OULUN AMMATTIKORKEAKOULU

Arttu Kemppainen

MITTAUSKIRJASTON KEHITYS NR-RADIOLLE

MITTAUSKIRJASTON KEHITYS NR-RADIOLLE

Arttu Kemppainen
Opinnäytetyö
Syksy 2022
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

Tekijä(t): Arttu Kemppainen
Opinnäytetyön nimi: Mittauskirjaston kehitys NR-radiolle
Työn ohjaaja(t): Teemu Leppänen
Työn valmistumislukukausi ja -vuosi: Syksy 2022 Sivumäärä: 37

Opinnäytetyön tavoitteena on toteuttaa nykyisen mittausskriptin pohjalta uusi mittauskirjasto osaksi NR-radion ohjelmiston testiympäristöä. Opinnäytetyön toimeksiantaja on Nokia Solutions and Networks Oy ja mittauskirjasto toteutetaan Nokialla Radio-CI-tiimissä.

Radio-CI:n tarkoitus on testata eri NR- ja LTE-radioiden ohjelmistoja sekä kehittää ja automatisoida testiympäristöjä, joilla radioiden ohjelmistoja testataan.

Työn tuloksena on mittauskirjasto, joka on edeltäjäänsä selkeämpi ja helpommin ylläpidettävä. Mittauskirjastoa pystyy ajamaan käyttäen Robot Framework -testiautomaatiotyökalua käyttäen ja se tukee uutta kantataajuusyksikköä.

Asiasanat:

Python, Robot Framework, Radio, RF, 5G, NR, CI, testiautomaatio, jatkuva integraatio

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Software Development

---

Author(s): Arttu Kemppainen
Title of thesis: Measurement library
Supervisor(s): Teemu Leppänen
Term and year when the thesis was submitted: Autumn 2022
Number of pages: 37

The aim of this thesis was to develop and implement a new measurement library based on current measurement script as part of the test environment of NR-radio's software. The measurement library should replace the current measurement script. The measurement library was implemented at Nokia in the Radio CI team.

The purpose of Radio CI is to test the software of different NR and LTE radios and to develop and automate test environments for radio software testing.

The result of the work was a measurement library which was clearer and easier to develop and update than its predecessor, it could be run using the Robot Framework test automation tool, and it supported the new base band unit.

Keywords:

Python, Robot Framework, Radio, RF, 5G, NR, test automation, CI, continuous integration

SISÄLLYS

SANASTO

4G	Neljännen sukupolven mobiiliverkko
5G	Viidennen sukupolven mobiiliverkko
ACP	Adjacent Channel Power, viereisen kanavan teho
ACLR	Adjacent Channel Leakage Ratio, viereisen kanavan vuotosuhde
agile	Ketterän sovelluskehityksen malli
API	Rajapinta
BLER	Block Error Rate, lähetettyjen ja vastaanotettujen pakettien virhesuhde
CI	Continuous Integration, Jatkuva integraatio
CSV	tiedostoformaatti
DL	Downlink. Radiolta lähtevä signaali
EVM	Error Vector Magnitude, virhevektorin suuruus
Git	Versionhallintatyökalu
IQ-data	Radiotaajuusdataa
Jenkins	Automaatiotyökalu/-palvelu
kovakoodaus	Muuttujan arvon kirjoittaminen lähdekoodiin, joka ei ole muutettavissa, syötettävissä tai muualta haettavissa
JSON	tiedostoformaatti
LTE	Neljännen sukupolven mobiiliverkko
NR-Radio	5G-tukiasemissa käytettävä radioyksikkö
pcap	Verkkoliikenteen sieppaus
PRACH	Physical Random Access Channel. Fyysinen hajasaantikanava
PUSCH	Physical UL Shared Channel. UL-suunnan datakanava
Python	Ohjelmointikieli
REST	Internetprotokolla
SA	Signaalianalysaattori
SG	Signaaligeneraattori

Robot Framework	Testiautomaatiotyökalu/-kehys
UL	Uplink, radiolle saapuva signaali
XML	tiedostoformaatti

1 JOHDANTO

1.1 Nokia ja Nokian Radio-CI

Nokia Oyj on suomalainen maailmanlaajuisesti toimiva tietoliikennealan yhtiö, jonka pääliiketoimintoja ovat verkkoinfrastruktuuri, sekä teknologiakehitys ja lisensointi (12). Opinnäytetyö toteutettiin Nokian Radio-CI-tiimissä. Radio-CI-tiimin tarkoitus on testata LTE- ja NR-radioiden ohjelmistopäivityksiä. Radio-CI-tiimi myös kehittää ja automatisoi omia testiympäristöjään sekä integroi muiden tiimien toteutuksia osakseen radio-ohjelmistojen testiympäristöjä.

Oulun Radio-CI-tiimi on Nokian yksi monista Radio-CI-tiimeistä. Nokialla on useita Radio-CI-tiimejä ympäri maailmaa. Jokaisella Radio-CI-tiimillä on omat testiympäristöt, mutta kyseiset tiimit kehittävät testi- ja testiautomaatioympäristöjään yhteistyössä toistensa kanssa. Oulun Radio-CI-tiimi tekee tiivistä yhteistyötä myös muiden Nokian Oulun verifiointi- ja integraatiotiimien kanssa.

1.2 Työn tavoitteet

NR-radiot ovat tukiasemissa käytettäviä laitteita, joiden avulla mobiililaitteet yhdistyvät 5G-matkapuhelinverkkoon. Radiossa on kiinni antenneja, joiden kautta data kulkee puhelimelta radiolle ja radiolta puhelimelle. LTE-radiot puolestaan yhdistävät puhelimet 4G-matkapuhelinverkkoon. Osa radiomalleista tukevat myös LTE:tä ja NR:ää.

Tämän opinnäytetyön tavoitteena oli luoda vanhan mittausskriptin pohjalta uusi mittauskirjasto NR-radioiden ohjelmistojen testaamista varten. Uutta mittauskirjastoa voidaan ajaa Robot Framework -testiautomaatiotyökalulla. Uusi mittauskirjasto on edeltävää mittausskriptiä selkeämpi ja helpommin ylläpidettävä, paremmin parametrisoitava sekä se tukee uutta kantataajuusyksikköä.

1.3 Opinnäytetyön rakenne

Luvussa 2 esitetään työn teknistä taustaa, luvussa 3 kerrotaan hieman Radio-CI:n testiympäristöistä, radiolle tehtävistä mittauksista sekä siihen käytettävästä mittausskriptistä, jonka pohjalta

uusi mittauskirjasto toteutettiin. Luvussa 4 käydään sitten läpi itse työn eri vaiheet, luvussa 5 kerrotaan uuden mittauskirjaston testauksesta ja käyttöönotosta ja luvussa 6 pohditaan työn tulosta.

2 OPINNÄYTETYÖN TAUSTA

2.1 Opinnäytetyön aihe

Opinnäytetyön tarkoituksena on luoda uusi Robot Frameworkilla käytettävä mittauskirjasto, jota voidaan käyttää NR-radioiden testaamiseen. Mittauskirjasto tulee tukemaan myös LTE-mittauksia, mutta tämä opinnäytetyö ei enää kata tätä osa-aluetta. Mittauskirjasto tulee korvaamaan Radio-CI-tiimin aikaisemmin mittaukseen käytetyn Python-skriptin. Mittauskirjasto tehdään vanhan mittaukseen käytettävän Python-skriptin pohjalta.

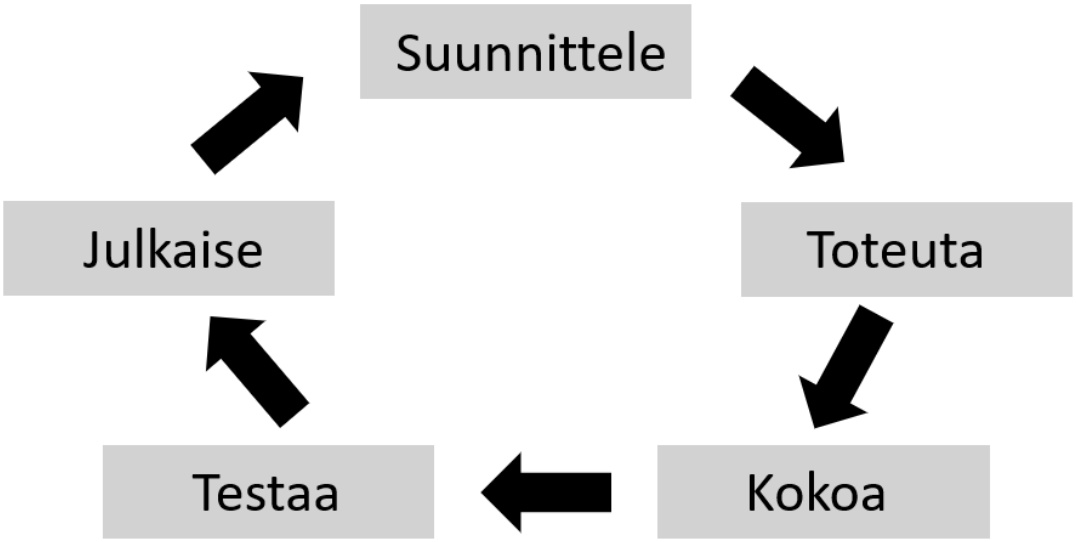
Alun perin testejä on ajettu Jenkkisissä ns. hybridi mallilla mikä tarkoittaa, että testiajot ajetaan osaksi käyttäen Robot Framework -testiautomaatiotyökalua ja osaksi suoraan Pythonilla. Alun perin mittaukseen käytettyä skriptiä on ajettu suoraan Pythonia käyttäen, mutta uutta mittauskirjastoa tulee voida ajaa täysin Robot Frameworkilla tehdyllä testilinjalla, joka sisältää mittauksen lisäksi muitakin testitapauksia, koska Radio-CI-tiimi on siirtymässä vanhasta hybridimallista täysin Robot Frameworkilla ajettaviin testilinjoihin. Täysin Robot Frameworkilla ajettavalta testilinja mahdollistaa myös paremman sekä laajemman tulosten raportoinnin palveluille, mistä radio-ohjelmistojen testiajoja voidaan seurata.

Vanha mittausskripti siirretään omaan Git-projektiin ja sen pohjalta aletaan luomaan uutta mittauskirjastoa. Mittauskirjaston tulee olla myös paljon selkeämpi ja helpommin ylläpidettävä kuin vanha mittausskripti, sekä sen tulee myös tukea uutta kantataajuusyksikköä.

2.2 Jatkuva integraatio

CI on lyhenne sanoista continuous integration, joka tarkoittaa suomeksi jatkuvaa integraatiota. Jatkuva integraatio on ketterä projektinhallintamenetelmä. Ketterä projektinhallintamenetelmä (engl. agile) perustuu tiimityöhön, jaoteltuihin tehtäviin sekä muutostilanteissa joustamiseen. Jatkuvan integraation ketterässä projektinhallintamenetelmässä uusia toiminnallisuuksia ja päivityksiä liitetään osaksi toimivaa koodia nopealla syklillä. Jatkuvan integraation mallissa hyödynnetään hajautettuja versionhallintatyökaluja, kuten esimerkiksi Git. Myös testaus on osa jatkuvaa integraatiota. Testaus vaiheessa päivitetään viimeisimmät versiot toteutuksista ja testataan niiden

toimivuus hyödyntäen jotain testiautomaatiotyökalua, kuten Jenkinsiä. Testauksen jälkeen tehdystä toteutuksesta luodaan julkaisu, mikäli testauksen yhteydessä ei havaittu ongelmia. Mikäli testauksen aikana havaitaan ohjelmia, koodiin toteutetaan virheenkorjauksia, ja ajetaan testit uudelleen. Vanhojen versioiden säilyttäminen on tärkeää jatkuvassa integraatiossa, koska mikäli koodiin jää jokin virhe, jota ei testausvaiheessa havaita, ohjelmisto voidaan palauttaa väliaikaisesti vanhaan versioon tarpeen vaatiessa. (Kuva 1.)



