Opinnäytetyö (AMK)

Tieto- ja viestintätekniikka

2024

Kristo Ilmanen

Radiomodeemin komentoparserin analyysi ja uudelleenkirjoitus

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tieto- ja viestintätekniikka

2024 | 43 sivua

Kristo Ilmanen

Radiomodeemin komentoparserin analyysi ja uudelleenkirjoitus

Radioteknologia mahdollistaa luotettavan tiedonsiirron useissa kaupallisissa järjestelmissä. Tämänkaltaiset laitteet tarvitsevat käyttöliittymiä sekä interaktiiviseen että ohjelmalliseen hallintaan. SATELin radiomodeemeissa, molemmat voidaan toteuttaa SL-komentojen avulla.

SL-komennot ovat olleet käytössä pitkään, jonka aikana niistä vastaavasta komentoparserista on tullut hankala ylläpitää. Tästä syystä, tämän opinnäytetyön tavoitteena oli aloittaa uuden SL-komentoparserin kehitys.

Tämän opinnäytetyön aikana toteutettiin useita SL-komentoparserin keskeisimpiä ominaisuuksia. Uusi komentoparseri myös integroitiin onnistuneesti yhteen SATELin radiomodeemeista. Tämän lisäksi tutkittiin komentoparserin suorituskykyä sekä toiminnallisuustestausta.

Jatkokehityksen jälkeen uutta komentoparseria voidaan hyödyntää SATELin radiomodeemeissa. Tällä hetkellä keskeisin puute on useiden välttämättömien komentojen puuttuminen. Ennen käyttöönottoa on myös suoritettava kattavampaa testausta mahdollisilta regressioilta välttymiseksi. Alustava käyttöönotto voidaan aloittaa porrastetusti, vähemmän kriittisistä tuotteista.

Asiasanat:

komentoparseri, radiomodeemi, laiteohjelmisto

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences

Information and Communications Technology

2024 | 43 pages

Kristo Ilmanen

Analysis and rewrite of a radio modem’s command parser

Radio technology enables reliable data transmission for various commercial systems. Such devices require interfaces for interactive and programmatic control. In SATEL’s radio modems, both can be accomplished with SL commands.

SL commands have been in use for a long time, during which the command parser used for handling them has become hard to maintain. Hence, the purpose of this thesis was to begin the implementation of a new SL command parser.

During this thesis several of the SL command parser’s key features were implemented. The new command parser was also successfully integrated to one of SATEL’s radio modems. Performance analysis and functionality testing were also explored.

After further development, the new SL command parser can be utilized in SATEL’s radio modems. At the moment, the most notable defect is the lack of several essential commands. Before deployment, the new command parser must also be comprehensively tested to avoid any possible regressions. Initial deployment can start gradually from less critical devices.

Keywords:

command parser, radio modem, firmware

Sisältö

Kuvat

Taulukot

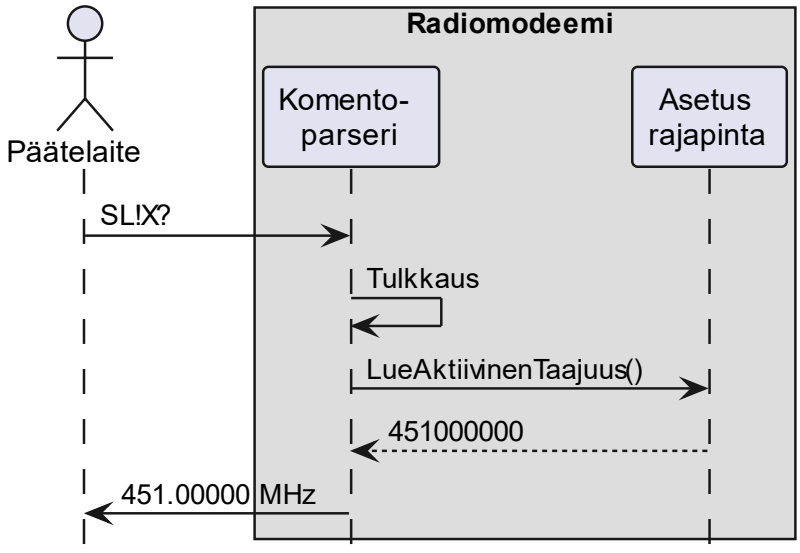
Lyhenteet

AT-komento	Modeemien hallintaan käytetty AT-alkuinen merkkijono. (Attention-komento)
CR	Rivin alkuun palauttava kontrollimerkki. (Carriage return)
CSV	Taulukointiin käytetty tekstipohjainen tiedostomuoto. (Comma-Separated Values)
CWE	Yhteisökehitetty tietokanta yleisistä laitteiston ja ohjelmiston haavoittuvuuksista. (Common Weakness Enumeration)
EEPROM	Tietokonejärjestelmissä käytetty haihtumaton muisti. (Electronically Erasable Programmable Read-Only Memory)
ITU-T	Kansainvälinen televiestintäliitton televiestinnän standardointisektori (International Telecommunication Union Telecommunication Standardization Sector)
LF	Seuraavalle riville siirtävä kontrollimerkki. (Line Feed)
NMS-protokolla	SATELin radiomodeemien hallintaan käytetty binääripohjainen protokolla. (Network Management System)
SL-komento	SATELin radiomodeemien hallintaan käytetty SL-alkuinen merkkijono. (SATEL-komento)
UHF	300 – 3000 MHz radioaaltojen taajuusalue. (Ultra High Frequency)
USB	Yleinen oheislaitteiden liittämiseen käytetty sarjaväylä. (Universal Serial Bus)

1 Johdanto

Radioteknologia toimii tärkeänä työkaluna useissa kaupallisissa järjestelmissä. Sekä pienet että suuret yritykset käyttävät erilaisia radiolaitteita päivittäisissä liiketoimissaan, kuten laitteiston ja henkilöstön koordinoinnissa. [1] Radioteknologiaa käytetään varsinkin toimintakriittisissä järjestelmissä, joissa tiedonsiirron luotettavuus ja latenssin vähäisyys ovat äärimmäisen tärkeitä [2]. Tämän opinnäytetyön toimeksiantaja, SATEL, on yksi radioteknologian valmistajista.

SATELin radiomodeemeja voidaan ohjata muun muassa SL-komentojen (SATEL-komento) avulla. SL-komennot ovat lyhyitä SL-alkuisia merkkijonoja, joka mahdollistava radiomodeemin interaktiivisen sekä ohjelmallisen hallinnan. SL-komentoja käytetään varsinkin integroidessa radiomoodi osaksi suurempaa järjestelmää, asiakkaan oman käyttöliittymän taakse. Esimerkiksi taajuuden ja osoitteiden muuttaminen onnistuu SL-komennoilla. [3] Kuva 1 esittää aktiivisen taajuuden lukemista SL-komennon avulla.



Kuva 1. Sekvenssikaavio SL-komennon suorituksesta.

Vuonna 1986 perustettu SATEL on toimintakriittisen radiokommunikaation asiantuntija, joka kehittää, valmistaa sekä myy langattomia yhteysratkaisuja

useihin teollisuussovelluksiin [4]. SATELin radiomodeemeja on käytetty muun muassa Mount Everestin kaltaisessa ankarassa ympäristössä, jossa perinteisemmät mobiiliverkkoratkaisut eivät ole mahdollisia [5]. Hieman lähempänä merenpintaa, SATELin radiomodeemeja on käytetty Santiagon kansainvälisellä lentokentällä reaaliaikaisten säätietojen välittämiseen kiitoradan ja lennonjohdon välillä [6]. Keskeinen ominaisuus näissä käyttötapauksissa on luotettavuus, jonka tulee heijastua myös radiolaitteessa käytetyssä ohjelmistossa.

Tämän opinnäytetyön tavoitteena on aloittaa uuden SL-komentoparserin kehitys. SL-komennot kehitettiin alun perin yli kaksi vuosikymmentä sitten ja tätä samaa ohjelmistokoodia on vuosien varrella laajennettu eri projektien tarpeisiin. Nykyisessä komentoparserissa on kuitenkin useita ongelmia, joista keskeisin tekninen velka, joka hankaloittaa ohjelmistokoodin ylläpitoa. Tästä syystä nykyinen komentoparseri on päätetty kirjoittaa uudestaan. SL-komentojen pitkän historia takia, uuden komentoparserin on kuitenkin oltava taaksepäin yhteensopiva vanhojen toimintojen kanssa. Uuden komentoparserin on myös toimittava jo kentällä käyttöön otetuissa laitteissa.

