sensoriverkosta, jonka osia ovat antureita sisältävät solmut ja tiedon keräävä laite (sink), yhdyskäytävänä toimiva Raspberry Pi sekä Cinet -palvelin, joka tarjoaa REST -rajapinnan sekä MongoDB -tietokannan. Järjestelmäarkkitehtuuri näkyy kuvassa 6.1. Kuvassa 6.2 on noodi.

SINNE-järjestelmään kuuluu siis itse sensoriverkko, joka toimittaa mittaukset raakadatana yhdyskäytävälle, jossa ne muunnetaan luettavampaan muotoon. Yhdyskäytävältä tieto viedään MongoDB-tietokantaan. SINNE-järjestelmä tarjoaa asiakkaiden käyttöön WSN OpenAPI:n tiedostomuotoa noudattavan rajapinnan. Rajapinnan kautta asiakasohjelmien käyttöön tarjotaan autentikointi ja sensoriverkon



Kuva 6.1: SINNE-järjestelmäarkkitehtuuri

mittaustiedot eri ajanhetkinä. Mittaustiedot tarjotaan REST-arkkitehtuurityyliin mukaisesti ja käytetty protokolla on HTTP/HTTPS.

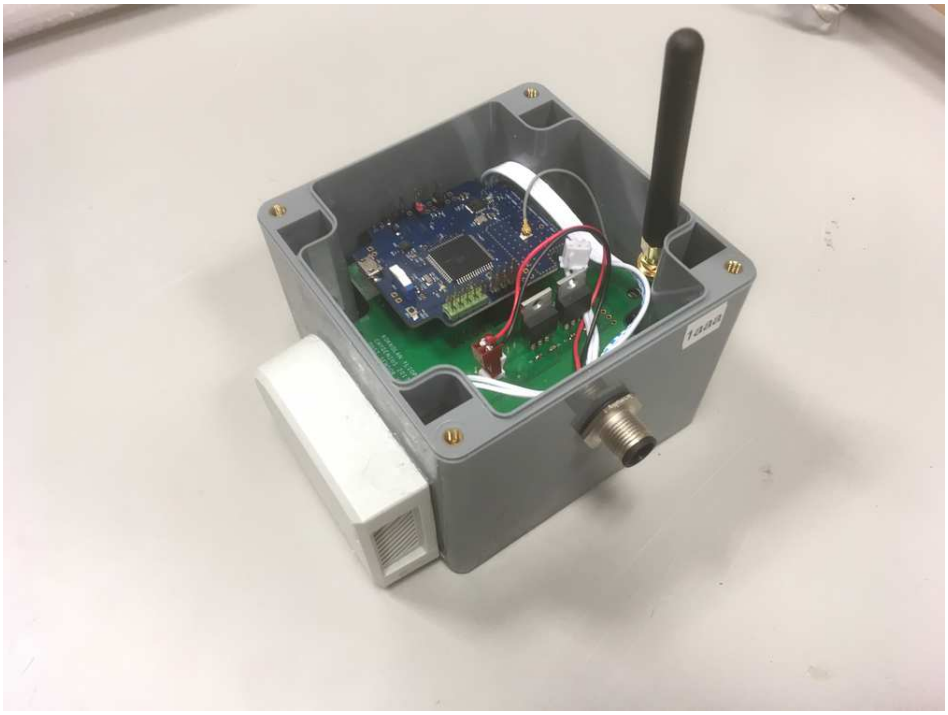
6.1.1 Tiedon esitys tallennus

SINNE-järjestelmän tuottamat mittaustiedot tallennetaan palvelimelle POST-metodilla käyttäen URL:ia `https://[palvelin]/apiv2/storage/network_id`. Usean minuutin mittaustiedot tallennetaan koko verkosta kerralla virran säästämiseksi. Jos tiedon tallennus ei onnistu esimerkiksi verkkovirheen tai palvelinohjelmiston kaatumisen takia, tiedot tallennetaan muistiin yhdyskäytävän paikalliseen tietovarastoon, mutta niitä ei yritetä sen jälkeen tallentaa uudestaan Cinet-tietokantaan.

Mittaustiedot tallennetaan Mongo-tietokantaan JSON-formaatissa. Yksittäinen mittaus tietyllä ajanhetkellä näkyy esimerkissä 6.1.

Listing 6.1: Alkuperäinen mittaustiedon esitys

```
{
  "networkId": "network_1",
  "nodeId": "001F3996",
  "sensorId": "battery",
  "measurements": [
    {
```



Kuva 6.2: Noodi

```
        "type": "battery",
        "value": 9.474,
        "timestamp": "2018-09-28T13:55:05.170Z"
      }
    ]
}
```

Tieto tallennetaan tietokantaan niin sanoittuihin kokoelmiin (collection). Kokoelmia ovat käyttäjätiedot tallentava users, networks, joka sisältää listan verkoista, jokaisen verkon oma kokoelma, joka sisältää mittaustulokset ja datasources, johon on tallennettu noodi ja anturi yhdessä.