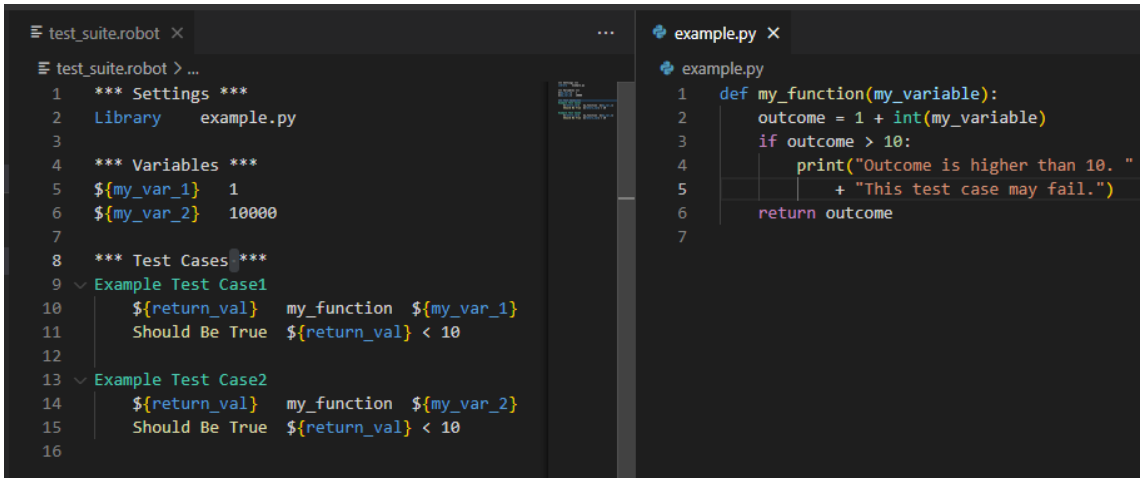
KUVA 1. Jatkuva integraatio.

2.3 Robot Framework

Robot Framework on avainsanapohjainen testiautomaatiotyökalu. Robot Frameworkilla voidaan kirjoittaa testilinjoja (engl. test suite), jotka sisältävät eri testitapauksia (engl. test case). Testitapaukset koostuvat avainsanoista, joilla voidaan esimerkiksi kutsua Python-funktioita.

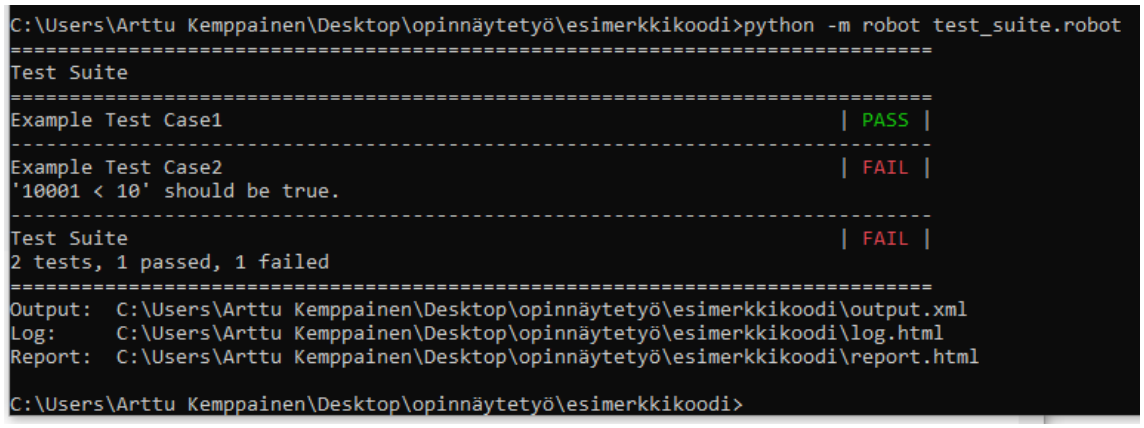
Robot Frameworkin vahvuus on sen helposti kirjoitettava ja selkeä syntaksi. Robot Framework luo ajon päätteeksi log.html ja report.html-tiedostot, joista testilinjan tapahtumat ovat helposti luettavissa. Robot Framework luo ajon pääteeksi myös output.xml-tiedoston, josta voidaan parsia esimerkiksi testituloksia. Lähteissä video, missä esitellään Robot Framework -testiautomaatiotyökalua (4).

Kuvassa 2 vasemmalla näkyy lyhyt esimerkki kirjotusta Robot Framework -testistä ja oikealla lyhyt Python-esimerkki, mitä vasemmalla oleva Robot Framework -testi kutsuu. Vasemmalla olevassa Robot Framework -testissä alkuun tuodaan oikealla olevan Python-moduulin funktiot *** Settings *** -rivin alapuolella. Tämän jälkeen alustetaan kaksi muuttujaa. Python-moduulin tuonnin ja muuttujien alustuksen jälkeen *** Test Cases *** -rivin alapuolelle on kirjoitettu kaksi ”testitapausta”. Kyseisessä testitapauksessa kutsutaan oikealla näkyvää funktiota. Funktio kasvattaa sille annettua arvoa yhdellä ja palauttaa sen. Lopuksi ”testitapaus” tarkastaa, että palautettu arvo on alle 10. (Kuva 2.)



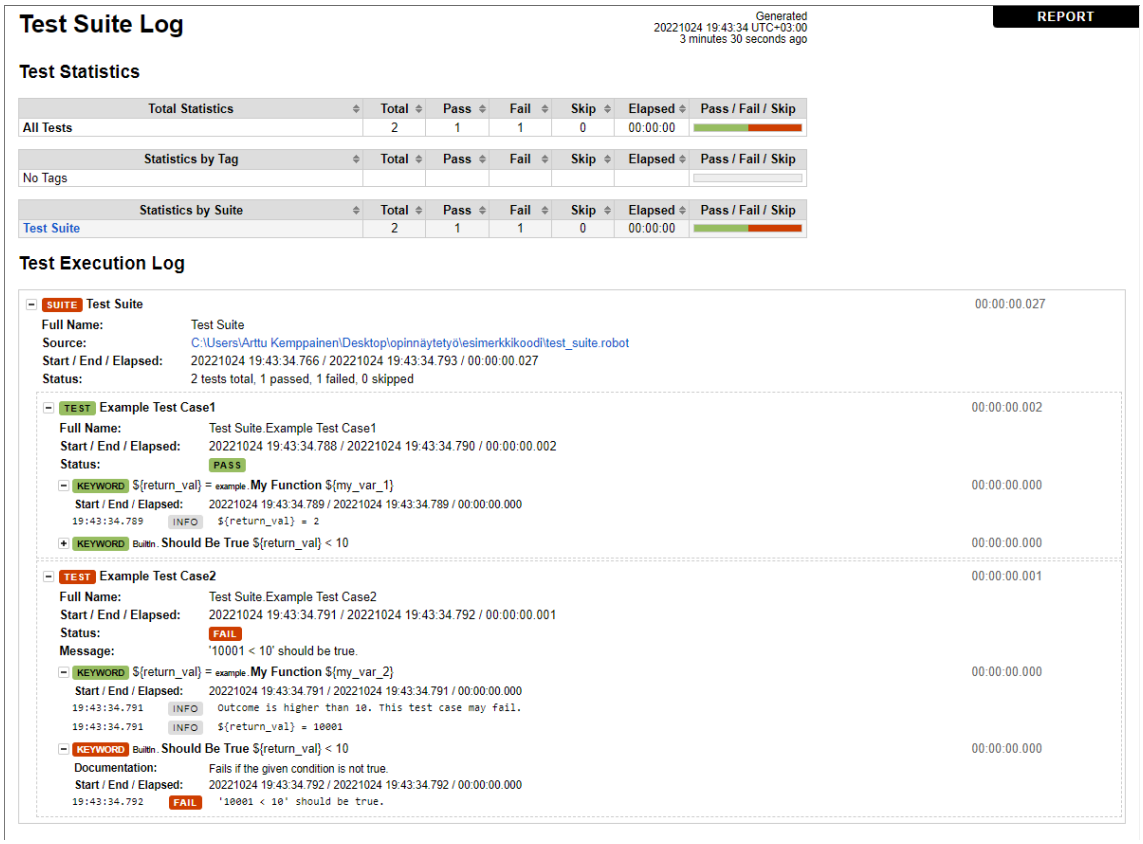
Kuva 2. Lyhyt sekä yksinkertainen esimerkki Robot Frameworkilla tehdystä ”testitapauksesta”.

Kuvassa 3 näkyy Robot Framework -testitapauksen ajo komentokehotteella. Robot Framework -testejä ajaessa komentokehotteelle tulee näkyviin testiajon eri vaiheet ja onko testin eri vaiheet onnistuneet vai epäonnistuneet. Lopuksi Robot Framework kirjoittaa yhteen vedon testeistä ja sen luomat tulostiedostot. (Kuva 3.)



Kuva 3. Robot-tiedoston ajo komentorivillä.

Kuvassa 4 näkyy Robot Frameworkin luoman log.html-tiedoston suoritetusta testistä. Tässä tiedostossa on tarkempi kuvaus testien etenemisestä. Mikäli testi epäonnistuu, kyseinen tiedosto auttaa paljon virheiden syiden paikantamisessa.



Kuva 4. Robot Framework -testin ajosta saatu log.html-tiedosto.

2.4 Git

Git on ilmainen avoimeen lähdekoodiin perustuva helposti käytettävä versionhallintatyökalu, jota käytetään ohjelmistokehityksessä. Käyttäjät voivat luoda projektin Git-työkaluun, jonne voidaan varastoida koodia/tiedostoja. Git mahdollistaa saman koodin kehittämisen usean ihmisen toimesta saman aikaisesti.