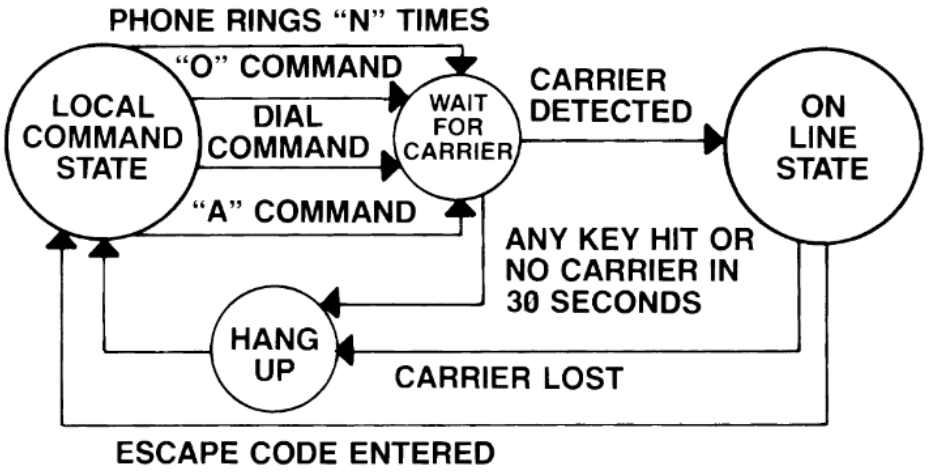
Tämä opinnäytetyö alkaa SL-komentojen sekä niitä edeltäneiden AT-komentojen (attention-komento) esittelyllä, luvussa 2. Taustaesittelyn jälkeen, luvussa 3 perehdytään uudelle SL-komentoparserille asetettuihin vaatimuksiin. Ennalta määritettyjen vaatimusten lisäksi, luvussa 4 nykyisen komentoparserin tilannetta tutkittiin myös omalla testauksella. Uuden komentoparserin keskeisimpiä ominaisuuksia esitellään luvussa 5. Toteutuksen jälkeen, uuden komentoparserin alustavaa integraatiota käydään läpi luvussa 6. Lopuksi luvussa 7 kuvaillaan uuden komentoparserin testausprosessia sekä dokumentaatiota.

2 Tausta

Ennen SL-komentojen tarkempaa esittelyä on palattava takaisin 80-luvulle, aikaan, jolloin tietokoneteollisuus oli vasta alkutekijöissään ja yhteys internetiin muodostettiin pääasiassa puhelinlinjojen kautta [7]. Tämän yhteyden mahdollisti modeemi, eli modulaattori-demodulaattori, joka toimii kääntäjänä datan ja audiosignaalien välillä [8]. Tässä luvussa esitellään tämän opinnäytetyön keskiössä olevat SL-komennot, sekä niitä edeltäneet Hayesin AT-komennot.

2.1 AT-komennot

Dennis Hayesin perustama Hayes Microcomputer Products, Inc. kehitti 80-luvulla tekstipohjaisen komentokielen, jonka avulla päätelaite kykeni ohjaamaan sarjaväylään kytketyn modeemin toimintaa [7]. Hayesin modeemit toimivat joko online- tai komentotilassa. Online-tilassa modeemi suorittaa sille tyypillisiä toimintoja, eli kommunikoi puhelinlinjojen kautta toisen modeemin kanssa. Erikoisominaisuus, joka erotti Hayesin modeemin kilpailevista tuotteista on paikallinen komentotila, jossa se analysoi ja suorittaa päätelaitteelta saapuneita komentoja. Nämä komennot voivat olla joko käyttäjän tai ohjelman syöttämiä. Komentoja ei suoriteta online-tilassa, mutta käyttäjä kykenee pakokoodin (engl. escape code) avulla siirtämään modeemin tilapäiseen komentotilaan. esittää komento- ja online-tilan välillä siirtymistä. [9]



Kuva 2. Hayesin modeemin komento- ja online-tilan välillä siirtyminen [9].

Erityistä `A\` toistokomentoa lukuun ottamatta, kaikki komentorivit alkavat AT-etuliitteellä, joka toimii huomiokoodina. Yhteensopivuuden lisäämiseksi, Hayesin modeemeissa on myös autobaud ominaisuus, eli automaattinen baudinopeuden tunnistus, jonka avulla sarjaväylän asetukset päätellään ensimmäisistä ”A” ja ”T” kirjaimista. Komentorivin loppuosa sisältää yhden tai useamman komennon, jotka modeemi suorittaa yksi kerrallaan. Komentorivi päättyy rivinvaihtomerkkiin, mutta ennen sen syöttämistä yksinkertainen interaktiivinen muokkaus on mahdollista askelpalauttimen avulla. [9]

AT-komennot mahdollistivat minkä tahansa laitteen kommunikoinnin Hayesin modeemin kanssa, mikä teki siitä laitteistoriippumattoman. Tämä oli merkittävä ominaisuus aikana, jolloin useimmat tietokonejärjestelmät käyttivät yhteensopimatonta laitteistoa. Pian Hayesin menestyksen nähtyään myös kilpailevat yritykset ottivat käyttöön kyseisen komentosarjan, josta täten muodostui epävirallinen standardi rajapinta modeemien ohjaamiseen. [7] Tähän perustuu termi ”Hayes yhteensopiva modeemi”.

Kuvassa 3 on Hayes Smartmodeemin mainos Byte-lehdessä vuodelta 1986, mikä havainnollistaa hyvin AT-komentojen merkityksellisyyttä tuona ajanjaksona. Hayes-yhteensopivuus ja AT-komennot nostetaan heti ensimmäisenä esille modeemin tärkeimpien ominaisuuksien joukossa.

Kuva 3. Hayes modeemin mainos Byte-lehdessä vuonna 1986 [10].

2.1.1 Laajennetut AT-komennot

Vuosien varrella useat valmistajat olivat toteuttaneet AT-komentoja, vaikka Hayes ei ollut julkaissut niille virallista standardia. Myöhemmin AT-komennot ovat kuitenkin esiintyneet useissa standardeissa, esimerkiksi kansainvälinen televiestintäliiton televiestinnän standardointisektorin (ITU-T) suosituksessa V.250, joka luokitteli olemassa olevat AT-komentojen käytänteet ja määritteli formaatin niiden laajentamiseen. Taaksepäin yhteensopivuuden säilyttämiseksi, V.250 pyrki tunnistamaan AT-komentojen yleisimmät käytänteet, jotka havaittiin olevan yhtenäisiä modeemivalmistajien välillä. [11]

Laajennetut AT-komennot, joita on esitetty myös taulukossa 1, olivat mahdollisesti AT-komentojen standardisoinnin merkittävin tulos. Laajennetut AT-komennot on jaettu kahteen pääkategoriaan: toiminto- ja parametrikomentoihin. Toimintokomennoilla voidaan suorittaa ennalta määritettyjä toimenpiteitä, kun taas parametrikomennoilla käsitellään modeemin asetuksia. Näiden komentojen olemassaolon, sekä niiden hyväksymät parametrit, voidaan tarkistaa testikomentojen avulla. [11]

Taulukko 1. Alkuperäisiä sekä laajennettuja AT-komentoesimerkkejä [11].

Komento	Selite
`ATA`	Vastaa puheluun
`ATH0`	Katkaise puhelu
`AT+GMI`	Pyydä valmistajan tunnistetietoja (laajennettu komento)

Useiden vuosikymmenien jälkeen AT-komennot ovat yhä käytössä, esimerkiksi useimmissa älypuhelmissa, joissa niiden merkityksellisyys on huomattavasti laajentunut valmistajakohtaisten lisäysten takia. Jotkin erityisemmät AT-komennot eivät edes suorita puhelinverkkoon liittyviä toimintoja, vaan käyttävät laitteen muita resursseja. Perinteisemmän sarjaväylän sijaan, älypuhelimissa AT-komentoja voidaan lähettää esimerkiksi Bluetooth- tai USB-yhdeyden (Universal Serial Bus) kautta. [12]

2.2 SL-komennot

SL-komentoja voidaan pitää SATELin omana AT-komentojen muunnoksena. Kuten nimestä voi päätellä, AT-etuliitteen sijaan, kaikki SL-komennot alkavat SL-etuliitteellä. SL-komennot kehitettiin, koska AT-komennoista puuttui radiokeskeisiä toimintoja, mutta kuten aiemmin todettiin, tämä ei ole estänyt muita valmistajia lisäämästä uusia AT-komentoja omiin tarpeisiinsa.