6.2 CINET WSN Open API

Uudessa järjestelmässä mittaustiedot ovat saatavilla hierarkkisesti jaoteltujen URL -osoitteiden kautta. Merkittävän ero SINNE-järjestelmän rajapintaan on nykyisen järjestelmän datasources -kokoelman tiedon jakaminen erikseen noodiin ja anturei-hin sekä datapoint -tietueiden esittäminen noodin ja anturin kautta mittaustieto-tietueena. Olemassa olevaa tietoa ei kuitenkaan muokattu mitenkään, vaan tehtiin

Taulukko 6.1: Resurssit ja URL:t

Resurssi	URL
Listaus verkoista	/apiv2/
Verkon tiedot	/apiv2/:networkId
Verkon solmut	/apiv2/:networkId
Solmun tiedot	/apiv2/:networkId/:nodeId
Solmun mittalaitteen tiedot	/apiv2/:networkId/:nodeId/:sensorId
Mittaukset	/apiv2/:networkId/:nodeId/:sensorId/measurement

erillinen rajapinta, jonka kautta voidaan hakea tietoa eri muodossa.

6.2.1 Rajapinta

Rajapinta toteutettiin HTTP/HTTPS -muotoisena. HTTP:n operaatioista käytettävissä ovat vain GET ja POST, joista jälkimmäistä käytetään ainoastaan kirjautumisen yhteydessä. Lista kaikista tarjolla olevista verkoista saadaan seuraavalla komennolla:

```
GET https://cinet.chydenius.fi/cinet-wsnopenapi/apiv2/
```

Pyyntö tarkistaa käyttäjän oikeudet eri verkkoihin ja palauttaa tuloksena vain ne verkot, jotka on sallittu. Vastaukset eivät sisällä linkkejä suoraan toisiin resursseihin, mutta kun tiedetään urlien logiikka ja verkon noodit, niin voidaan rakentaa URL:t niiden perusteella. Esimerkiksi halutun verkon tietyn anturin akun viimeisimmän mittausarvon saa seuraavalla komennolla:

```
GET https://cinet.chydenius.fi/cinet-wsnopenapi/
apiv2/lore/0xE105/battery/measurement
```

Rajapinnalta voidaan pyytää myös aikaleimoilla ilmoitetun ajanjakson kaikki mittaustulokset. Tällöin URL:iin lisätään start- ja end-parametrit muodossa YYYY-MM-DDTHH:MM:SSZ. Järjestelmän tarjoamat URL:t eri resursseille löytyvät taulukosta 6.1. WSN OpenAPI:n käytettyjen rajapintojen toteutukset Cinet WSN OpenAPI-järjestelmässä ovat koottuna taulukkoon 6.2. Rajapinta toteuttaa myös vaaditut prosessointipyynnöt eli voidaan pyytää pienin ja suurin arvo aikaväliltä, aikavälin keskiarvo, mittausten määrä ja mittausarvojen summa.

Esimerkiksi `https://cinet.chydenius.fi/cinet-wsnopenapi/apiv2/sinne/` viittaa palvelimella olevaan sinne-nimiseen verkkoon. Solmuilla voi olla mittaus-

Taulukko 6.2: Rajapintojen vastaavuudet

WSN OpenAPI -operaatio	SINNE REST API
AuthenticationRequest	POST /apiv2/authenticate
GetStructure	GET /apiv2/
Sensor Archive Data Format Query	GET /apiv2/:networkId/:nodeId/ :sensorId/measurement/

laitteina esimerkiksi äänenvoimakkuus (noise), tuuli (wind) ja lämpötila (temperature).

Kaikki muut resurssit toimivat myös samalla logiikalla ja tiedot saadaan haettua GET-operaatiolla.

6.2.2 Tiedon esitys

Luvussa viisi esitettiin WSN OpenAPI:n viestiformaatit XML- ja CSV-muodoissa. Seuraavaksi käydään läpi esimerkein eri WSN OpenAPI:n määritelmää käyttäen JSON-versiot tarvittavista viestiformaateista.

Tiedon esityksen hierarkkinen muoto näkyy koodiesimerkissä A.1. Loput WSN OpenAPI-määrittelyn mukaiset vastausviestit löytyvät liitteestä A.

Listing 6.2: WSN OpenAPI:n mukainen JSON-vastaus

```
{
  "version": "1.4",
        "responseCode": "200",
        "messageId": "1",
        "networks": [
    {
      "name": "Test network",
      "networkId": "network_1",
      "nodes": [
        {
          "nodeId": "001F3996",
          "name": "LoraNode"
        }
      ]
    },
```

```
    {
      "name": "Rooms",
      "networkId": "rooms",
      "nodes": [
        {
          "nodeId": "0x1001",
          "name": "0x1001"
        },
        {
          "nodeId": "0xA009",
          "name": "KARVO"
        }
      ]
    }
  ]
}
```