2.5 Jenkins

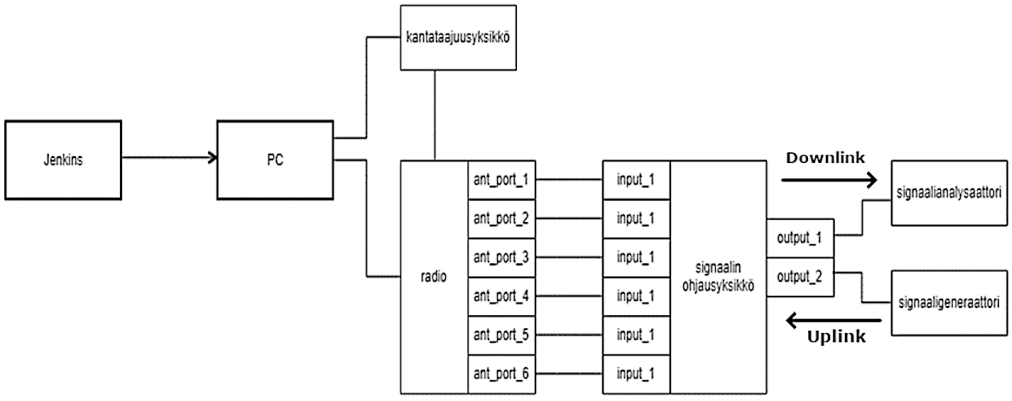
Jenkins on avoimeen lähdekoodiin perustuva Java-pohjainen automaatio- ja testiautomaatiotyökalu. Jenkinsissä voidaan luoda automaatioratkaisuja ja sitä voidaan käyttää esimerkiksi CI:ssä

automatisoimaan ympäristön rakennusta ja testausta. Nokian Oulun Radio-CI-tiimi käyttää pääosin Jenkinsiä radio-ohjelmistojen testaamiseen ja uusien kirjastoversioiden asennukseen testipaikoille.

3 NYKYTILANNE

3.1 Testiympäristöt

Nokian Oulun Radio-CI-tiimi ylläpitää testiympäristöjä ja testaa niillä radioiden ohjelmistopäivityksiä. Tyypilliseen testiympäristöön sisältyy PC, jolla testejä ajetaan radioille. PC on kiinnitettynä verkkokaapelilla kantataajuusyksikköön (engl. baseband unit) ja itse radioon, jolle testejä tehdään. Radiolta lähtee myös kaapeli suoraan kantataajuusyksikköön, jota pitkin kantataajuusyksikkö saa dataa radiolta. Radion antenniportteihin on kytkettynä signaaligeneraattori ja signaalianalysaattori, joita käyttäen radiolle voidaan tehdä mittauksia. DL-mittaukset tehdään antenniportille, johon on kytkettynä signaalianalysaattori ja UL-mittaukset tehdään antenniportille, jolle on kytkettynä signaaligeneraattori. Testipaikalla voi olla käytössä myös signaalin ohjausyksikkö, jolloin pystytään mittaamaan useampaa antenniporttia yhdellä signaaligeneraattorilla ja signaalianalysaattorilla. Oulun Radio-CI käyttää Jenkins-testiautomaatiotyökalua testien ajamiseen. (Kuva 5.)



Kuva 5. tyypillinen signaalin ohjausyksiköllinen radio-ohjelmiston testiympäristö yksinkertaistettuna.

Eri testilinjoilla käytetään mm. eri mallisia radioita ja kantataajuusyksiköitä. Kantataajuusyksikköä käytetään radion konfigurointiin. Kantataajuusyksiköltä voidaan lukea myös radiolta saatua dataa esimerkiksi osana UL-mittauksessa.

Eri testiympäristöissä ajetaan hieman erilaisia testejä. Joillakin testipaikoilla ajetaan vain NR-mittauksia, joissain vain LTE-mittauksia sekä joillain testipaikoilla samalle radiolle ja radio-

ohjelmistolle saatetaan ajaa sekä NR- että LTE-mittauksia. Joillain testiympäristöillä saatetaan mitata esimerkiksi kahta antenniporttia, kun toisilla mitataan kaikkia 64:ää radion antenniporttia, jolloin käytössä on useimmiten signaalinohjausyksikkö. Tässäkin voi olla radiokohtaisia eroavaisuuksia, koska eri radiomalleissa voi olla eri määrä antenniportteja.

3.2 Nykyinen mittausskripti

Mittausskriptin tehtävä on tehdä DL- ja UL-mittauksia. Ennen mittauksia radiolle pystytetään testimalli ja kantoaalto/kantoaallot mitattaviin antenniportteihin. Tämän jälkeen mittausskripti tekee mittaukset radiolle ja luo mittaustuloksista CSV- sekä JSON-tiedostot. Mittausskriptin luomat mittaustulostiedostot voidaan sitten lähettää palveluille, joista radio-ohjelmistoille tehtyjen testien tuloksia voidaan seurata.

3.2.1 Radiolta lähtevän signaalin mittaus

DL-mittauksessa lähetetään signaalia antenniportista, johon on kiinnitetty signaalianalysaattori. Mittausskripti konfiguroi mittauskirjastolle annettavan XML-konfiguraatiotiedoston mukaisesti signaalianalysaattorin lähettämällä REST-rajapinnan kautta oikeanlaiset asetukset signaalianalysaattorille. Tämän jälkeen signaalianalysaattori mittaa radiolta lähtevää signaalia. Tämän jälkeen mittausskripti hakee signaalianalysaattorilta mittaustuloksia signaalianalysaattorille tehdyn REST-rajapinnan kautta ja luo tuloksista CSV- ja JSON-tiedostot. DL-mittauksesta saatavat tulokset ovat kantoaallon teho (engl. carrier power), lähtöteho (engl. output power), virhevektorin suuruus (EVM) ja virheisen kanavan vuotosuhde (ACLR).

3.2.2 Radiolle tulevan signaalin mittaus

UL-mittauksissa puolestaan mitattavaan antenniporttiin on signaalianalysaattorin sijaan kiinnitetty signaaligeneraattori. Mittausskripti konfiguroi alkuun signaaligeneraattorin lähettämään haluttua signaalia lähettämällä XML-konfiguraatiotiedoston mukaiset parametrin signaaligeneraattorille REST-rajapinnan kautta. Tämän jälkeen mittausskripti ottaa yhteyden kantataajuusyksikölle ja lataa sieltä pcap-tiedoston. Tämä pcap-tiedoston data sitten analysoidaan ja analysoinnista saadaan mittauksen testitulokset. Näistä tuloksista mittausskripti tekee vielä lopuksi CSV- ja JSON-tiedostot.

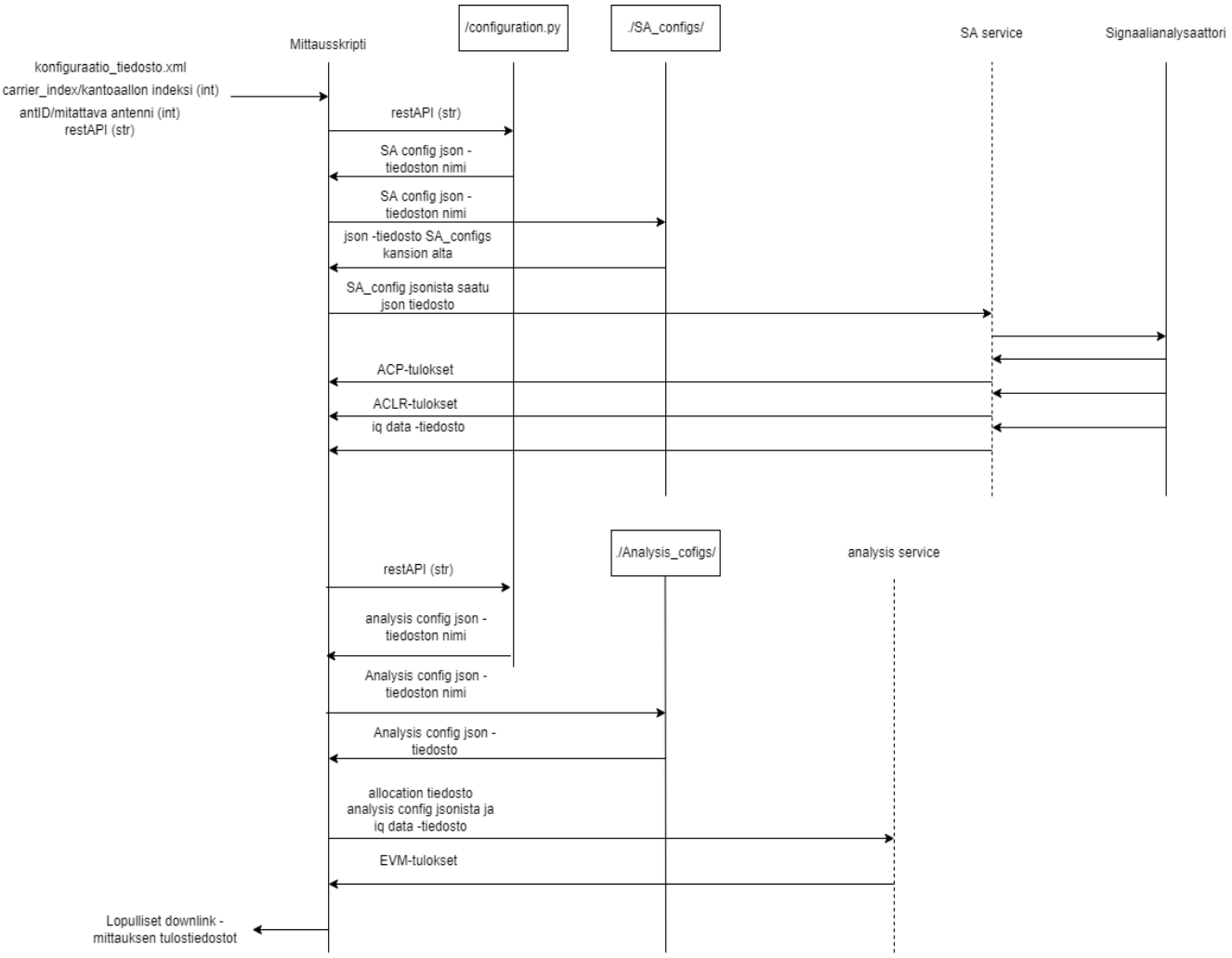
Mittausskriptillä voidaan tehdä kahdenlaisia UL-mittauksia, PUSCH- ja PRACH-mittauksia. PUSCH-mittauksissa mitataan tulevan signaalin suunnan datakanavaa. UL-suunnan PUSCH-mittauksista saadaan tuloksena BLER, EVM, UL-suunnan teho ja ajoituksen sekä taajuuden poikkeama.

PRACH-mittauksissa puolestaan UL-suunnan hajasaantia. PRACH-mittauksesta saadaan tuloksena EVM, UL-suunnan teho sekä ajoituksen ja taajuuden poikkeama. PRACH-mittauksessa ei lasketa BLER-tulosta toisinkuin PUSCH-mittauksessa.

3.3 Mittausskriptin arkkitehtuuri

3.3.1 Radiolta lähtevän signaalin mittaus

Mittausskriptiä käytettiin ajamalla sitä komentokehotteella, josta voitiin ajaa mittauksen eri vaiheet parametrin avulla. DL-mittaus voitiin ajaa yhdellä komennolla antamalla sille mittaukseen käytettävät parametrit, eli RestAPI-merkkijono, jonka perusteella mm. haettiin haluttu JSON-tiedosto SA_configs-kansion alta, joka sisältää tarvittavia parametrejä signaalianalysaattorin konfigurointiin, kantoaallon indeksi, jolla määritellään mitattava kantataajuus, mitattava antenni ja XML-konfiguraatiotiedosto. Oikean SA-konfiguraatiotiedoston hakemisen jälkeen signaalianalysaattorille lähetettiin tarvittavat parametrit SA Servicen kautta signaalin mittausta varten. Tämän jälkeen SA Service palautti saadut ACLR- ja ACP-mittaustulokset sekä IQ-data tiedoston signaalianalysaattorilta. IQ-datan analysointia varten haettiin JSON-tiedosto Analysis_configs-kansion alta käyttämällä RestAPI-parametria, joka sisälsi oikean allokaationtiedoston nimen. Tämän jälkeen mittauksesta saatu IQ-data-tiedosto ja allokaatiotiedoston nimi lähetettiin analysis_servicelle, josta saatiin mittauksen EVM-tulos. Lopuksi saaduista ACP-, ACLR- ja EVM-tuloksista kasattiin tulostiedostot tehdystä mittauksesta. (Kuva 6.)