Laajennettujen AT-komentojen tavoin, myös SL-komennot voidaan jakaa joko toiminto- tai parametrikomentoihin. SL-komennot eivät kuitenkaan seuraa kaikkia laajennettujen AT-komentojen käytänteitä. Esimerkiksi testikomentoja ei tueta, vaan mahdolliset toiminnot on tarkistettava radiomodeemin ohjekirjasta. Yhdellä komentorivillä ei voi myöskään ketjuttaa useampaa SL-komentoa, vaan seuraavan komennon saa lähettää vasta edellisen vasteen jälkeen. Yleisimmät komentojen vasteet ovat `OK` ja `ERROR`. [3]

Myös SATELin radiomodeemeissa on paikallinen komentotila, jonka aikana SL-komennot ovat käytössä ja radio on tilapäisesti pois päältä. Komentotilaan siirrytään `+++` sekvenssillä, jossa jokaisen merkin välissä on oltava lyhyt tauko. SL-komentoja voidaan kuitenkin käyttää myös komentotilan ulkopuolella. [3]

Laitevariantista ja yhteensopivuustilasta riippuen, SATELin radiomodeemeissa saatetaan käyttää myös perinteisempiä AT-komentoja. Nämä AT-komennot eivät kuitenkaan seuraa kaikkia Hayes-yhteensopivia käytänteitä, vaan ne toimivat SL-komentoja vastaavalla tavalla. AT-komentojen suorittamisesta vastaa sama komentoparseri, jonka on erityisessä tilassa käsiteltävä myös AT-etuliitteisiä komentoja.

Taulukossa 2 on esitetty muutamia esimerkkejä SL-komennoista, joita tässä opinnäytetyössä käytetty SATEL-TR4+ -lähetin-vastaanotinmoduuli tukee. TR4+ radiomoduuli esitellään myöhemmin uuden SL-komentoparserin integraation yhteydessä, luvussa 6.

Taulukko 2. TR4+ SL-komentoesimerkkejä [3].

Komento	Selite
`SL%B?`	Lue sarjaportin asetukset.
`SL%Z=0`	Aseta SL-komennot pois päältä.
`SL@C=1,5,MYMESSAGE`	Lähetä kutsumerkki ”MYMESSAGE” 5 minuutin välein.

AT-komennoista poiketen, SL-komennoissa on myös merkittävä reaaliaikaan sidottu ominaisuus. Komentoja ei voi interaktiivisesti kirjoittaa suoraan radiomodeemille, sillä tavallisen päätemerkin sijaan SL-komennon loppua ilmaistaan sarjaväylän tauolla. Tavalliset työpöytä käyttöjärjestelmät saattavat kuitenkin tuottaa tahattomia taukoja, joka johtaa datan jakautumiseen useampaan pakettiin. Tämän vuoksi tauon pituus voidaan määritellä päätelaitteen ominaisuuksien mukaan. Esimerkiksi, jos sarjaväylän asetukset ovat 9600 baudia, kahdeksan databittiä, ei pariteettibittiä ja yksi stop-bitti, niin yhden merkin tauko vastaa noin yhtä millisekuntia. [3] Tästä syystä SL-komentojen interaktiivinen käyttö vaatii niihin erikoistuneen pääteohjelman. SL-komentojen lisäksi, sarjaväylän taukoja käytetään muun muassa radiolähetyksen päättymisen sekä osoitteiden tunnistamiseen [3].

Yksinkertaisen muotoilunsa vuoksi, AT- ja SL-komennot soveltuvat sekä interaktiiviseen, että ohjelmalliseen käyttöön. Tämä on varmasti yksi syy siihen, että molemmat ovat yhä aktiivisessa käytössä, pitkistä historioistaan huolimatta.

3 Vaatimukset

Kuten johdannossa todettiin, tämän opinnäytetyön tavoitteena ei ollut tuottaa SL-komentoparseriin uusia ominaisuuksia, vaan korjata useita pitkäaikaisia ongelmia sekä yhtenäistää SL-komentojen käsittely yhden siirrettävän ohjelmistomoduulin alle. Yksi jaettu toteutus vähentää huomattavasti myös tuotekehityksen, ylläpidon ja testauksen työmäärää.

On syytä huomioida, että merkittävän ohjelmistokomponentin uudelleenkirjoitusta ei tulisi aloittaa kevyin perustein. Vanhaa koodia on vuosien varrella käytetty, testattu sekä korjattu, eikä siihen itsestään synny uusia ongelmia, ilman muutoksia. [13] Juuri tämän riski-hyötysuhteen takia, vanhimpien radiomodeemien ohjelmistoon ei tulla tekemään merkittäviä muutoksia.

SL-komentoja on arviolta noin 200, tosin tarkkaa lukua on hankala määrittää useiden tuotevarianttien ja erityiskomentojen takia. Tämän lisäksi uusia komentoja kehitetään yhä tasaisin väliajoin eri projektien tarpeisiin. Näin ollen, vanhan SL-komentoparserin täydellinen korvaaminen on tämän opinnäytetyö tavoitteiden ulkopuolella. Sen sijaan, tarkoituksena oli ennen kaikkea havainnollistaa uudelleenkirjoituksen hyötyä toteuttamalla komentoparserin keskeisimpiä toimintoja, jotta uutta ja vanhaa komentoparseria voidaan vertailla keskenään.

Tässä luvussa on käyty läpi projektin aloitustapaamisessa esille nousseita vaatimuksia uudelle SL-komentoparserille.

3.1 Reaaliaikaisuus

Reaaliaikaisella järjestelmällä tarkoitetaan järjestelmää, joka vastaa ulkoisiin tapahtumiin ennalta määritetyssä ajassa. Toisin sanoen, reaaliaikaisen järjestelmän oikeanlainen toiminta ei ole ainoastaan riippuvainen oikeista laskennallisista tuloksista, vaan myös niiden tuottamiseen käytetystä ajasta.

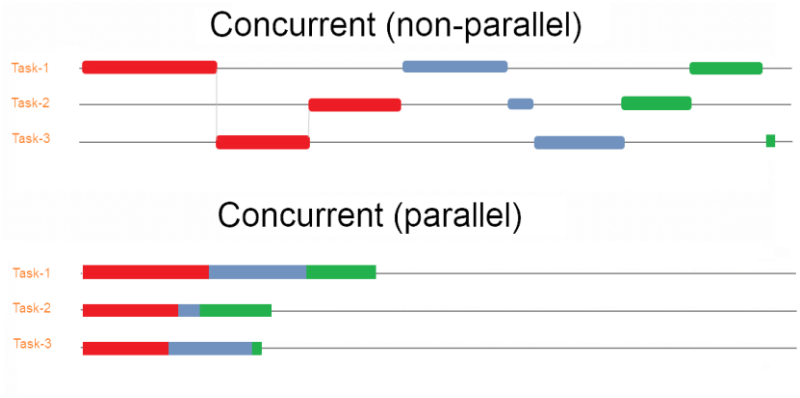
Reaaliaikaisen järjestelmän ympäristö luo nämä ulkoiset tapahtumat, jotka voivat olla luonteeltaan joko synkronisia tai asynkronisia. Ulkoisten tapahtumien oikeanlaiseen käsittelyyn kuuluu tapahtuman tunnistaminen, vaaditun prosessoinnin suorittaminen sekä mahdollisen vastauksen tulostaminen. [14]

Reaaliaikaiset järjestelmät voidaan jakaa aikarajojensa kriittisyyden perusteella pehmeisiin ja koviin reaaliaikaisiin järjestelmiin. Pehmeissä reaaliaikajärjestelmissä aikarajoista poikkeaminen vain heikentää tuloksen laatua, kun taas kovissa reaaliaikaisissa järjestelmissä virhetoleranssi on lähes olematon ja viivästykset saattavat joissain tilanteissa jopa johtaa ihmishenkien menetykseen. [15]

Vaikka SATEL tarjoaa myös Linux-pohjaisia tuotteita, sen perinteiset radiomodeemit ovat yksiytimisiä, käyttöjärjestelmättömiä (engl. bare metal), reaaliaikaisia järjestelmiä. Keskeytyksiä lukuun ottamatta, laiteohjelmistoa suoritetaan lineaarisesti yhdessä isossa silmukassa, jossa ohjelmisto prosessoi dataa useista ulkoisista lähteistä. Tällaisessa järjestelmässä yksikään ohjelmistomoduuli ei saa omia prosessoria odottamalla ulkoisia tapahtumia, sillä se johtaisi kaikkien muiden toimintojen viivästymiseen. Tämän takia reaaliaikavaatimusten huomioiminen on täysin välttämätön ominaisuus myös SL-komentoparserissa. Jos komennon toimintoa ei ole mahdollista suorittaa saman tien, esimerkiksi radion asetusten muuttaminen lähetyksen aikana, on siihen tarvittavat parametrit tallennettava sopivampaa aikaa varten. Tilannetta hankaloittaa myös se, että SL-komennon vastetta ei saa lähettää eikä seuraavaa komentoa hyväksyä ennen edellisen komennon valmistumista [3].