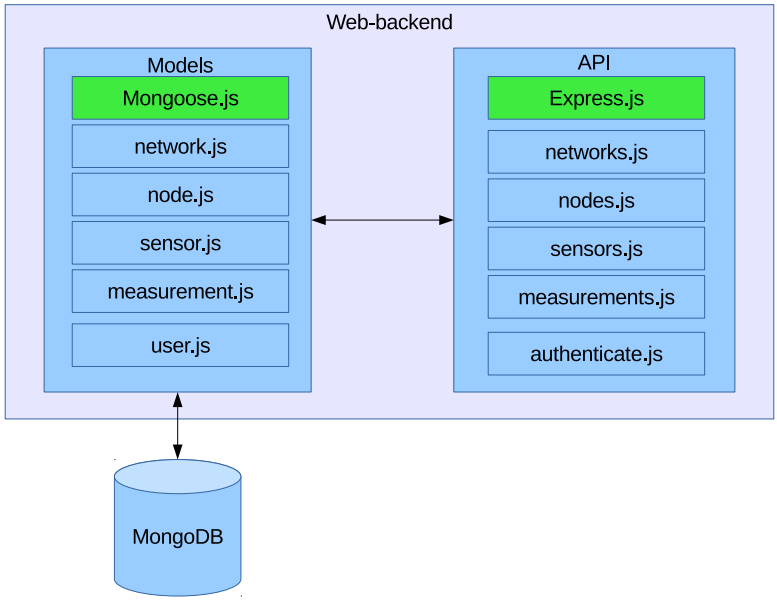
6.2.3 Backend -järjestelmän modulit

REST-rajapinta on toteutettu node.js:n avulla. Tietokantayhteys hoidetaan Mongoose -kirjastolla ja REST-rajapinnan mukaiset URL:t tarjotaan express.js:n avulla. Koko järjestelmän runkona on web-backend -moduli. Tämä käynnistää järjestelmän eri palvelut.

Järjestelmä jakautuu rakenteellisesti kahteen eri osaan, API ja Models. API pitää sisällään rajapinnan asiakkaalle näkyvän osan. API kattaa siis eri express.js -reitit ja niiden toiminnan. Models sisältää tietokantayhteyden käyttämät määritelmät ja tietokannan skeeman. Arkkitehtuuri ja modulien yhteyden näkyvät kuvassa 6.3.

Sensoriverkkojen tiedot tarjotaan networks.js ja network.js -modulien avulla. Nämä modulit tarjoavat API:n juuresta GET -pyynnölle vastaukseksi listauksen eri verkoista ja verkon tunnisteella tiedot kyseessä olevasta verkosta, mukaan lukien verkkoon kuuluvat noodit. Nodes.js ja node.js vastaavat yksittäisen noodin tietojen tarjoamisesta. Noodin tietoihin kuuluu muun muassa sen tarjoamat anturit. Sensors.js ja sensor.js listaavat vastaavasti yksittäisen sensorin tiedot. Measurements.js ja measurement.js hakevat joko viimeisimmän mittaustiedon tai aikaleimoilla eroteltuna tietyn aikavälin mittaukset.

Näiden lisäksi järjestelmä edellyttää tunnistautumisen. Authenticate.js ja user.js vastaavat käyttäjätietojen hakemisesta kannasta ja autentikoinnista JWT:n (JSON



Kuva 6.3: Backend-arkkitehtuuri

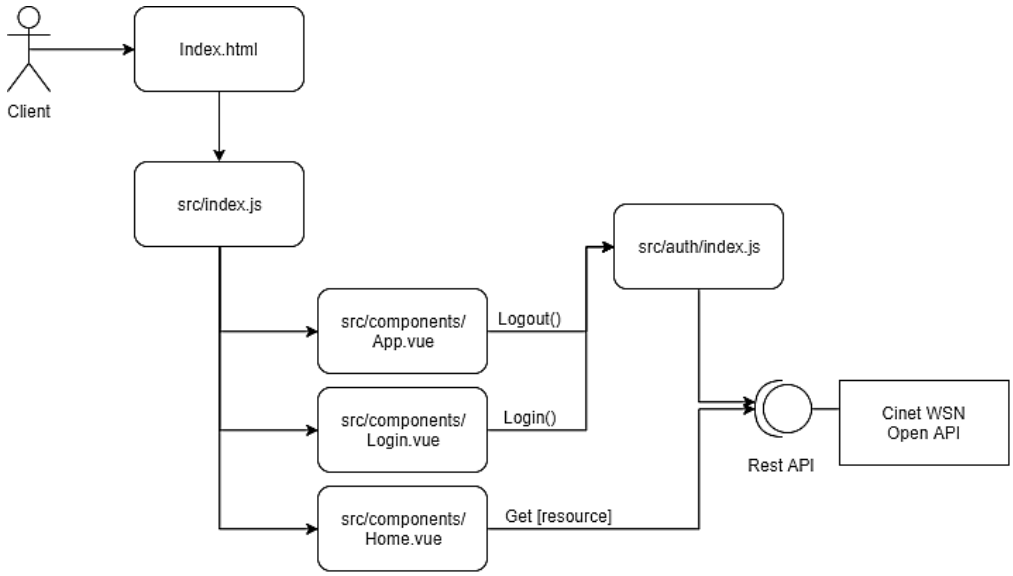
web token) avulla.

6.3 Asiakasohjelma

REST-rajapinnan lisäksi tehtiin Vue.js:llä toteutettu yksinkertainen sovellus, joka tarjoaa REST-rajapinnasta haetun tiedon kirjautumiseen käytetyn käyttäjätunnuksen näkemistä verkoista, näiden solmuista ja mittalaitteista. Asiakasohjelma koostuu HTML-sivusta, joka lataa käännetyn javascript-tiedoston. Ulkoasu on muotoiltu Bootstrap-kirjastolla.