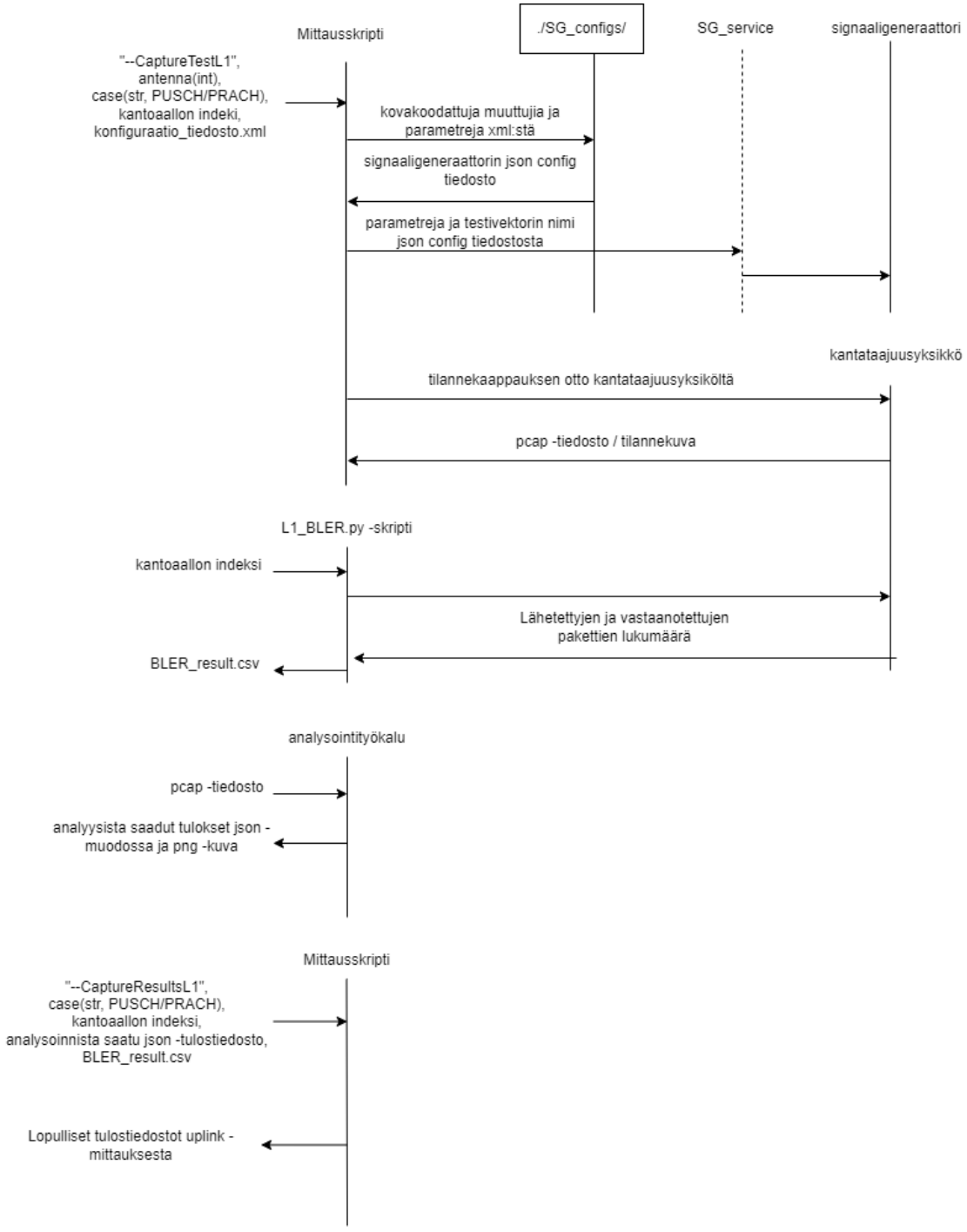


Kuva 6. Yksinkertaistettu kuva mittausskriptillä suoritettavasta DL-mittauksesta.

3.3.2 Radiolle tulevan signaalin mittaus

UL-mittauksen suorittamiseen jouduttiin puolestaan antamaan useampi komento komentoriviltä mittauksen suorittamiseen. Alkuun mittausskripti ajettiin antamalla sille parametrinä –CaptureTestL1, mitattava antenni kokonaislukuna, PUSCH tai PRACH riippuen kumpi mittaus haluttiin suorittaa, mitattavan kantoaallon indeksi ja XML-konfiguraatiotiedosto. Tämän jälkeen annetun XML-konfiguraatiotiedoston perusteella haettiin oikea JSON-tiedosto SG_configs-kansion alta, joka sisälsi tarvittavia parametrejä signaaligeneraattorin konfigurointia varten. Signaaligeneraattorin konfiguroinnin jälkeen mittausskripti otti tilannekuvan kantataajuusyksiköltä pcap-tiedoston muodossa. Tilannekuvan oton jälkeen ajettiin L1_BLER.py-skripti, jolla mitataan lähetettyjen ja vastaanotettujen lohkojen/pakettien lukumäärää. Lopuksi BLER-laskentaan käytetty skripti laski lähetettyjen ja vastaanotettujen pakettien suhteen ja loi tulostiedoston, jossa lukee kuinka monta

prosenttia lähetetyistä paketeista ei saatu vastaanotettua. BLER laskennan jälkeen ajettiin analysointityökalu, jolle annettiin parametrinä aikaisemmasta tilannekuvan kaappauksesta saatu pcap-tiedosto. Tästä analysoinnista saatiin ulos JSON-tiedosto, joka sisälsi UL-mittaustulokset. Lopuksi vielä mittausskripti ajettiin antamalla sille parametrinä –CaptureResultsL1, PUSCH tai PRACH, mitatun kantoaallon indeksi, analysoinnista saatu tulostiedosto ja BLER-laskennasta luotu tiedosto. Näistä parametreistä mittauskirjasto loi lopulliset mittaustulostiedostot. (Kuva 7.)



Kuva 7. Yksinkertaistettu kuva mittausskriptillä suoritettavasta UL-mittauksesta.

4 UUSI MITTAUSKIRJASTO

4.1 Työn suunnittelu ja aloitus

Työ aloitettiin tekemällä uusi Measurement-niminen projekti Git-versionhallintatyökaluun. Kyseiseen projektiin siirrettiin vanha mittausskripti eräästä toisesta projektista. Kyseinen mittausskripti oli yli 4000 riviä pitkä, joten seuraava askel oli ruveta suunnittelemaan, miten skripti olisi järkevintä jaotella useampaan pienempään Python-moduuliin ja luokkaan.

Vanhalle mittausskriptille annettiin parametrinä XML-konfiguraatiotiedosto, mutta mittausskripti sisälsi kuitenkin paljon kovakoodattuja muuttujia, joita tarvitsi joskus käsin muokata, kun haluttiin muuttaa mittausten konfigurointia ja asetuksia.

Yksi päätavoitteista oli myös päästä eroon näistä kovakoodatuista parametreistä ja muuttujista sekä hyödyntää parametrinä annettavaa XML-konfiguraatiotiedostoa mahdollisimman paljon.

Mittauskirjastoa tuli voida käyttää myös Robot Frameworkin avulla. Tätä varten mittauskirjastolle tuli luoda Robot Framework -rajapintatiedosto, joka sisältää avainsanoja, joiden avulla mittaukset voidaan suorittaa. Mittauskirjastoa Robot Frameworkin kautta ajaessa CSV- ja JSON-tiedostojen lisäksi Robot Framework luo myös output.xml- sekä output.json-tiedostot testiajosta. Nämä Robot Frameworkin luomat tiedostot mahdollistavat testitulosten raportoinnin oikeassa formaatissa palvelulle, josta testituloksia voidaan seurata eri radio-ohjelmistoille.

TAULUKKO 1. Lista mittauskirjastolle annetuista vaatimuksista suunnittelu vaiheessa

	Mittauskirjaston vaatimukset
1.	Mittausskripti tulee pilkkoa useaan Python-moduuliin ja koodin tulee olla selkeämpi
2.	Mittauskirjasto tulee olla paremmin parametrisoitava eri testitapauksille ja -ympäristöille
3.	Mittauskirjastolla tulee voida tehdä UL-mittauksia uudemmalla ja vanhemmalla kantataajuusyksiköllä
4.	Kaikki yhteyden otot kantataajuusyksikölle tulee ulkoistaa eri Python-kirjastolle
5.	Mittauskirjastoa tulee voida käyttää Robot Frameworkilla
6.	Mittauskirjastolla tulee voida tehdä yhden ja kahden kantoaallon DL-mittauksia

7.	Mittauskirjastolla tulee voida tehdä yhden ja kahden kantoaallon UL-mittauksia

4.2 Mittausskriptin pilkkominen ja siivoaminen

Kun mittausskripti oli siirretty omaan Git-projektiin, se oli tarkoitus pilkkoa useampaan Python-moduuliin/-skriptiin. Vanha mittausskripti oli yli 4000 riviä pitkä, joka oli yksi suurimmista syistä, miksi sen ylläpidettävyys oli vaikeaa.

Tämän jälkeen mittausskripti jaoteltiin kahteen erilliseen skriptiin. Toiseen skriptiin siirrettiin DL-mittauksen toiminnallisuudet, sekä toiseen skriptiin siirrettiin UL-mittauksen toiminnallisuudet.