3.2 Samanaikaisuus

Samanaikaisuus, joka useasti sekoitetaan rinnakkaisuuteen, tarkoittaa useiden toimintojen suorittamista saman aikaikkunan sisällä. Samanaikaisuuteen ei kuitenkaan vaadita, että enemmän kuin yksi näistä toiminnoista olisi käynnissä tiettynä ajanhetkenä. [16] Samanaikaisuuden ja rinnakkaisuuden eroa on helpoin havainnollistaa kuvan 4 kaltaisella kaaviolla.



Kuva 4. Samanaikaisuuden ja rinnakkaisuuden ero [17].

Reaaliaikaisissa käyttöjärjestelmissä samanaikaisuus toteutetaan usein tehtävien tai säikeiden avulla. Jokaisella tehtävällä on oma kontekstinsa, johon muun muassa prosessorin rekisterit sekä pino voidaan tallentaa siirtyessä tehtävästä toiseen. Tätä tapahtumaa kutsutaan kontekstin vaihdoksi. [15] Kuitenkin jopa kaikkein yksinkertaisimmat, sulautetut järjestelmät kykenevät samanaikaisuuteen jakamalla laiteohjelmiston pieniin itsenäisesti suoritettaviin osiin. Reaaliaikaista käyttöjärjestelmää hyödyntävää ohjelmistoa voi olla helpompi kehittää, mutta kontekstinvaihdoilla on myös vaikutus sulautetun järjestelmän suorituskykyyn [15].

Oikein toteutettuna, nopeasti vaihteleva samanaikaisuus vaikuttaa käyttäjän näkökulmasta siltä, että useita toimintoja suoritetaan rinnakkain, vaikkei se olisikaan laitteistolle mahdollista. Aito rinnakkaisuus, jota esimerkiksi nykypäivän henkilökohtaiset tietokoneet hyödyntävät, vastaavasti vaatii moniajoon kykenevän prosessorin. Rinnakkaisuus siis edellyttää samanaikaisuutta, mutta samanaikaisuus ei edellytä rinnakkaisuutta. [16]

Radiomodeemista ja sen asetuksista riippuen, SL-komentoja voidaan käyttää myös samanaikaisesti useasta sarjaportista. Tällöin vaste kirjoitetaan samaan porttiin, josta alkuperäinen komento on peräisin. [3] Tämä ominaisuus on kuitenkin jälkikäteen lisätty nykyiseen komentoparseriin, joka ilmenee esimerkiksi niin, että muutamassa erityisessä komennossa vasteen ulostulo on kovakoodattu ensisijaiseen dataporttiin.

3.3 Siirrettävyys ja konfiguroitavuus

C-kielen esikääntäjä on erillinen makroprosessori, jota kääntäjä kutsuu ohjelmakoodin muokkaamiseen, ennen varsinaisen kääntämisen aloittamista. Yhdessä otsikkotiedostojen (engl. header file) kanssa, joissa esitellään funktioiden prototyypit sekä makromääritykset, esikääntäjä mahdollistaa nykypäivän ohjelmistokehityksessä itsestäänselvyytenä pidetyn ominaisuuden, eli jaettujen ohjelmistomoduulien hyödyntämisen. [18]

Varsinkin laiteohjelmiston parissa, ehkä yksi esikääntäjän hyödyllisimmistä ominaisuuksista on ehdollinen kääntäminen. Esikääntäjän ehtolauseilla voidaan ohjelmallisesti määrittää, otetaanko jokin koodin osa mukaan lopulliseen käännökseen. Kaksi tärkeintä syytä ehdollisen kääntämisen käyttöön ovat erilaisten ohjelmistojen tuottaminen samasta lähdetiedostosta sekä vaihtoehtoisen koodin käyttäminen laitteistoriippuvuuksien muuttuessa. [18] Esikääntäjää käytetään runsaasti SATELin laiteohjelmistoissa, varsinkin eri tuotevarianttien hallintaan.

Samankaltainen muokattavuus voidaan vaihtoehtoisesti toteuttaa myös funktio-osoittimien avulla. Funktio-osoittimia käytetään varsinkin matalan tason ajurikoodissa olio-ohjelmointia muistuttavan paradigman mahdollistamiseen [19]. Tällöin rajapinnan toimintaa on mahdollista myös kontrolloida ajon aikana, lisäämällä ja poistamalla osoittimia tarpeen mukaan. Koska C-kielen esikääntäjä toimii vain tekstitasolla, virhetilanteessa se saattaa tuottaa hyvin vaikeaselkoisia ongelmia, kun taas funktio-osoittimien käyttö ottaa täyden hyödyn irti kääntäjästä ja sen tyyppijärjestelmästä.

Siirrettävyyden lisäämiseksi, uusi ohjelmistomoduuli tulee olla konfiguroitavissa muokattavan otsikkotiedoston (engl. configuration header file) avulla. Muokattavalla otsikkotiedostolla mahdollistetaan ennalta määritetyt, projektikohtaiset muutokset, perehtymättä muokattavan lähdekoodin yksityiskohtiin. Muun muassa radioliikenteen salauksesta vastaava ohjelmistomoduuli on toteutettu tällä tavalla SATELin radiomodeemeihin. SL-komentoparserin tapauksessa, tässä otsikkotiedostossa määritellään rajapinta,

jolla se kommunikoi laitteiston, pääasiassa radion sekä EEPROMin (Electronically Erasable Programmable Read-Only Memory), kanssa. Otsikkotiedostossa määritellään myös muistinkäyttöön liittyviä asetuksia, kuten syöttö- ja ulostulopuskurien kapasiteetti.

3.4 Rivinvaihtojen käsittely

On tärkeä huomioida, että yhä tänä päivänä useimmat tietokonejärjestelmät esittävät rivinvaihtoa eri tavalla tekstipohjaisessa datassa. Yleisimmät rivinvaihtomerkit ovat Unixin `\n`, vanhemmissa Applen järjestelmissä käytetty `\r` sekä Windowsin käyttämä `\r\n`. Kaksimerkkinen `\r\n` rivinvaihto on peräisin vanhoista kirjoituskoneista, joissa rivin vaihtaminen oli jaettu kahteen toimintoon: vasempaan reunaan palaamiseen sekä seuraavalle riville siirtymiseen. [20] AT- ja SL-komentojen yhteydessä `\r` ja `\n` merkit esitetään usein myös `<CR>` (carriage return) ja `<LF>` (line feed) muodoissa.

Kuten aiemmin todettiin, SL-komennon loppua ilmaistaan sarjaväylän tauolla, eikä rivinvaihdolla, toisin kuten AT-komennoissa. Tästä huolimatta, SL-komennot on määritelty niin, että valinnainen rivinvaihto hyväksytään komennon lopussa [3]. Käytännössä tämä toteutuu tällä hetkellä vain osassa komennoista, käyttäjän näkökulmasta melkein sattumanvaraisesti. Tämä johtuu siitä, että mahdollista rivinvaihtoa ei karsita pois ennen komennon tulkkausta. Uudessa toteutuksessa halutaan päästä eroon tästä ristiriitaisuudesta. Kaikkien komentojen tulisi hyväksyä yllä esitellyt kolme yleisintä rivinvaihtoa.