Asiakasohjelma koostuu viidestä modulista. Perusnäkymä rakennetaan src/index.js:n avulla. Tämä moduli koostaa reitityksen ja tarjoaa kirjautumisnäkymän oletuksena. Kirjautumisen logiikan hoitaa src/auth/index.js. Kirjautumismoduli sisältää login, logout, checkAuth ja getAuthHeader -metodit. Käyttöliittymän logiikan tarjoaa App.vue, Home.vue ja Login.vue -modulit. App.vue määrittää sivun ulkoasun ja asettelun. Login.vue tarjoaa kirjautumistoiminnot. Tietojen haku, niiden käsittely ja esittäminen tapahtuu Home.vue -modulissa. Asiakasohjelman arkkitehtuuri näkyy kuvassa 6.4.

Asiakasohjelman peruskäyttöliittymä näkyy kuvassa 6.5. Käyttäjä voi valita ensin haluamansa ajanjakson kuten kuvassa 6.6 näkyy. Aikavälin voi jättää myös tyhjäksi, jolloin haetaan vain viimeisin mittaustulos. Seuraavaksi käyttäjä valitsee pu-



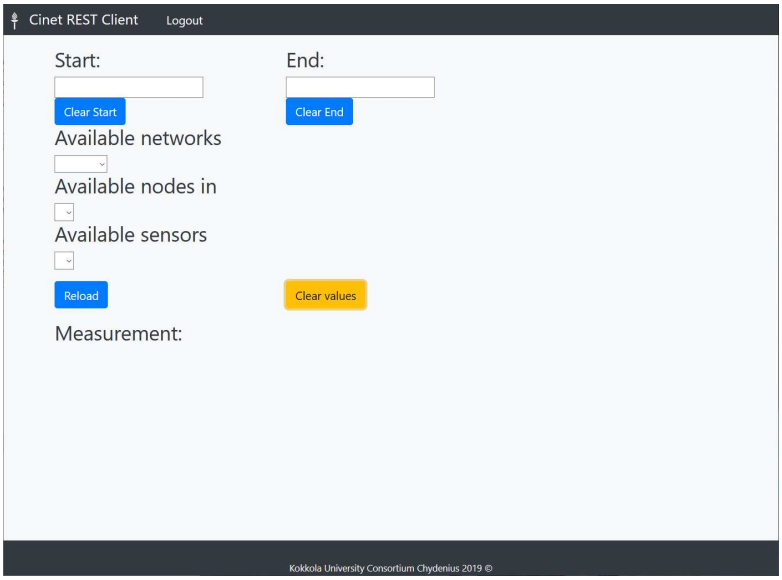
Kuva 6.4: Asiakasohjelman arkkitehtuuri

dotusvalikosta tarjotuista verkoista haluamansa. Tämän jälkeen ohjelma hakee rajapinnan kautta listan verkon noodeista ja asettaa noodit pudotusvalikkoon. Noodin valinnan jälkeen ohjelma hakee listan noodin antureista. Käyttäjän valittua haluamansa anturin haetaan aikavälin mukaiset tiedot tai viimeisin tieto. Mittaustulokset esitetään kuvan 6.7 mukaisesti.

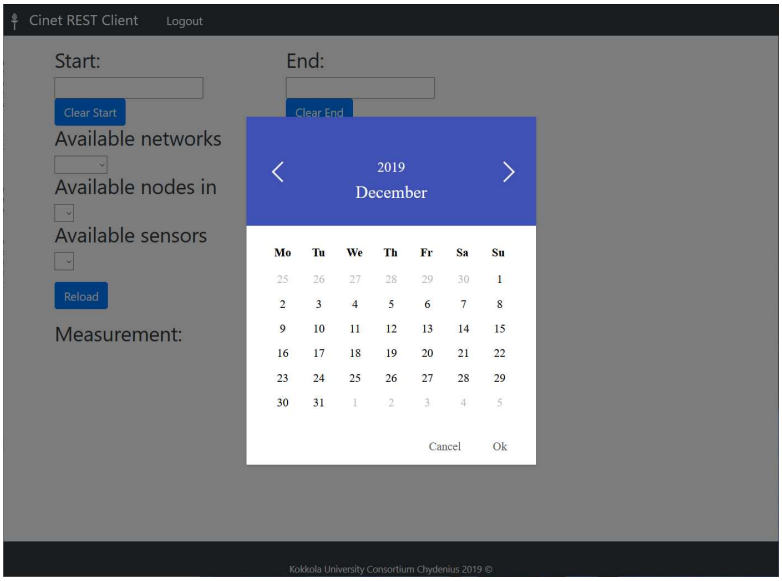
6.4 Ajatuksia ja havaintoja

Käytetty JSON-formaatti helpotti tiedon hyödyntämistä, koska se on hyvin tuettu nykyaikaisten kehitystyökalujen ja ohjelmointikielten puolesta. Tiedon esitys on myös helppolukuinen, joten asiakasohjelman kehitys sujui nopeasti. Myös resurssien URL:ien hierarkkisuus selkeytti tiedon hakua.