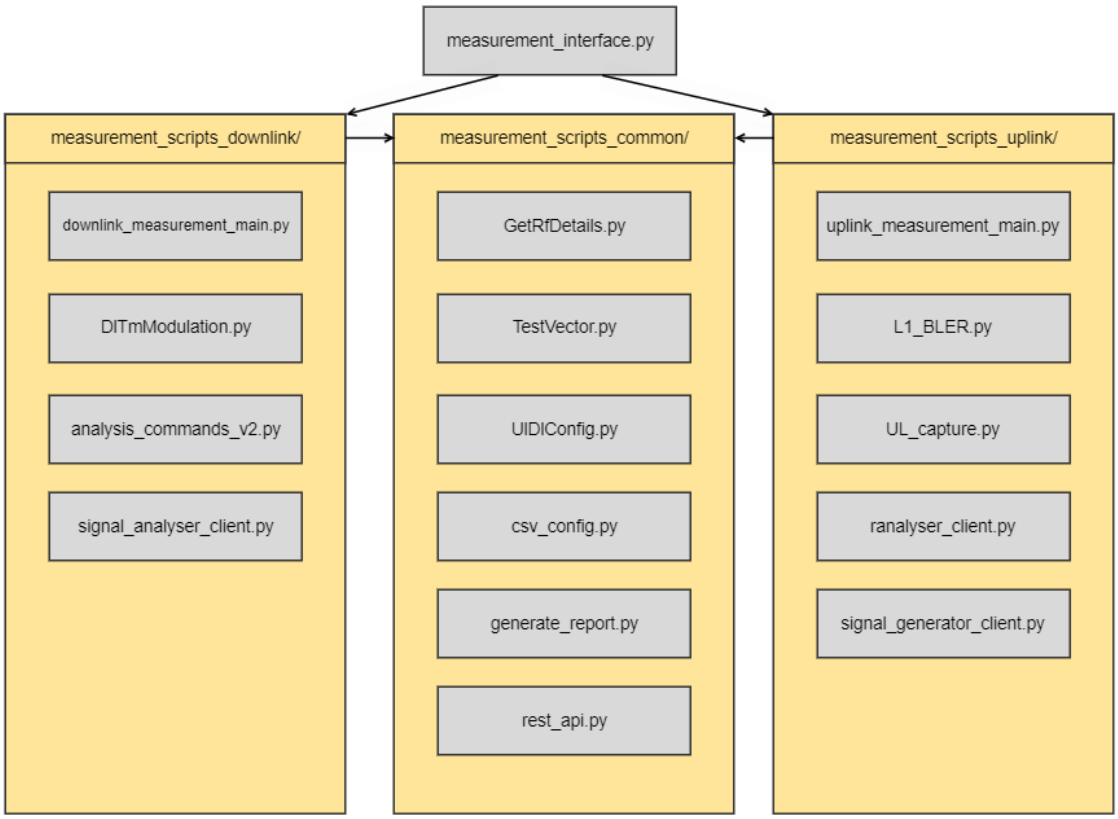
Vanhassa mittausskriptissä oli satoja rivejä pitkiä funktioita, jotka olivat vaikeasti ymmärrettäviä, koska sama funktio saattoi sisältää toiminnallisuuksia signaaligeneraattorin ohjauksesta kantataajuusyksikön kaappauksen suorittamiseen. Kaikki funktiot käytiin läpi ja jaoteltiin eri toiminnallisuudet omiin luokkiin ja funktioihin. Näin DL- ja UL-mittauksen skripteistä saatiin paljon selkeämpiä.

Seuraavaksi signaaligeneraattorin ohjaukseen viittaavat toiminnallisuudet siirrettiin signal_generator_client-nimiseen Python-moduuliin ja signaalianalysaattorin ohjaukseen liittyvät toiminnallisuudet siirrettiin signal_analyzer_client-nimiseen Python-moduuliin.

Mittauskirjaston UL-mittaukseen käytettävästä skriptistä poistettiin kaikki muu, paitsi signal_generator_client.py:n käyttö, jolla ohjattiin signaaligeneraattoria. Tähän oli syynä se, että pcap-tiedoston lataaminen kantataajuusyksiköltä sekä sen analysointi ulkoistettiin muille kirjastoille myöhemmin.

Mittauskirjaston yhteydenotot radiolle myös ulkoistettiin eräälle toiselle Radio-CI-tiimin toteuttamalle Python-kirjastolle. Mitä enemmän käytetään jo toteutettuja ratkaisuja, sitä vähemmän on ylläpidettävää koodia.

Mittausskripti myös pilkottiin omiin Python-moduuleihin, jotka on jaoteltu kolmeen eri kansioon. Kansioon measurement_scripts_downlink siirrettiin DL-mittaamiseen käytetyt tiedostot, measurement_scripts_uplink-kansioon siirrettiin UL-mittaamiseen käytettävät tiedostot ja measurement_scripts_common-kansioon siirrettiin tiedostot, joita käytettiin DL- sekä UL-mittauksiin. Tämän lisäksi tehtiin myös measurement_interface.py, jota kutsumalla voidaan ajaa DL- ja UL-mittaukset. (Kuva 8.)



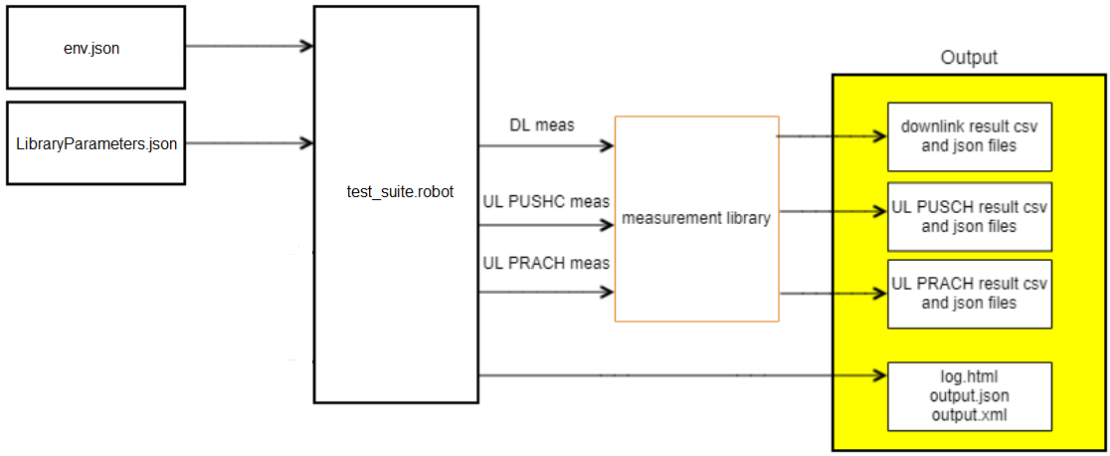
Kuva 8. Mittausskripti pilkkomisen jälkeen

4.3 Parametrisointi

Vanha mittausskripti sisälsi paljon testien konfigurointi ja ympäristö kohtaisia muuttujia, joita jouduttiin välillä muuttamaan käsin. Tämä teki vanhasta mittausskriptistä vaikeasti ylläpidettävän, koska mikä tahansa arvo kuten, esimerkiksi mittalaitteen herätyksen viiveen pituus saattaa muuttua testattavien laitteiden tai ohjelmistojen muuttuessa. Kaikista kovakoodatuista muuttujista oli siis päästävä eroon uudessa mittauskirjastossa.

Suurimman osan testitapauskohtaisista muuttujista oli mahdollista saada mittauskirjastolle syötettävästä XML-konfiguraatiotiedostosta. Vanha mittausskripti käytti jo valmiiksi samanlaista XML-konfiguraatiotiedostoa, mutta mittauskirjaston tuli hyödyntää tätä tiedostoa tehokkaammin, jotta mittauskirjaston ylläpidettävyys olisi helpompaa.

Kaikkia muuttujia ei kuitenkaan ollut mahdollista saada kyseisestä XML-konfiguraatiotiedostosta. Suurin osa tämmöisistä muuttujista oli testitapauskohtaisien muuttujien sijasta testiympäristökohtaisia muuttujia. Näitä muuttujia varten luotiin oma env.json-tiedosto, jonne lisättiin kyseiset muuttujat. Kyseinen env.json-tiedosto annetaan mittauskirjastolle parametrinä XML-konfiguraatiotiedoston tapaisesti, josta mittauskirjasto osaa hakea testiympäristökohtaiset muuttujat. Kyseisiä testiympäristökohtaisia muuttujia olivat esimerkiksi mittalaitteen sekä signaaligeneraattorin IP-osoite, kansion polku jonne testitulostiedostot kirjoitetaan jne. Nämä parametrisointimuutokset helpottavat huomattavasti mittauskirjaston käyttöä. (Kuva 9.)



Kuva 9. Mittauskirjaston parametrisointi.

4.4 Lähtevän signaalin mittaus

UL-mittausskriptiin oli jätetty vain signaaligeneraattorin ohjaus. Tämän lisäksi UL-toteutukseen piti lisätä antennin tilannekuvan kaappaaminen kantataajuusyksiköltä, pcap-tiedoston generointi saadusta tilannekuvasta, pcapin analysointi ja mittaustulostiedostojen luominen analysoidusta pcap-tiedostosta.

Mittausskriptistä oli otettu vanha UL toiminnallisuus pois, joka kaappasi pcapin kantataajuusyksiköltä. Tämä pcap-tiedoston kaappaustoiminnallisuus ei toiminut uudemmalla kantataajuusyksiköllä, koska vanha kantataajuusyksikkö käytti IPv4:ää ja uusi kantataajuusyksikkö käytti IPv6:ta. Kantataajuusyksikön ohjaukseen liittyviä toiminnallisuuksia ei muutenkaan saanut olla mittauskirjastossa, koska mittauskirjasto tuli olla riippumaton käytettävästä kantataajuusyksiköstä, joten kaappauksen ottaminen kantataajuusyksiköltä tuli ulkoistaa eri Python-kirjastolle

Pcap-tiedoston kaappaukseen käytettiin siis toista Nokian sisäistä Python-kirjastoa, jota kutsutaan mittauskirjastosta. Tämä kirjasto ei kuitenkaan tukenut pcap-tiedoston ottoa suoraan uudelta kantataajuusyksiköltä, vaan sillä sai tehtyä antennin tilannekuvan ja ladattua sen tietokoneelle. Antennin tilannekuvasta voitiin tämän jälkeen luoda pcap-tiedosto toista Nokian sisäistä työkalua kutsumalla mittauskirjastolta. Tässä vaiheessa UL-mittauksessa siis konfiguroitiin signaaligeneraattori, otettiin tilannekuva antennilta kantataajuusyksikön kautta ja ladattiin se tietokoneelle sekä luotiin pcap-tiedosto saadusta tilannekuvasta.

Seuraavaksi mittauskirjastolle piti lisätä BLER-mittaus, joka on osa UL-mittausta. BLER-mittauksessa mitataan virheellisesti vastaanotettujen lohkopakettien määrää. BLER mitataan ottamalla ensin yhteys kantataajuusyksikölle, josta voidaan lukea rekisteriloki, josta voidaan lukea vastaanotettujen, vastaanottamattomien sekä tunnistamattomien pakettien määrä solu/kantoaalto kohtaisesti. Näistä voidaan sitten laskea BLER-tulos prosentteina. Jos BLER-tulos on 0 %, silloin kaikki lohkopaketit on vastaanotettu onnistuneesti.

BLER-mittaukseen sekä laskentaan oli olemassa jo valmis Python-skripti, mutta se ei tukenut uutta kantataajuusyksikköä. Kyseinen BLER-Python-skripti siirrettiin osaksi mittauskirjastoa, ja siihen tehtiin lisäyksiä, jotta se saatiin toimimaan myös uudella kantataajuusyksiköllä. Skriptiin lisättiin toiminto, joka ottaa yhteyden kantataajuusyksikölle IPv6:ta IPv4:n sijaan, mikäli käytössä on uudempi kantataajuusyksikkö. Tämän jälkeen kantataajuusyksikön rekisterilokin parsintaan ja BLER-laskentaan tehtiin muutoksia, koska rekisteriloki eroaa hieman vanhan ja uuden kantataajuusyksikön välillä.