3.5 Käyttöoikeustaso

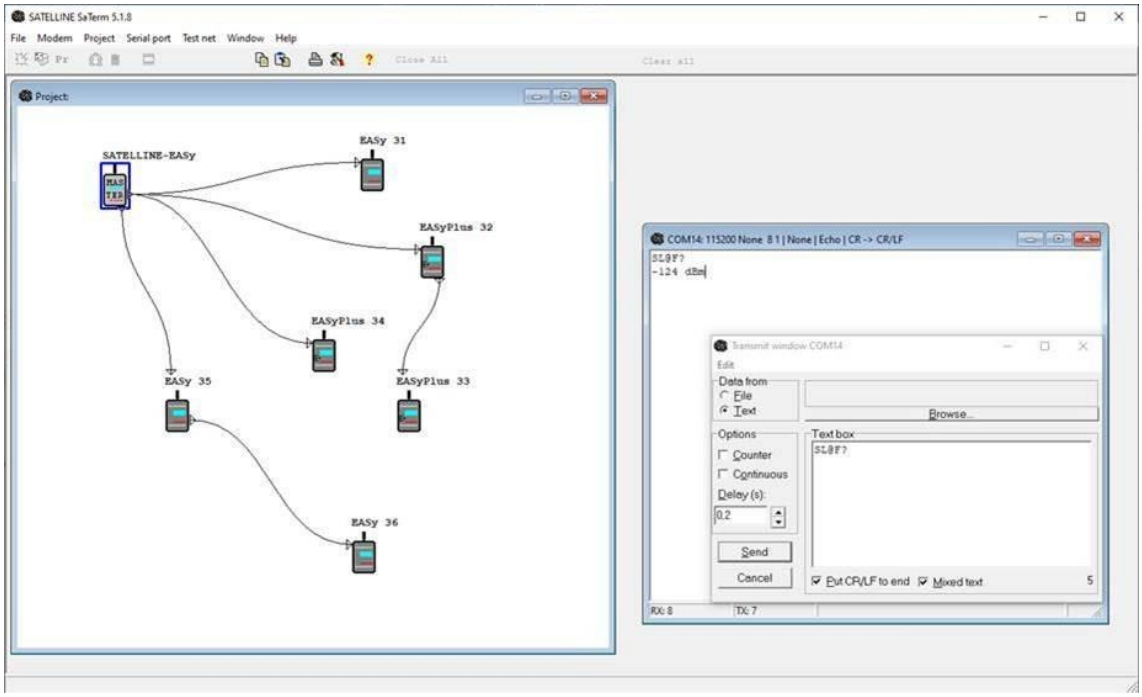
SL-komentoihin halutaan myös lisätä NMS-protokollaa vastaava käyttöoikeustaso. NMS-protokolla (Network Management System) on toinen SATELin kehittämä binääripohjainen protokolla, jossa jokaista asetusta, diagnostiikka arvoa ja komentoa vastaa uniikki NMS-tunniste [21].

Käyttöoikeustasolla voidaan jaotella radiomodeemin toimintoja eri käyttötarpeisiin.

4 Analyysi

Projektin ennalta määritettyjen vaatimusten lisäksi, nykyisen SL-komentoparserin tilannetta kartoitettiin myös omalla testauksella. Tässä luvussa on käyty läpi muutamia havaittuja ongelmia.

Testauksen aikana SL-komentoja lähetettiin SATELLINE SaTermin avulla, joka on SATELin radiomodeemien konfigurointiin, testaukseen ja uudelleenohjelmointiin käytettävä pääteohjelmisto. SL-komentojen kannalta SaTermin tärkein ominaisuus on pääteikkunatila, jonka avulla dataa voidaan syöttää radiomodeemin sarjaporttiin. Kuten kuvassa 5 ilmenee, SaTerm ei kuitenkaan ole ainoastaan tavallinen pääteohjelma, vaan sitä voidaan käyttää myös esimerkiksi radioverkkojen piirtämiseen. [22] Pitkästä iästään huolimatta, SATELLINE SaTermin on ilmeisesti yhä yksi asiakkaiden suosikkiohjelmistoista, vaikka useita uudempia sovelluksia on saatavilla.



Kuva 5. SATELLINE SaTerm kuvakaappaus [23].

4.1 Ylimääräiset merkit

Yleinen ongelma, joka esiintyi useimmissa SL-komennoissa, on ylimääräisen datan hyväksyminen komennon lopussa. Käytännössä tämä ei johda radiomodeemin virheelliseen käyttäytymiseen, mutta se kuitenkin rikkoo SL-komentojen määritelmää: ”ylimääräisiä merkkejä ei sallita SL-komennon lopussa” [3] ja täten luokitellaan viaksi. Esimerkiksi virheellistä syötettä `SL%Z?1234` käsitellään kuten komentoa `SL%Z?`, eli tulkkaus pysähtyy heti kun hyväksyttävä komento on löytynyt, eikä komentorivin loppuosaa tarkisteta.

4.2 Nollamerkkien käsittely

Merkkijonojen sisäinen esitys C-kielessä on siitä erityinen, että toisin kuin monissa muissa kielissä, merkkijonon pituutta ei välttämättä tunneta. Sen sijaan, merkkijonoa on tarkoitus käydä läpi tavu kerrallaan, kunnes löydetään nollamerkki. Nollamerkki-päätteisillä merkkijonoilla, tai tarkemmin sanottuna niiden pituuden toistuvalla laskemisella, voi olla myös merkittävä vaikutus ohjelmiston suorituskykyyn.

Nollamerkkien erityisen merkityksen takia, C-kielillä kirjoitetut jäsentimet saattavat pysähtyä ennenaikaisesti törmätessään nollamerkkiin. Käsittelytavasta riippuen, odottamattomat nollamerkit saattavat jopa aiheuttaa ohjelmiston kaatumisen. Nollamerkkien aiheuttamat ongelmat ovat niin yleisiä, että niille on määritetty oma CWE-luokkansa: ”Improper Neutralization of Null Byte or NUL Character”. [24] CWE (Common Weakness Enumeration) on yhteisökehitetty tietokanta yleisistä ohjelmiston sekä laitteiston haavoittuvuuksista [25].

Myös nykyisen SL-komentoparserin havaittiin pysähtyvän komennon keskellä sijaitsevaan nollamerkkiin, kuten havainnollistettu listauksessa 1, jossa `\0` indikoi nollamerkkiä.

Listaus 1. Esimerkki tulkkauksen pysähtymisestä nollamerkkiin.

```
SL@C=0,1,ABC\0DEF
OK
SL@C?
0,1,ABC
```

4.3 Kutsumerkin validointi

Kutsumerkki (engl. call sign) on radiolähettäjän tunnistamiseen käytetty uniikki aakkosnumeerinen tunniste. Kutsumerkit ovat erityisen tärkeitä julkisissa radiolähetyksissä, joissa esimerkiksi radioamatööreille kutsumerkin käyttö on pakollista. [26] Myös SATELin radiomodeemeihin on mahdollista määrittää kutsumerkki.

Jos kutsumerkkiominaisuus on käytössä, radiomodeemi lähettää tasaisin väliajoin sen asetuksiin määritetyn kutsumerkin. Lähetyksen aikaväliksi voidaan asettaa arvo yhden ja kolmenkymmenen minuutin väliltä, kun taas kutsumerkki on enintään kuudentoista merkin pituinen aakkosnumeerinen merkkijono. Nämä arvot voidaan asettaa muun muassa SL-komennolla `SL@C=`. [3]

`SL@C=` hyväksyy kolme pilkulla erotettua parametria: lähetyksen tilan, aikavälin ja kutsumerkin. Kuten muissakin käyttöliittymissä, näiden parametrien tulee noudattaa edellä määritettyjä raja-arvoja. Kuitenkin, jos aikavälin ja kutsumerkin väliin lisää ylimääräisen pilkun, kuten esitetty listauksessa 2, kutsumerkkiin hyväksytään virheellisesti myös ei aakkosnumeerisia merkkejä.