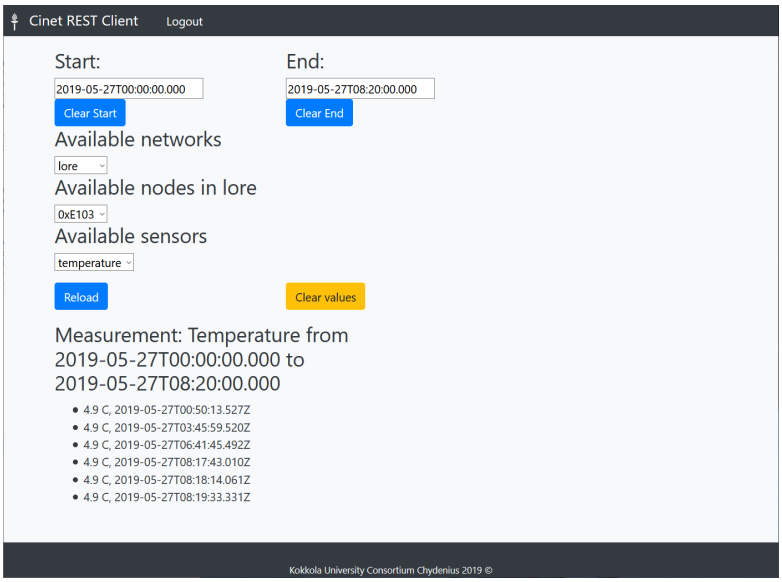
Yhdyskäytävää tehdessä olemassa olevan datan saattaminen haluttuun muotoon oli kohtalaisen työlästä. Tarkkuutta vaati erityisesti SINNE-järjestelmän niin sanottujen datapoint- ja datasource -tietuiden yhdistäminen ja käsittely erikseen noodeina, antureina ja mittaustuloksina. Onneksi kaikki tarvittava tieto oli kuitenkin olemassa kannassa ja eroteltavissa.



Kuva 6.5: Asiakasohjelman käyttöliittymä



Kuva 6.6: Ajankohdan valinta



Kuva 6.7: Asiakasohjelman hakutulos

7 Yhteenveto

Työn tarkoituksena oli selvittää mitä ovat WWW-sovelluspalvelut ja sensoriverkot sekä miten sensoriverkkojen keräämä tieto saadaan esitettyä ja käytettäväksi WWW-sovelluspalveluteknologian avulla. Lisäksi toteutettiin palvelu, joka tarjoaa sensoriverkon mittaustiedot tietyn formaatin mukaisesti sekä osoitettiin tämän palvelun käytettävyys yksinkertaisella asiakasohjelmalla.

Mittaustietojen esityksestä on olemassa montaa eri käytäntöä ja keskenään kilpailevaa standardia. Tarjolla olevat standardit ovat hyvin erilaisia ja niiden fokus on jokaisella eri. OGC:n standardi keskittyy enemmän paikkatietoon ja eri mittauslaitteet ovat erilaisessa asemassa verrattuna WSN Open API:in.

Tarjolla olevista eri WWW-sovelluspalveluvaihtoehdoista toteutukseen valittiin REST. REST tarjoaa löyhän kytkennän ja yhdenmukaisen rajapinnan. Rest myös mahdollisti nopean kehityksen. REST ei myöskään sido mihinkään tiettyyn tiedonsiirtoprotokollaan ja halutessa voitaisiin toteuttaa vastaanlainen rajapinta esimerkiksi CoAP:n avulla. Protokollaksi valittiin kuitenkin HTTP/HTTPS, koska mittaustiedot halutaan tarjota käytettäväksi WWW:n kautta.

REST -arkkitehtuurityyli ei kuitenkaan ihan suoraan sovellu WSN OpenAPI:n toteutukseen, koska WSN OpenAPI:ssa määritellyt rajapinnat ovat enemmän RPC-tyylisiä. Näin ollen toteutettavaksi valittiin vain tiedon esitys. Tiedon esityksen hierarkisuus ohjasi myös resurssien URL:ien suunnittelua hierarkiseen tyyliin. Yhdestä järjestelmästä saatava tieto voidaan tarjota rajapinnan kautta monessa eri muodossa. REST ei ota kantaa tiedon esitykseen, vaan tiedon esitys on WSN Open API:n mukainen.

WSN OpenAPI on kuitenkin pienen toimijan kehittämä eikä todennäköisesti tule yleistymään, joten jatkoa ajatellen tulee selvittää isompien standardointiorganisaatioiden projektien tilanne. Hierarkisesti tallennettu tieto on kuitenkin mahdollista esittää toisen organisaation käyttämän standardin mukaisessa muodossa esimerkiksi avaamalla sille kokonaan oma rajapinta tai palauttamalla parametrisoidulle pyynnölle eri muodossa oleva esitys. Jos halutaan tarjota tiedot jonkin toisen standardin mukaisesti, ongelmia voi syntyä jos ei ole edes kaikkia toteutettavan rajapinnan vaatimia tietoja. Tällöin joutuu lisäämään tietokantaan tallennettavaa tietoa tai käyttämään jotain vakioarvoa palauttaessa vastaukset.

Niin kauan kuin ei ole yhtä yleisesti käytettyä standardia, siiloutuminen ja yh-

teensopivuusongelmat on yleensä ratkaistava yhdyskäytäviä käyttäen. Tämä tarkoittaa jokaiselle erilliselle järjestelmälle kehitettävää omaa sovitinkerrosta.