Seuraava vaihe oli lisätä pcap-tiedoston analysointi. Tähän myös käytettiin erästä Nokian sisäistä Python-kirjastoa, jota kutsuttiin mittauskirjastolta. Tästä saatiin ulostulona CSV-tiedosto, jossa oli mittauksen tulokset. Kyseinen CSV-tiedosto ei ollut kuitenkaan halutussa formaatissa, jota voidaan raportoida eteenpäin. Tätä varten tehtiin muokkauksia mittauskirjaston Python-skriptiin, joka luo

CSV- ja JSON-tulostiedostot. Python-skriptin muokkausten jälkeen sillä pystyttiin hakemaan pcap-tiedoston analysoinnista saadusta CSV-tiedostosta halutut kentät sekä BLER-mittauksesta saatu tulos ja näistä luotiin CSV- ja JSON-tiedostot, jotka voitiin raportoida mittausten jälkeen palveluun, josta voidaan seurata radio-ohjelmistolle tehtyjen testien tuloksia.

4.5 Vanhan kantataajuusyksikön tuki

Mittauskirjastolle tuli myös toteuttaa tuki vanhalle kantataajuusyksikölle. Mittauskirjaston DL-mittaustoiminnallisuuksien puolelle ei tarvinnut tehdä mitään muutoksia, koska DL-mittauksessa ei operoida kantataajuusyksikköä, vaan testitulokset saadaan mittalaitteelta, joka on kiinnitettynä antenniporttiin, josta DL-mittaus suoritetaan. UL-mittauksessa mitattavaan antenniporttiin on kiinnitetty signaaligeneraattori, joka lähettää haluttua signaalia antenniportille, jonka jälkeen kantataajuusyksikölle otetaan yhteyttä, josta saadaan otettua tilannekuva mitattavasta antenniportista.

UL-mittaus ei tässä vaiheessa vielä tukenut vanhempaa kantataajuusyksikköä, joten aloitettiin ottamaan selvää, miten se saataisiin toimimaan myös vanhemmalla kantataajuusyksiköllä. Signaaligeneraattorin ohjaukseen liittyen ei tarvinnut tehdä muutoksia, koska siinä ei kommunikoida vielä kantataajuusyksikön kanssa, mutta tämän jälkeen lokitiedostojen ja antennikanavan tilannekuvan otto ei onnistunut vanhalla tavalla, koska vanhempi kantataajuusyksikkö toimii hiukan eri tavalla kuin uusi.

Pcap-tiedoston kaappauksen ottoon löydettiin toimiva funktio myös vanhalle kantataajuusyksikölle samasta Python-kirjastosta, jota oli käytetty myös uudemman kantataajuusyksikön pcapin kaappauksen toteuttamiseen.

Siinä missä uudella kantataajuusyksiköllä saatiin haettua antennin tilannekuva ja siitä pcap luotua siihen tarkoitetulla työkalulla, vanhalta kantataajuusyksiköltä saatiin otettua suoraan pcap siihen käytetyllä Python-kirjastolla. Täten vanhalle kantataajuusyksikölle kaappauksen ottaminen oli yksinkertaisempaa, kun uudemmalle kantataajuusyksikölle.

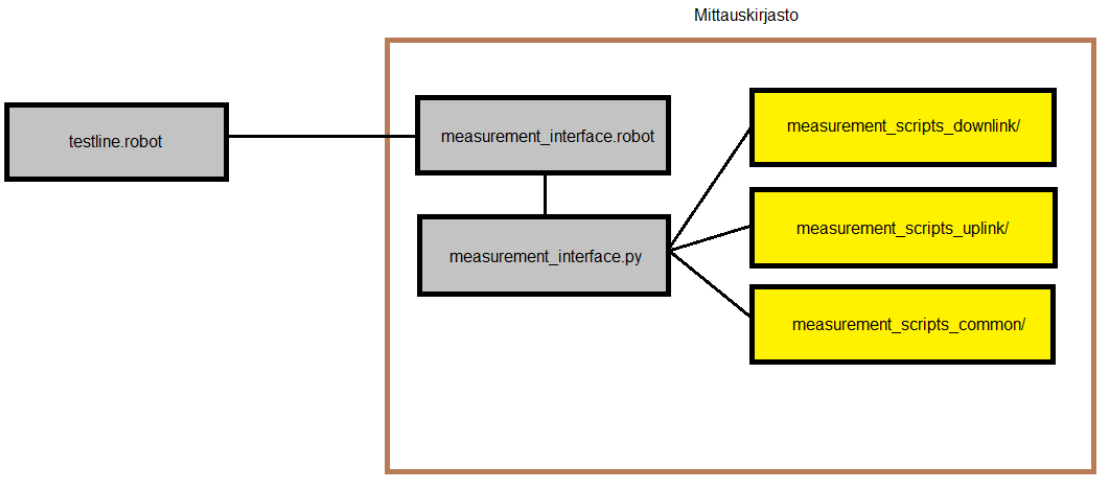
Tämän jälkeen, kun pcap oli saatu haettua suoraan kantataajuusyksikölle, sen analysointi ja tulosten muodostaminen voitiin tehdä samalla tavalla kuin uudemmallekin kantataajuusyksikölle.

4.6 Robot Framework -tuki

Mittauskirjastolle tehtiin Robot Framework -rajapinta, jotta mittauskirjasto voitiin ottaa osaksi Robot Framework -testilinjaa. Mittauskirjaston Robot Frameworkilla kutsuttava rajapinta oli robot-tiedosto, joka sisälsi Robot Framework -avainsanoja, joita kutsumalla Robot Framework testilinjalta voitiin ajaa mittauksia radiolle. Näin mittauskirjasto voitiin ottaa osaksi laajempaa Robot Framework testilinjaa (Kuva 10.). Mittauskirjaston käyttöönotto osana Robot Framework -testilinjaa oli erityisen tärkeää, koska Robot Framework -testilinjalta saadaan tulostiedosto koko testilinjan ajosta, jota tarvittiin tulosten raportointiin palveluun, josta testituloksia pystyttiin seuraamaan eri radio-ohjelmistoille korkeammalta tasolta.

DL-mittauksen suorittamiseen luotiin yksi avainsana, jota voitiin käyttää DL-mittauksiin riippumatta siitä, mikä radio tai kantataajuusyksikkö oli käytössä. UL-mittauksen suorittamiseen jouduttiin kuitenkin luomaan useampi avainsana, joita kutsumalla voitiin tehdä mittaukset. Syynä tälle oli eroavaisuuden tilannekaappauksen ottamisessa vanhemmissa ja uudemmissa kantataajuusyksiköissä. Esimerkiksi vanhempi kantataajuusyksikkö käytti IPv4-protokollaa, kun taas uudempi kantataajuusyksikkö käytti IPv6-protokollaa. Myös UL-mittauksissa BLER-mittaus laitettiin omaan avainsanaan, koska UL PRACH -mittauksissa ei mitata BLER-tulosta, kun taas UL PUSCH -mittauksessa mitataan.

Loppujen lopuksi UL-mittauksille oli avainsanat signaaligeneraattorin konfigurointiin, BLER-mittauksiin, tilannekaappauksen ottaminen uudelle tai vanhalle kantataajuusyksikölle, tilannekaappauksen analysointi, sekä analysoinnista saatujen tulosten generoiminen. UL-mittaukseen käytettäville avainsanoille annettiin argumenttina testitapaus, jolla voitiin määritellä, että halutaanko ajaa PUSCH- tai PRACH-mittaus.

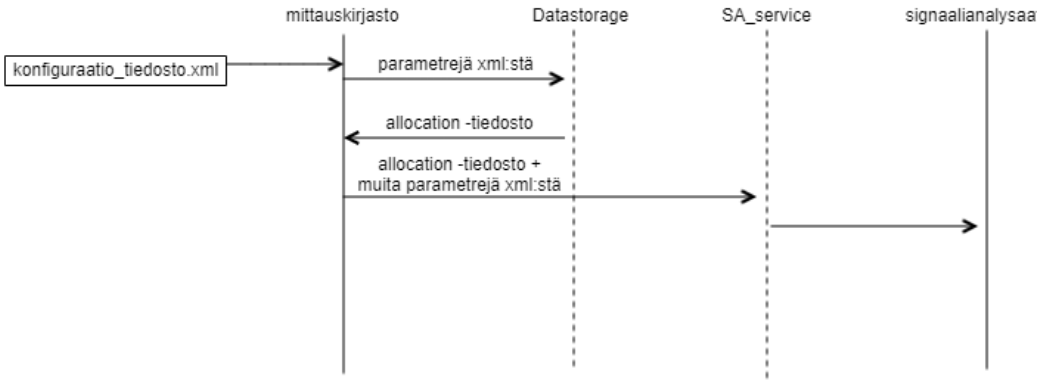


Kuva 10. Mittauskirjaston avainsanojen kutsuminen Robot Framework -testistä

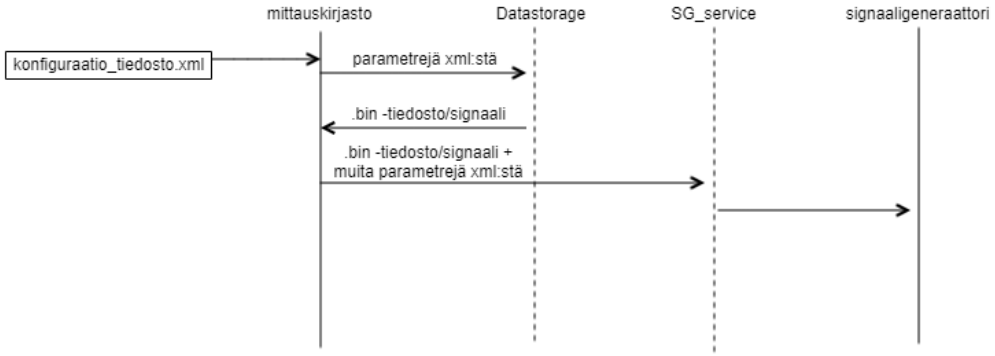
4.7 Testivektorin ja allokaatiotiedoston haku Datastoragesta

DL-mittauksessa signaalianalysaattorilta saadun kaappauksen analysointiin käytetään allokaatiotiedostoa ja UL-mittauksessa signaaligeneraattorille annetaan testivektori binaaritiedostona. Aikaisemmin mittauskirjastossa on käytetty erillistä JSON-konfiguraatiotiedostoa näiden binaari- ja allokaatiotiedostojen käsittelyyn, jossa oli listattu tiedostojen nimet. Mittauskirjastolle annettiin käsin konfiguraatioindeksi argumenttina, jonka perusteella haettiin halutut binaari- ja allokaatiotiedostot. Tämän jälkeen tiedostojen nimet annettiin parametrinä signaaligeneraattorille tai -analysaattorille niiden ohjauksen yhteydessä, jonka jälkeen signaalianalysaattorirajapinta tai signaaligeneraattorirajapinta haki nimen perusteella tiedostot paikallisesti allokaatio- tai binaaritiedostoja sisältävästä kansiosta.

Tästä käsin annettavasta konfiguraatioindeksiparametrista ja JSON-tiedostosta, joka sisälsi eri allokaatio- ja binaaritiedostonimiä, haluttiin päästä eroon. Tässä voitiin hyödyntää Nokian sisäistä Datastorage-rajapintaa, josta voitiin hakea allokaatio- ja binaaritiedostojen nimet lähettämällä sille mittaukseen käytettäviä parametrejä, joita ovat mm. testimalli, signaali, kantataajuuden leveys jne. Kyseiset parametrit voidaan saada mittauskirjastolle annettavasta XML-konfiguraatiotiedostosta, joka sisältää suuren määrän radio-ohjelmiston testaamiseen käytettäviä parametrejä.
Näin mittauskirjastossa päästiin taas eroon yhdestä argumentista ja ylläpidettävästä JSON-tiedostosta, jolloin mittauskirjaston käyttö ja ylläpidettävyys helpottuu. (Kuva 11, Kuva 12.)