Listaus 2. Esimerkki virheellisen kutsumerkin hyväksymisestä

```
SL@C=0,1,@#$%
ERROR
SL@C=0,1,,@#$%
OK
SL@C?
0,1,,@#$%
```

5 Toteutus

Yksi parhaista tavoista varsinkin siirrettävän sulautetun ohjelmiston kehittämiseen, on toteuttaa valtaosa koodista natiivissa ympäristössä. Vaikka keskiverto ohjelmistokehittäjän työasema eroaa huomattavasti tyypillisestä sulautetusta järjestelmästä, juuri tämä perspektiivin muutos auttaa oikeanlaisten rajojen ja abstraktioiden määrittämisessä, mikä erottaa ohjelmiston laitteiston yksityiskohdista. Kun uusi ohjelmistomoduuli on todettu toimivaksi, se voidaan siirtää kohdelaitteistoon, sillä luottamuksella, että mahdolliset ongelmat rajoittuvat vain integraation aikana tehtyihin muutoksiin. [27]

Myös uusi SL-komentoparseri kehitettiin ja testattiin aluksi natiivissa ympäristössä. Tässä luvussa on käyty läpi toteutetun ohjelmistomoduulin ominaisuuksia yleisellä tasolla. Komentoparserin integroimista radiomodeemiin on kuvailtu luvussa 6.

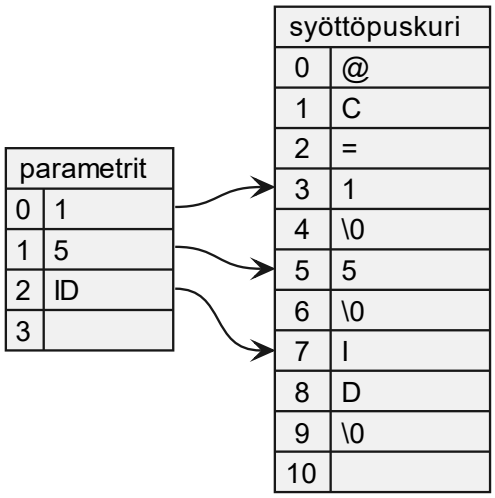
5.1 Datastruktuurit

Radiomodeemilla voi olla vain yksi SL-komentoparseri, mutta tällä komentoparserilla voi teoriassa olla mielivaltainen määrä samanaikaisesti toimivia portteja. Reaaliaikavaatimusten vuoksi, syötettä ei välttämättä saada luettua yhdellä kertaa, joten komentoparserilla on oltava jokaiselle portille oma syöttöpuskuri, johon syöte voidaan kerätä pala kerrallaan. On tietysti mahdollista, että sarjaportilta saapuva data ylittää syöttöpuskurin kapasiteetin, jolloin kyseinen komento hylätään ja komentoparseri jää odottamaan seuraavan paketin alkua. Tästä prosessista vastaa porttikohtainen tilakone, joka esitellään tarkemmin seuraavassa alaluvussa.

Sisäisesti komentoa esitetään struktuurina, johon on määritetty komennon nimi, parametrien erotinmerkki ja odotettu määrä, käyttöoikeustaso sekä osoitin käsittelyfunktioon. Tunnetut komennot on tallennettu yhteen muuttumattomaan taulukkoon, jossa komennon nimi toimii avaimena. Tästä syystä useammalla komennolla ei voi olla samaa nimeä, mutta joissain tapauksissa samaa

toimintoa voidaan käyttää eri komennoissa. Esimerkkinä identtisistä komennoista on salausavaimen hajautuksen palauttavat `SL%A?` ja `SL%K?` [3].

SL-komennon mahdolliset parametrit voidaan erotella tunnetun erotinmerkin avulla. Kuvassa 6 on havainnollistettu komentoparserin sisäistä esitystä syötteelle `SL@C=1,5,ID`. Syöttöpuskurin erottelu tapahtuu tallentamalla parametrien ensimmäisen merkin osoitteen ja korvaamalla erotinmerkin nollamerkillä.



Kuva 6. Sisäinen esitys komennolla `SL@C=1,5,ID`.

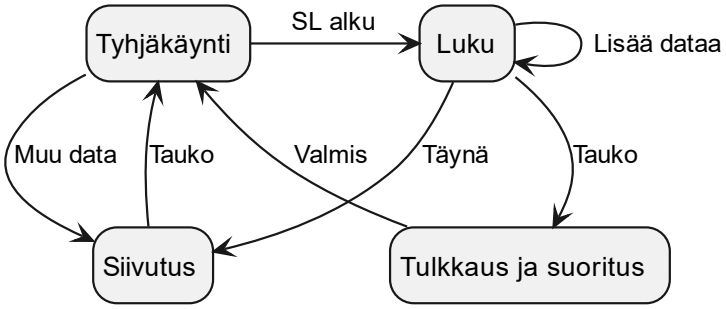
Osoitintaulukko on myös huomattavasti muistitehokkaampi ratkaisu, verrattuna erillisten kiinteäkokoisten puskurien varaamiseen jokaiselle mahdolliselle parametrille. Parametritaulukon koko määritellään käännösaikana, aiemmin esitellyn muokattavan otsikkotiedoston avulla.

5.2 Algoritmit

Ohjelma, joka tilansa mukaan käsittelee syötettään eri tavalla, toteutetaan usein äärellisenä tilakoneena. Äärellinen tilakone on algoritmi, joka voi nimensä mukaan olla äärellisessä määrässä ennalta määritettyjä tiloja. Kyseinen tila

määrittää kuinka järjestelmä reagoi syötteeseen, mihin tilaan se siirtyy seuraavaksi sekä mitä se tuottaa. Tilakoneet voidaan jakaa kahteen alakategoriaan: Mealy- ja Moore-koneisiin. Mealy-koneessa tuloste riippuu sekä nykyisestä tilasta että syötteestä, kun taas Moore-koneessa tuloste riippuu ainoastaan nykyisestä tilasta. Varsinkin dokumentaation kannalta, yksi tilakoneen suurimmasta hyödystä on logiikan graafinen esittäminen tilakaavioiden avulla. [28] Tästä syystä tilakoneet ovat suosittuja varsinkin sulautetuissa järjestelmissä.

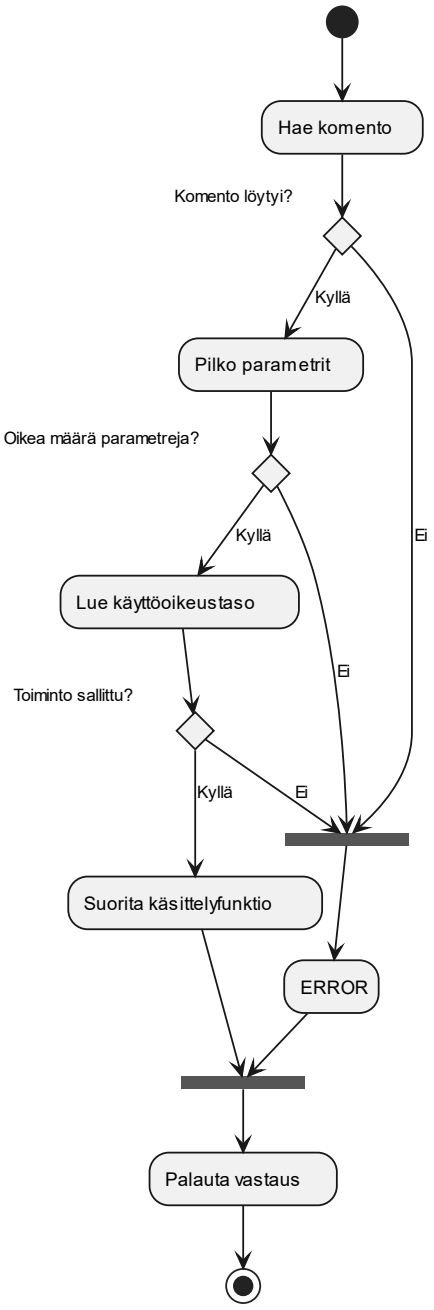
Kuten aiemmin todettiin, syötteen alustavasta käsittelystä vastaa porttikohtainen tilakone, jota on havainnollistettu kuvassa 7. Tätä käsittelyprosessia voidaan ajatella kierroksena, joka alkaa aina tyhjäkäyntitilasta, jossa komentoparseri odottaa seuraavan paketin alkua. Jos saapuva paketti ei ala SL-etuliitteellä, siirrytään siivutustilaan, jossa dataa ei kerätä vaan jäädään odottamaan paketin loppua, eli sarjaväylän taukoa. Jos taas kyseessä on SL-komento, siirrytään lukutilaan, jossa paketti kerätään komentoparserin sisäiseen puskuriin. Koska radioväylälle tarkoitettu data saattaa myös alkaa SL-etuliitteellä, SL-komennot voidaan myös kytkeä pois päältä [3].



Kuva 7. Yksinkertaistettu komentoparserin tilakaavio.