Lähteet

[1] AIHKISALO, T., JA PAASO, T. Latencies of service invocation and processing of the rest and soap web service interfaces. *IEEE Eighth World Congress on Services* (2012), 100–107.

[2] AKYILDIZ, I. F., SU, W., SANKARASUBRAMANIAM, Y., JA CAYIRCI, E. Wireless sensor networks: A survey. *Comput. Netw. 38*, 4 (Mar. 2002), 393–422.

[3] ALLIANCE, Z. Zigbee specification, 2017. URL: http://www.zigbee.org/download/standards-zigbee-specification/, haettu 5.6.2018.

[4] ATZORI, L., IERA, A., JA MORABITO, G. The internet of things: A survey. *Computer Networks 54*, 15 (2010), 2787 – 2805.

[5] BANKS, A., JA GUPTA, R. Mqtt version 3.1.1, 2014. URL: `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf`, viitattu 30.3.2016.

[6] BARRY, D. K. *Web Services and Service Oriented Architectures - A Savvy Manager's Guide*. Addison-Wesley, San Fransisco, CA, USA, 2003.

[7] BARTHOLOMEW, D. Sql vs. nosql. *Linux J. 2010*, 195 (July 2010).

[8] BOOTH, D., HAAS, H., JA MCCABE, F. Web services architecture. URL: http://www.w3.org/TR/ws-arch/wsa.pdf, haettu 14.8.2015.

[9] BORMANN, C., CASTELLANI, A., JA SHELBY, Z. Coap: An application protocol for billions of tiny internet nodes. *Internet Computing, IEEE 16*, 2 (March 2012), 62–67.

[10] BORMANN, C., LEMAY, S., TSCHOFENIG, H., HARTKE, K., SILVERAJAN, B., JA RAYMOR, B. CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets. RFC 8323, Feb. 2018.

[11] BRAY, T. The javascript object notation (json) data interchange format. RFC 7159, RFC Editor, March 2014. `http://www.rfc-editor.org/rfc/rfc7159.txt`.

[12] BRAY, T., PAOLI, J., SPERBERG-MCQUEEN, C. M., MALER, E., JA YERGEAU, F. Extensible markup language (xml) 1.0 (fifth edition). World Wide Web Consortium, Recommendation REC-xml-20081126, November 2008.

[13] CATTELL, R. Scalable sql and nosql data stores. *SIGMOD Rec. 39*, 4 (May 2011), 12–27.

[14] CENTENARO, M., VANGELISTA, L., ZANELLA, A., JA ZORZI, M. Long-range communications in unlicensed bands: the rising stars in the iot and smart city scenarios. *IEEE Wireless Communications 23*, 5 (October 2016), 60–67.

[15] CHRISTENSEN, E., CURBERA, F., JA MEREDITH, GREG JA WEERAWARANA, S. Web service definition language (wsdl), March 2001. URL: http://www.w3.org/TR/wsdl, haettu 22.1.2013.

[16] CURBERA, F., DUFTLER, M., KHALAF, R., NAGY, W., MUKHI, N., JA WEERAWARANA, S. Unraveling the web services web: an introduction to soap, wsdl, and uddi. *Internet Computing, IEEE 6*, 2 (March 2002), 86–93.

[17] DESAI, P., SHETH, A., JA ANANTHARAM, P. Semantic gateway as a service architecture for iot interoperability. Julkaisusarjassa *2015 IEEE International Conference on Mobile Services* (June 2015), pp. 313–319.

[18] FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., JA BERNERS-LEE, T. Hypertext transfer protocol – http/1.1. URL: http://www.rfc-base.org/rfc-2616.html, haettu 14.8.2015.

[19] FIELDING, R. T. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, 2000.

[20] GOMEZ, C., JA PARADELLS, J. Wireless home automation networks: A survey of architectures and technologies. *IEEE Communications Magazine 48*, 6 (June 2010), 92–101.

[21] GUDGIN, M., HADLEY, M., MENDELSOHN, N., LAFON, Y., MOREAU, J.-J., KARMARKAR, A., JA NIELSEN, H. F. SOAP version 1.2 part 1: Messaging framework (second edition). W3C recommendation, W3C, Apr. 2007. URL: http://www.w3.org/TR/2007/REC-soap12-part1-20070427/, haettu 22.1.2013.

[22] GUDGIN, M., HADLEY, M., MENDELSOHN, N., MOREAU, J.-J., FRYSTYK NIELSEN, H., KARMARKAR, A., JA LAFON, Y. Soap version 1.2 part 1: Messaging framework (second edition). World Wide Web Consortium, Recommendation REC-soap12-part1-20070427, April 2007. haettu 9.7.2014.

[23] GUINARD, D., ION, I., JA MAYER, S. In search of an internet of things service architecture: Rest or ws-*? a developers' perspective. Julkaisusarjassa *Mobile and Ubiquitous Systems: Computing, Networking, and Services* (2011), Springer Berlin Heidelberg, pp. 326–337.