Kuva 11. allokaatiotiedoston hakeminen signaalianalysaattorille Datastoragesta DL-mittauksessa



kuva 12. Signaalin hakeminen signaaligeneraattorille Datastoragesta UL-mittauksessa

4.8 Kahden kantoaallon tuki radiolta lähtevän signaalin mittaukseen

Uutta mittauskirjastoa tehdessä oli myös huomioitu useamman kantoaallon mittaukset, mutta mittauskirjaston toimivuutta ei ollut vielä tähän mennessä varmennettu kuin yhdellä kantoaallolla.

Testatakseen kahden kantoaallon mittausta oli ensin konfiguroitava radio antamaan kahden kantoaallon signaalia antenniportista. Tämä tapahtui lisäämällä XML-konfigurointitiedostoon parametrit toiselle kantoaallolle. Tämän jälkeen radio konfiguroitiin ajamalla testimalli uudelleen ylös kyseisen XML-konfiguraatiotiedoston kanssa. Kyseessä on sama konfigurointitiedosto, joka annetaan myös mittauskirjastolle parametrina, kun ajetaan mittauksia.

Kahden kantoaallon mittausten kokeilussa huomattiin, että mittaukset itsessään toimivat, mutta tulosten varmennukseen sekä CSV-tulostiedostojen kirjoitukseen täytyi tehdä pieniä korjauksia, koska kahden kantoaallon mittauksesta tulee kahdet erilliset tulokset. Kahden kantoaallon mittauksissa täytyy huomioida kohinatulosten varmennuksessa vierekkäinen kantoaalto. Tässä toiminnallisuudessa huomattiin bugi, joka täytyi korjata. CSV-tulostiedostojen kirjoituksessa puolestaan täytyi tehdä korjaus, että mittauskirjasto osaa kirjoittaa tulostiedostoon tulokset kaikkien kantoaaltojen mittauksista.

4.9 Kahden kantoaallon tuki radiolle tulevan signaalin mittaukseen

Mittauskirjastolla ajettiin UL-mittaukset Robot Framework -testilinjalta kutsumalla mittauskirjaston avainsanoja, joilla konfiguroitiin signaaligeneraattori, kaapattiin pcap-tiedosto kantataajuusyksiköltä, analysoitiin vastaanotettu pcap tiedosto ja jolla luotiin CSV- ja JSON-mittaustulostiedostot analysoidusta pcap-tiedostosta.

Kyseisille avainsanoille annettiin argumenttina ID, jolla voitiin valita mitattava kantoaalto. Mittauskirjastolla siis voitiin jo tehdä mittauksia useammalle kantoaallolle, mutta ongelmana kyseisessä toteutuksessa oli, että esimerkiksi kahdella kantoaallolla näitä Robot Framework -avainsanoja jouduttiin erikseen kutsumaan kahteen kertaan, kun haluttiin mitata kumpikin kantoaalto. Robot Framework -testilinja tulisi olla samanlainen riippumatta käytettävistä konfiguroinneista.

Eri kantataajuusyksiköillä käytettiin myös eri metodia antennin tilannekuvan ja pcap-tiedoston ottamiseen, jolloin Robot Framework -testilinjan UL-mittaus jouduttiin suorittamaan hieman eri avainsanoja käyttäen riippuen siitä, mitä kantataajuusyksikköä käytettiin.

Mittauskirjastolle luotiin kaksi uutta Robot Framework -avainsanaa koko UL-mittauksen ajamiseen. Toinen avainsana oli vanhemmalle ja toinen oli uudemmalle kantataajuusyksikölle. Kyseiset avainsanat olivat yhdistelmä edellisistä avainsanoista UL-mittaukselle, sillä erotuksella, että mittauksessa oli silmukka (engl. loop), joka ajoi mittauksen jokaiselle kantoaallolle. Täten siis koko UL-mittaus voitiin suorittaa molemmalle kantoaallolle käyttäen yhtä avainsanaa DL-mittauksen tapaan.

5 TESTAUS JA KÄYTTÖÖNOTTO

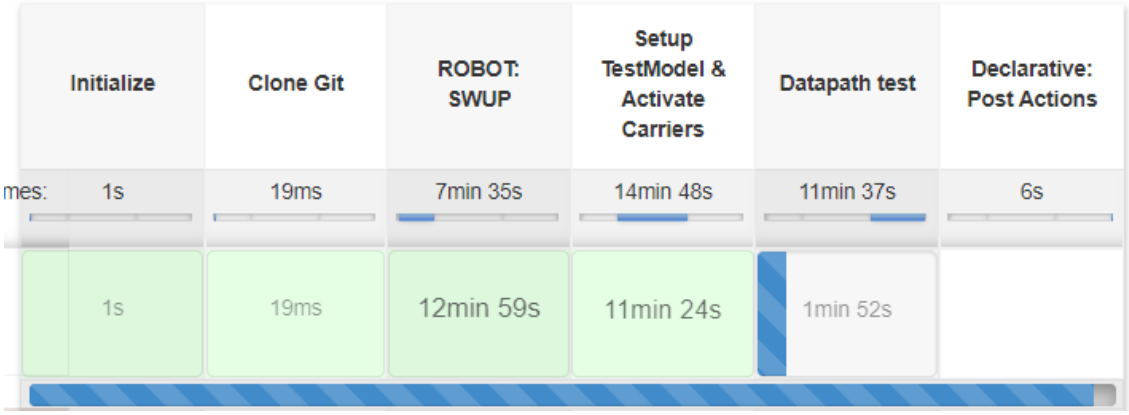
5.1 Uuden version verifiointi

Kun mittauskirjasto ja sitä ajava Robot Framework testilinja oli valmis, oli aika aloittaa sen käyttöönotto radio-ohjelmistojen testiautomaatiossa, jossa sillä voidaan ajaa automaattisesti testejä yöaikaan radio-ohjelmistoille. Testiautomaatio pyörii Jenkinsissä.

Saadakseen radio-ohjelmistojen testit pyörimään automaattisesti ja lähettämään tulokset palveluille, joista voidaan seurata radio-ohjelmistojen testituloksia, täytyi muokata aikaisemmin luotua Jenkins-testilinjaa, jota oli käytetty aikaisemmin mittauskirjaston toiminnallisuuksien verifiointiin. Jenkins-testilinjan rakennetta täytyi muokata, jotta laaja Robot Framework -testilinja, joka sisälsi myös mittauskirjaston kutsuja, saatiin ajettua Jenkinsin kautta. Myös tulosten käsittelyyn tehtiin muutoksia, jotta ne saatiin lähettämään testitulokset halutulla tavalla radio-ohjelmistojen testitulosten seurantapalvelulle. Kun tulokset oli saatu näkymään testitulosten seurantapalveluilla haluttuun tapaan, testiajoja verifioitiin vielä automaattiajolla muutaman yön verran, ennen kuin testilinja otettiin käyttöön CI:ssä.

Kuvassa 13 nähdään Jenkins ajon eri vaiheet, joissa verifioitiin Robot Frameworkilla luotua datapolkutestiä (engl. datapath test) ja mittauskirjastoa. Datapolkutestissä testataan radion tiedonsiirtoa. Kyseinen Robot Frameworkilla luotu datapolkutesti sisältää uudella mittauskirjastolla tehtävät mittaukset. Ensimmäisessä ”Initialize” -vaiheessa alustetaan testiajo- sekä ympäristökohtaisia parametrejä, mm. konfiguraatiotiedostot, testattavan radio-ohjelmiston haara. Seuraavassa ”Clone Git” -vaiheessa päivitetään kirjastot Git-versionhallintatyökalusta. Kuvassa näkyvästä ajosta se oli kuitenkin hetkellisesti otettu pois käytöstä. Kolmannessa ”ROBOT: SWUP” -vaiheessa ajetaan radio-ohjelmiston päivitys käyttäen sille tarkoitettua Robot Framework -testilinjaa. Neljännessä ”Setup TestModel & Activate Carriers” -vaiheessa radio konfiguroidaan ja sille pystytetään kantoaallot haluttuihin antenniportteihin. Tämä vaihe myös ajetaan sille tarkoitetulla Robot Framework -testilinjalla. Tämän jälkeen ”Datapath test” -kohdassa ajetaan Robot Framework -testilinja, joka sisältää mittaustestitapaukset, jotka suoritetaan käyttäen uutta mittauskirjastoa. Suoritettavat mittaukset ovat DL-mittaus, UL PUSCH -mittaus sekä UL PRACH -mittaus. Viimeisessä ”Declarative: Post

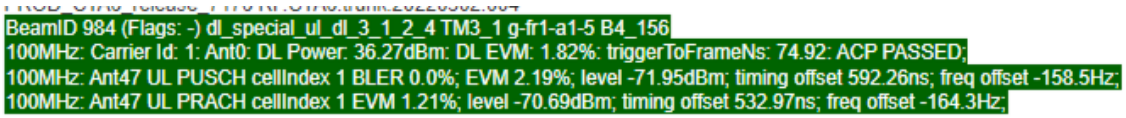
Actions” -kohdassa testitulokset laitetaan näkyviin Jenkins-testinäkymään sekä lähetetään palveluille, joista eri radio-ohjelmistojen testituloksia voidaan seurata. (Kuva 13.)



Kuva 13. Radio-ohjelmiston testaukseen käytettävän Jenkins-testilinjan testitapaukset

Kuvassa 14 näkyy tulokset Jenkins-testiajosta, missä ajettiin Robot Frameworkkia käyttäen radio-ohjelmiston päivitys, testimallin ja kantoaallon pystytys radiolle sekä datapolkutesti, jossa ajettiin mm. mittaukset radiolle uudella mittauskirjastolla. Kuvassa testattu radio-ohjelmisto meni läpi kyseisistä testeistä ja mittaustulokset olivat mittausrajojen sisäpuolella.

Kyseisessä kuvassa nähdään UL- ja DL-mittaustulokset. Kuvasta voidaan myös nähdä hieman, millä konfiguraatioilla mittaus ajettiin. Kuvasta mm. näkee mille antenneille mittaukset tehtiin, mittaukset tehtiin 100MHz:n kantoaallolle, millä testimallilla testit ajettiin jne. (Kuva 14.)



Kuva 14. Mittaustulokset Jenkins-testitulosnäkymä.