Lukutilan suojelee SL-komentoparseria ylivuodoilta siirtymällä siivutustilaan, jos syötteen pituus ylittää puskurin kapasiteetin. Käyttäjän näkökulmasta tämä ilmenee niin, että syöte jätetään huomiotta, eikä se myöskään lähde enää tässä vaiheessa radioväylälle. Jos SL-komento luettiin onnistuneesti, seuraavaksi aloitetaan komennon tulkkaus sekä mahdollinen suoritus, jota on havainnollistettu kuvan 8 vuokaaviossa. Suorituksen jälkeen tilakone palaa

tyhjäkäyntiin, jolloin komentoparseri on valmis vastaanottamaan seuraavan komennon ja käsittelyprosessi alkaa alusta.



Kuva 8. Komennon suorituksen vuokaavio.

Ensimmäinen askel tulkkauksessa on syöttöpuskurin vertaaminen tunnettujen komentojen taulukkoon. Kuten aikaisemmassakin SL-komentoparserissa, tällä hetkellä käytössä on kaikkein yksinkertaisin lineaarihaku, jonka suorituskyky on

suoraan riippuvainen komentotaulukon koosta. Jos se nähdään tarpeelliseksi, vaihtoehtoisten algoritmien vertailu on hyvä aihe jatkokehitykselle, sillä hakuprosessin muuttaminen voidaan toteuttaa ilman ympäröivään ohjelmaan vaikuttamista.

Komennon tunnistamisen jälkeen, mahdolliset parametrit erotellaan edellisessä alaluvussa kuvaillun prosessin mukaisesti. Koska myös parametrien minimi ja maksimi määrä tunnetaan, tulkkauksen aikana voidaan suorittaa yhtenäistä virheenkäsittelyä kaikille komennoille. Tämä vähentää toistuvaa koodia komentojen käsittelyfunktioissa.

Vielä lopuksi ennen käsittelyfunktioon siirtymistä luetaan radiomodeemin käyttöoikeustaso ja varmistetaan että käyttäjällä on oikeus suorittaa pyydetty toiminto. Jos jokin edellä mainituista vaiheista epäonnistuu, komentoparseri palauttaa `ERROR` vasteen, ilman toiminnon suorittamista.

6 Integraatio

Tähän asti uuden SL-komentoparserin ominaisuuksia on kyetty kuvailemaan ilman vuorovaikutusta varsinaisen kohdelaitteen kanssa. Tässä luvussa käydään läpi alustava integraatioprosessi yhteen SATELin radiomoduuleista, mikä mahdollisti myös suorituskyvyn ja muistin käytön vertailun.

Tyypillisessä sulautettujen järjestelmien kehitysympäristössä on kolme pääasiallista osapuolta: isäntälaite, kohdelaite sekä näiden kahden välissä toimiva, yksi tai useampi kommunikaatioväylä. Tämänkaltaista järjestelyä kutsutaan alustojen väliseksi kehitykseksi (engl. cross-platform development). Isäntälaitteen tärkeimpiin ohjelmistoihin kuuluu ristikääntäjä sekä lähdetason debuggeri, eli virheen jäljitin, joka kommunikoi kohdelaitteiston kanssa ohjelmointilaitteen tai debuggerin (engl. debug probe) avulla. [15]

Alustojen välisen ohjelmistokehityksen mahdollistaa viime kädessä ristikääntäjä. Tavallinen natiivi kääntäjä tuottaa objektikoodia, joka toimii isäntälaitteen prosessoriarkkitehtuurissa, kun taas ristikääntäjän tuotos on tehty suoritettavaksi täysin erilaisella laitteistolla. Ristikääntäjää käytetään tilanteissa, jossa kohdelaite ei kykene isännöimään omaa kääntäjäänsä, tai jos suoraan kohdelaitteistolla kehittäminen todetaan liian hitaaksi. [15] Hyvä esimerkki sulautetusta laitteesta, joka kykenee kääntämään omat ohjelmansa, on varsinkin harrastekäytössä suosittu Raspberry Pi korttitietokone.

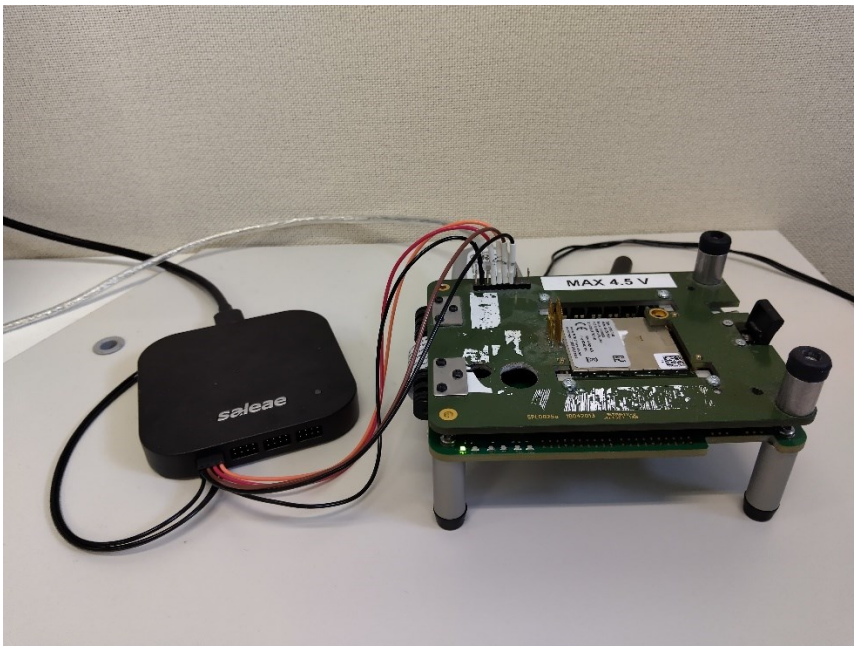
Prosessorit ja ihmiset lukevat ohjelmistoa hyvin erilaisella tavalla. Lähdetason debuggerilla on kuitenkin mahdollista yhdistää suoritettavan ohjelman tila takaisin sen tuottaneeseen lähdekoodiin. Natiivissa kehityksessä virheenkorjaus tapahtuu ohjelmistopohjaisesti, mutta alustojen välisessä kehityksessä tähän tarvitaan myös kohdelaitteiston yhteistyötä. Debuggaus-ympäristön mahdollistamiseksi isäntä- ja kohdelaitteen on keskusteltava ennalta määritetyn kommunikaatioprotokollan avulla [15]. Ehkä yksi yleisimmistä debuggerin toiminnoista on pysäytyspisteiden (engl. break point) asettaminen. Pysäytyspisteen kohdatessa ohjelmisto pysähtyy automaattisesti, jonka jälkeen

suoritusta voidaan askeltaa rivi kerrallaan. Tällöin voidaan myös lukea muuttujien sisältö, tai jopa muokata niitä.

6.1 Käytetyt työkalut

Useita ohjelmistoja sekä laitteistoja käytettiin uuden SL-komentoparserin alustavassa integraatiossa, joista osa on myös nähtävillä kuvassa 9. Näistä keskeisin on SATELin radiomodeemien kehittämiseen käytetty ympäristö, joka tarjoaa koodieditorin lisäksi muun muassa lähdetason debuggerin. Kehitysympäristöstä uusi laiteohjelmisto siirretään radiomodeemiin ohjelmointilaitteen avulla.