[24] HUNKELER, U., TRUONG, H. L., JA STANFORD-CLARK, A. Mqtt-s - a publish/-subscribe protocol for wireless sensor networks. Julkaisusarjassa *Communication Systems Software and Middleware and Workshops, 2008. COMSWARE 2008. 3rd International Conference on* (Jan 2008), pp. 791–798.

[25] JONES, S. Toward an acceptable definition of service. *IEEE Software 22*, 3 (2005), 87–93.

[26] KAEBISCH, S., JA KAMIUA, T. Web of things (wot) thing description, 2018. URL: https://w3c.github.io/wot-thing-description/, haettu 28.3.2018.

[27] KAJIMOTO, K., MATSUKURA, R., HUND, J., KOVATSCH, M., JA NIMURA, K. Web of things (wot) architecture, 2017. URL: https://w3c.github.io/wot-architecture/, haettu 21.3.2018.

[28] KALIN, M. *Java Web Services: Up and Running*, 1st ed. O'Reilly Media, Inc., Sebastopol, CA, USA, 2009.

[29] KARVONEN, H., SUHONEN, J., PETÄJÄJÄRVI, J., HÄMÄLÄINEN, M., HÄNNIKÄINEN, M., JA POUTTU, A. Hierarchical architecture for multi-technology wireless sensor networks for critical infrastructure protection. *Wireless Personal Communications 76*, 2 (2014), 209–229.

[30] KIS, Z., NIMURA, K., PEINTNER, D., JA HUND, J. Web of things (wot) scripting api, 2017. URL: https://w3c.github.io/wot-scripting-api/, haettu 2.5.2018.

[31] KLINE, K., KLINE, D., JA HUNT, B. *SQL in a Nutshell*, 3rd ed. O'Reilly Media, Inc., 2008.

[32] LANDRE, E., JA WESENBERG, H. Rest versus soap as architectural style for web services. *5th International OOPSLA Workshop on SOA & Web Services Best Practices* (2007).

[33] LANTHALER, M., JA GUTL, C. Towards a restful service ecosystem. Julkaisusarjassa *2010 4th IEEE International Conference on Digital Ecosystems and Technologies (DEST)* (2010), pp. 209–214.

[34] LI, Y., JA MANOHARAN, S. A performance comparison of sql and nosql databases. Julkaisusarjassa *Communications, computers and signal processing (PACRIM), 2013 IEEE pacific rim conference on* (2013), IEEE, pp. 15–19.

[35] LIANG, S. H., HUANG, C.-Y., JA KHALAFBEIGI, T. Ogc sensorthings api part i:sensing. Tekninen raportti, OGC, 2016. URL: http://docs.opengeospatial.org/is/15-078r6/15-078r6.html, haettu 23.4.2018.

[36] LOUGHRAN, S., JA SMITH, E. Rethinking the java soap stack. Julkaisusarjassa *IEEE International Conference on Web Services (ICWS)* (2005), vol. 5, pp. 12–15.

[37] MASSE, M. *REST API Design Rulebook*. Oreilly and Associate Series. O'Reilly Media, Sebastopol, CA, USA, 2011.

[38] MEKKI, K., BAJIC, E., CHAXEL, F., JA MEYER, F. Overview of cellular LPWAN technologies for IoT deployment: Sigfox, LoRaWAN, and NB-IoT. Julkaisusarjassa *2nd IEEE International Workshop on Mobile and Pervasive Internet of Things, PerIoT 2018* (Athens, Greece, Mar. 2018). Part of International Conference on Pervasive Computing and Communications, PerCom2018, Athens, Greece, March 19-23, 2018.

[39] MONTENEGRO, G., HUI, J., CULLER, D., JA KUSHALNAGAR, N. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944, Sept. 2007.

[40] PAPAZOGLOU, M. P. Service-oriented computing: concepts, characteristics and directions. Julkaisusarjassa *Proceedings of the Fourth International Conference on Web Information Systems Engineering* (2003), pp. 3–12.

[41] PAPAZOGLOU, M. P., JA GEORGAKOPOULOS, D. Introduction: Service-oriented computing. Julkaisusarjassa *Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003. WISE 2003.* (2003), vol. 46, 10, ACM, pp. 24–28.

[42] PAUTASSO, C., JA WILDE, E. Why is the web loosely coupled?: A multi-faceted metric for service design. Julkaisusarjassa *Proceedings of the 18th International Conference on World Wide Web* (2009), ACM, pp. 911–920.

[43] PAUTASSO, C., ZIMMERMANN, O., JA LEYMANN, F. Restful web services vs. "big" web services: making the right architectural decision. Julkaisusarjassa *Proceedings of the 17th international conference on World Wide Web* (2008), ACM, pp. 805–814.

[44] PRIYANTHA, N. B., KANSAL, A., GORACZKO, M., JA ZHAO, F. Tiny web services: design and implementation of interoperable and evolvable sensor networks. Julkaisusarjassa *Proceedings of the 6th ACM conference on Embedded network sensor systems* (2008), pp. 253–266.

[45] RAWAT, P., SINGH, K. D., CHAOUCHI, H., JA BONNIN, J. M. Wireless sensor networks: a survey on recent developments and potential synergies. *The Journal of Supercomputing 68*, 1 (2013), 1–48.