5.2 Uuden version käyttöönotto

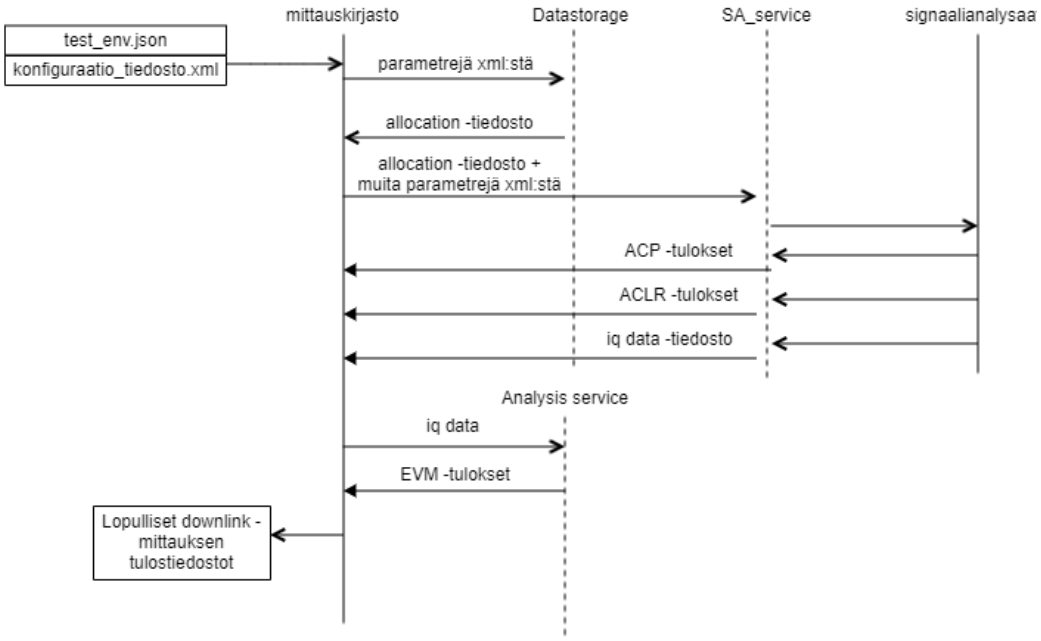
Kun Robot Framework -testilinja sekä mittauskirjasto oli todettu toimivaksi yön sekä viikonlopun yli Jenkinsissä ajetuissa testiajoissa, seuraava vaihe oli ottaa se myös muissa CI:n testipaikoilla käyttöön. Ensimmäisenä sitä ruvettiin ottamaan savutesti (engl. smoke test) -paikalla. Kyseisellä

testipaikalla oli lähes saman mallinen NR-radio ja vanhempi kantataajuusyksikkö. Kyseessä oli siis samankaltainen testipaikka, kun missä mittauskirjastoa oli kehitetty sekä verifioitu.

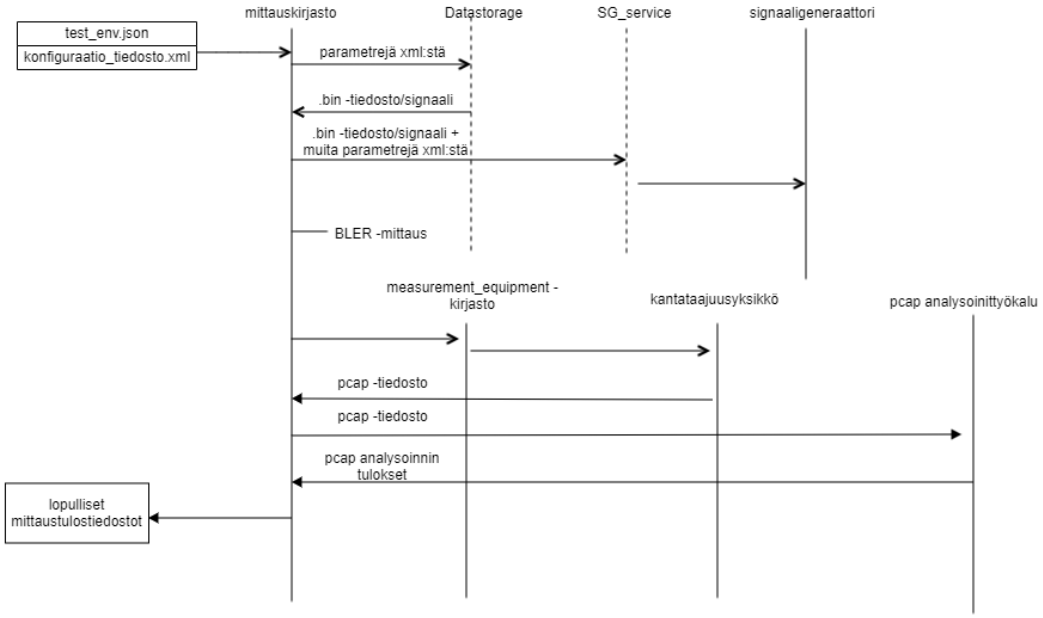
Savutesti tarkoittaa ohjelmiston/ohjelmistopäivityksen yksinkertaista testausta, jossa testataan ohjelmiston perustoiminnallisuuksia. Mikäli savutestissä havaitaan ongelmia, testattava ohjelmisto on julkaisukelvoton.

Tätä ennen kuitenkin Robot Framework -testilinjan ja mittauskirjaston käyttöönotto oli dokumentoitava hyvin, jotta niiden käyttöönotto olisi helpompaa sekä siksi etteivät testiympäristöt eroaisi toisistaan liikaa. Mikäli käyttöönottoa ei dokumentoitaisi, saattaisivat testipaikat erota liikaa toisistaan, koska moni tiimin jäsen ottaisi Robot Framework -testilinjan ja mittauskirjaston käyttöön eri tavalla. On siis hyvin tärkeää, että ympäristöt ovat mahdollisimman identtisiä, vaikka esimerkiksi järjestelmämoduuli tai radio olisikin eri. Lopuksi luotiin uusi ympäristö, eli Jenkins-testilinja, joka ajaa Robot Framewrok -testilinjaa, jossa suoritetaan mittaukset uudella mittauskirjasolla. Näin varmistettiin, ettei dokumentoinnissa ollut huolimattomuusvirheitä.

Tämän jälkeen tehty ympäristö kopioitiin CI:n savutestipaikalle. Tämän jälkeen savutestipaikalla oli mahdollista ajaa automaattisesti Jenkinsin kautta mittaukset uudella mittauskirjastolla ja muut testit Robot Framework -linjalla sekä tulokset lähetetään automaattisesti näistä testeistä palveluun, josta radio-ohjelmistojen testituloksia voidaan seurata.



Kuva 16. Yksinkertaistettu kaavio uudella mittauskirjastolla suoritettavasta DL-mittauksesta.



Kuva 17. Yksinkertaistettu kaavio uudella mittauskirjastolla suoritettavasta UL-mittauksesta.

6 TYÖN ARVIOINTI JA POHDINTA

Projektissa otettiin vanha Python-mittausskripti ja luotiin sille erillinen projekti Git-versionhallintatyöhaluun. Mittausskripti pilkottiin pienempiin Python-moduuleihin ja koodia siivottiin sekä yksinkertaistettiin. Tässä vaiheessa ei siis enää puhuta mittausskriptistä, vaan mittauskirjastosta. Mittauskirjastolle luotiin Robot Framework -tiedosto/rajapinta, jossa olevilla avainsanoilla mittauksia pystyttiin ajamaan Robot Frameworkkia käyttäen ja kutsumaan Robot Framework -testilinjalta. Mittauskirjaston parametrisointia myös parannettiin siellä olevien arvojen ja muuttujien kovakoodauksen korvaamisella ulkopuolelta tulevien konfiguraatiotiedostojen parametreillä. Myöskään signaalianalysaattorille ja -generaattorille lähetettävien konfigurointien ja testivektorien hallinnasta paikallisesti päästiin eroon ottamalla käyttöön Datastorage-rajapinta. Mittauskirjastolle lisättiin tuki sekä uudelle että vanhalle kantataajuusyksikölle.

Projekti oli haastava ja odotettua aikaa vievämpi, koska opittavaa oli paljon. Radio-ohjelmistojen automatisoidut testiympäristöt ovat yllättävän suuria ja monimutkaisia ja muuttuvia parametrejä on valtava määrä.

Vaikkakin mittauskirjasto saatiin käyttöön, jäi siihen vielä parannettavaa ja lisättäviä toiminnallisuuksia, kuten useiden kantoaaltojen mittaus sekä tuki LTE-mittauksia varten. Mittauskirjaston parametrisointiin liittyen löytyy myös parannettavaa, kuten yksittäisten antennikanavien tehohäviön määrittely.

Mittauskirjasto otettiin kuitenkin jo käyttöön CI-testipaikoille, joilla ajetaan hieman yksinkertaisempia mittauksia. Tämä on hyvä saavutus, kun otetaan huomioon, että opiskeltavaa oli todella paljon ja on edelleen, kun ajatellaan uuden mittauskirjaston tulevia toiminnallisuuksia.

Projektin aikana ymmärrys NR-radio-ohjelmistojen testaamisesta ja siihen käytettävistä työkaluista kasvoi valtavasti, joka on todella positiivista. Negatiivisena puolena ehkä joidenkin yksittäisen toimintojen toteutusta olisi voinut suunnitella hieman paremmin, joka olisi varmasti nopeuttanut niiden valmiiksi saamista. Vaikka mittauskirjaston käyttö ja ylläpito on nyt helpompaa, on sen käyttöönoton yksinkertaistamisessa vielä hieman tekemistä, koska Python-kirjastoriippuvuuksia on paljon.

LÄHTEET

1. Wikipedia. 2022. Nokia (yritys). Hakupäivä 30.11.2022. https://fi.wikipedia.org/wiki/Nokia_(yritys).

2. Mikkonen, Juuso. 2019. Mikä ihmeen Jenkins? Jatkuva integraatio pitää koodin tuoreena. Tiivi 12.4.2019 (päivitetty 2.6.2020). Hakupäivä 14.9.2022. https://www.tivi.fi/uutiset/mika-ihmeen-jenkins-jatkuva-integraatio-pitaa-koodin-tuoreena/61d2bad5-ca2b-4f12-a4f4-b3809e0e0dd7.

3. Wikipedia. 2022. Robot Framework. Hakupäivä 23.3.2022. https://fi.wikipedia.org/wiki/Robot_Framework.

4. Knowit. 2022. Robot Framework Tutorial 1 – Johdanto. Hakupäivä 25.3.2022. https://www.youtube.com/watch?v=H9YVlFKdOeM&ab_channel=Knowit.

5. Jenkins. 2022. Jenkins - Oracle Corporation. Hakupäivä 14.0.2022. https://www.jenkins.io/.

6. Git. 2022. Git – Microsoft Corporation. Hakupäivä 7.9.2022. https://git-scm.com/.

7. Fitzgibbons, Laura. 2019. Baseband unit (BBU). Hakupäivä 24.4.2022. https://www.techtarget.com/whatis/definition/baseband-unit-BBU.

8. Danel, Eve. 2021. EVM (Error Vector Magnitude): Why it Matters and How it's Measured. Litepoint 10.3.2021. Hakupäivä 8.5.2022 https://www.litepoint.com/blog/error-vector-magnitude-why-it-matters-and-how-its-measured/.

9. Acquisition Mode (5G NR). 2022. Hakupäivä 9.5.2022. https://rfmw.em.keysight.com/wireless/helpfiles/89600b/webhelp/subsystems/newradio/content/newradio_dlg_time_acquisitionmode.html.

10. 5G NR PRACH – contents, function, physical layer processing. 2022. RF Wireless World. Hakupäivä 18.8.2022 https://www.rfwireless-world.com/5G/5G-NR-PRACH.html.

11. Wikipedia. 2022. Pcap. Hakupäivä 14.9.2022 https://en.wikipedia.org/wiki/Pcap.

12. What is radio access network (RAN). 2021. Red Hat. https://www.redhat.com/en/topics/5g-networks/what-is-radio-access-network.