[46] RICHARDSON, L., JA RUBY, S. *RESTful web services*. O'Reilly Media, Incorporated, Sebastopol, CA, USA, 2007.

[47] ROBIN, A. Swe common data model encoding standard. *OGC: Wayland, MA, USA* (2011). URL: http://www.opengeospatial.org/standards/swecommon, haettu 23.4.2018.

[48] ROWLAND, C., GOODMAN, E., CHARLIER, M., LIGHT, A., JA LUI, A. *Designing Connected Products: UX for the Consumer Internet of Things*, 1st ed. O'Reilly Media, Inc., 2015.

[49] SHAFRANOVICH, Y. Common format and mime type for comma-separated values (csv) files. RFC 4180, RFC Editor, October 2005. `http://www.rfc-editor.org/rfc/rfc4180.txt`.

[50] SHELBY, Z. Embedded web services. *Wireless Communications, IEEE 17*, 6 (2010), 52–57.

[51] SHELBY, Z., ARM, HARTKE, K., JA BORMANN, C. The constrained application protocol (coap), 2014. haettu 30.3.2016.

[52] Sinne-projekti. URL: http://cinetcampus.fi/en/projects/sinne, haettu 9.12.2017.

[53] SNELL, J., TIDWELL, D., JA KULCHENKO, P. *Programming Web services with SOAP*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

[54] SUHONEN, J., KIVELÄ, O., VUORI, M., RENTTO, M., JÄNTTI, T., LAUKKARINEN, T., KAUTTO, I., JA HÄNNIKÄINEN, M. Wsn openapi technical specification. Tekninen raportti, Tampere university of technology, 2013. haettu 26.1.2017.

[55] SUHONEN, J., KIVELÄ, O., LAUKKARINEN, T., JA HÄNNIKÄINEN, M. Unified service access for wireless sensor networks. Julkaisusarjassa *Proceedings of the Third International Workshop on Software Engineering for Sensor Network Applications* (Piscataway, NJ, USA, 2012), SESENA '12, IEEE Press, pp. 49–55.

[56] THANGAVEL, D., MA, X., VALERA, A., TAN, H.-X., JA TAN, C. K. Y. Performance evaluation of mqtt and coap via a common middleware. Julkaisusarjassa *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on* (April 2014), pp. 1–6.

[57] VASSEUR, J.-P., JA DUNKELS, A. *Interconnecting Smart Objects with IP: The Next Internet*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010.

[58] WEBBER, J. *REST in Practice: Hypermedia and Systems Architecture*. O'Reilly, Sebastopol, CA, USA, 2010.

[59] YAZAR, D., JA DUNKELS, A. Efficient application integration in ip-based sensor networks. Julkaisusarjassa *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings* (2009), pp. 43–48.

[60] ZENG, D., GUO, S., JA CHENG, Z. The web of things: A survey (invited paper). *Journal of Communications 6*, 6 (2011).

[61] ZUR MUEHLEN, M., NICKERSON, J. V., JA SWENSON, K. D. Developing web services choreography standards: the case of rest vs. soap. *Decis. Support Syst. 40*, 1 (2005), 9–29.

A WSN OpenAPI -viestit

A.1 Get Networks


```
{
  "version": "1.4",
  "responseCode": "200",
  "messageId": "1",
  "networks":
  [
    {
      "name": "Test network",
      "networkId": "network_1",
      "nodes":
      [
        {
          "nodeId": "001F3996",
          "name": "LoraNode"
        }
      ]
    },
    {
      "name": "Rooms",
      "networkId": "rooms",
      "nodes":
      [
        {
          "nodeId": "0x1001",
          "name": "0x1001"
        },
        {
          "nodeId": "0xA009",
          "name": "KARVO"
        }
```

```
        ]
      }
    ]
}
```


A.2 Get Network


```
{
  "version": "1.4",
  "responseCode": "200",
  "messageId": "1",
  "networks":
  [
    {
      "name": "Test network",
      "networkId": "network_1",
      "nodes":
      [
        {
          "nodeId": "001F3996",
          "name": "LoraNode",
          "sensors":
          [
            {
              "name": "battery",
              "type": "battery",
            }
          ]
        }
      ]
    }
  ]
}
```


A.3 Get Node


```
{
  "version": "1.4",
```

```
  "responseCode": "200",
  "messageId": "1",
  "networks":
  [
    "networkId": "network_1",
    "nodes":
    [
      {
        "nodeId": "001F3996",
        "name": "LoraNode",
        "sensors":
        [
          {
            "name": "battery",
            "type": "battery",
            "triggers":
            [
              {}
            ]
          }
        ]
      }
    ]
  ]
}
```


A.4 Get Sensor


```
{
  "version": "1.4",
  "responseCode": "200",
  "messageId": "1",
  "networkId": "network_1",
  "nodeId": "001F3996",
  "sensorId": "battery",
  "type": ""
}
```

A.5 Get Measurement


```
{
  "response": {
    "version": "1.4",
    "responseCode": "200",
    "messageId": "1",
    "networkId": "network_1",
    "nodeId": "001F3996",
    "sensorId": "battery",
    "measurements":
    [
      {
        "type": "battery",
        "value": 9.474,
        "timestamp": "2018-09-28T13:55:05.170Z"
      }
    ]
  }
}
```