Kuvassa 9 radiomodeemiin sarjaporttiin on myös kytketty Saleae logiikka-analysaattori, joka tuotti alaluvussa 6.3 käytettyä ajoitusdataa. Saleae logiikka-analysaattori on pieni, mutta tehokas työkalu signaalien tallentamiseen digitaalisissa piireissä. Tallennettuja signaaleja voidaan jälkikäteen havainnollistaa ajoituskaavion muodossa Saleaen Logic 2 ohjelmistolla, joka kykenee purkamaan useita digitaalisia protokollia. [29]

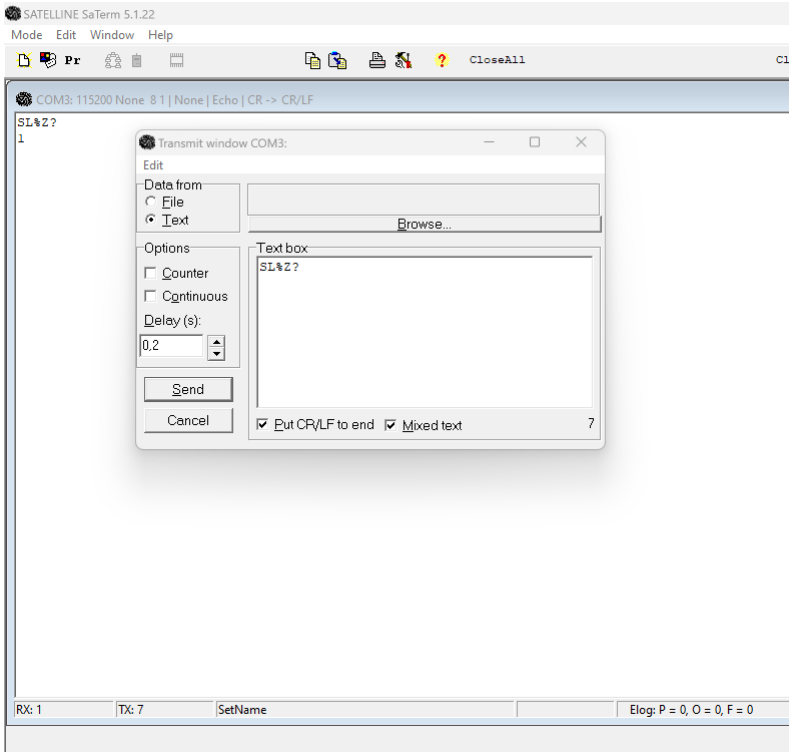


Kuva 9. Saleae logiikka-analysaattori ja SATEL TR4+.

Uusi SL-komentoparseri integroitiin ensimmäisenä SATEL TR4+ radiomoduulin. SATEL TR4+ on kompakti ja energiatehokas UHF-taajuusalueen (ultra high frequency) lähetin-vastaanotinmoduuli, joka on suunniteltu integroitavaksi varsinkin akkukäyttöisiin ja pienikokoisiin järjestelmiin [3].

6.2 Vanhan toteutuksen korvaaminen

Koodieditorin valinta riippuu pitkälti kehittäjän omista mieltymyksistä, mutta merkittävän ohjelmistokomponentin korvaaminen isossa koodikannassa vaatii kyvyn seurata viittauksia ohjelmistomoduulien välillä. Tässä vaiheessa pieniä muutoksia ja lisäyksiä oli tehtävä uuden SL-komentoparserin rajapintaan, jotta integraatio onnistui ilman merkittäviä muutoksia muihin laiteohjelmiston moduuleihin. Vanhaan komentoparseriin oli kuitenkin viittauksia melkein kymmenestä eri moduulista, jotka kaikki oli päivitettävä integraatioprosessin aikana. Kuvassa 10 uusi komentoparseri vastaa ensimmäistä kertaa SATELLINE SaTermissä, alustavan testauksen aikana.



Kuva 10. Uusi SL-komentoparseri vastaa ensimmäistä kertaa TR4+ radiomoduulissa.

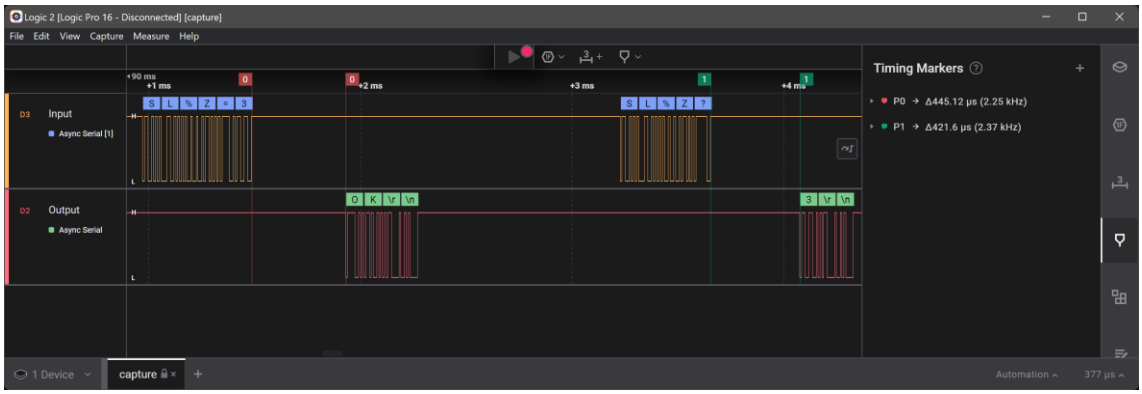
Laitteen asetuksia hallitseva komentoparseri on aina jollain tasolla riippuvainen muista ohjelmistomoduuleista, mutta jotkin näistä moduuleista ovat myös riippuvaisia komentoparserista. Esimerkiksi, SL-komentoparseri hallitsee radiomodeemin ohjelmointitilaa, jonka aikana useiden järjestelmän osien on toimittava ennalta määritetyllä tavalla. Integraatioprosessin aikana, ohjelmointi- ja komentotilan sekoittuminen johti vaikeaselkoiseen käyttäytymiseen, jossa aiemmin esitellyt NMS-komennot lakkasivat toimimasta eivätkä hallintaohjelmistot enää kyenneet kommunikoimaan radiomodeemin kanssa. Tämä on hyvä esimerkki kattavan toiminnallisuustestauksen tärkeydestä.

6.3 Suorituskyky

Reaaliaikaisessa järjestelmässä suorituskykyvaatimuksiin vaikuttaa varsinkin aikarajojen määrä sekä pituus. Vaikka laitteistolla on usein päärooli suorituskyvyssä, siihen voidaan vaikuttaa myös ohjelmistolla. Joskus

suorituskykyä mitataan ohjelmistoon lisätyillä aikaleimoilla, mutta kattavammat tulokset vaativat koko järjestelmän suorituskyvyn mittaamisen. [15]

Uuden SL-komentoparserin suorituskykyä mitattiin aiemmin esitellyn logiikka-analysaattorin avulla. Kuvassa 11 on tallenne sarjalinjoilta, jossa nähdään kuinka päätelaite lähettää SL-komentoja ja radiomodeemi vastaa niihin. Logic 2 ohjelmisto tulkkaa sarjaliikenteen ja kirjaa pakettien ylle helpommin luettavat merkkijonot. Syötteen ja vasteen välistä aikaa voidaan mitata jälkikäteen lisättyjen ajoitusmerkkien avulla.



Kuva 11. Kuvankaappaus Saleae Logic 2 ohjelmistosta.

Luvussa 3 todettiin, että vanhassa SL-komentoparserissa on lähes 200 komentoa, joista kaikkia ei ehditty toteuttamaan tämän opinnäytetyön aikana. Uuden ja vanhan komentoparserin vertaileminen keskenään tuottaisi siis puolueellisia tuloksia. Tästä syystä vanhasta komentoparserista kehitettiin vain tähän vertailuun käytetty pelkistetty versio, joka vastasi toiminnoiltaan uuden komentoparserin sen hetkistä tilaa. Lyhyyden vuoksi tässä alaluvussa tätä kutsutaan testiversioksi.

Jos suoritusajaksi määritellään vasteen ensimmäisen tavun ja komennon viimeisen tavun välinen erotus, sekä uudessa että testiversiossa useimmat komennot suoritetaan noin puolessa millisekunnissa. Taulukossa 3 uuden komentoparserin suorituskyky vaikuttaa jopa marginaalisesti heikentyneen, mutta tämän voidaan olettaa johtuvan lisääntyneestä virheenkäsittelystä, joka itsessään vaatii enemmän prosessointiaikaa. Vanhan komentoparserin

maksimiarvo havainnollistaa kuinka radiomodeemin taustaprosessointi vaikuttaa SL-komennon suoritusaikaan. Ilman näitä poikkeamia, todenmukaisempi keskimääräinen suoritusaika on 700 - 800 µs välissä. Toteutettujen komentojen määrällä on siis selkeä, vaikkakin ihmisen näkökulmasta erottumaton, vaikutus SL-komentoparserin suorituskykyyn.

Taulukko 3. Alustava suorituskyky vertailu SL-komentoparserien välillä.

Versio	Min (µs)	Max (µs)	Keskiarvo (µs)
Vanha	704	323 376	40 684
Testi	350	520	437
Uusi	424	670	486

6.4 Muistin käyttö

Sulautetun järjestelmän fyysinen koko sekä laitteiston kustannusrajoitteet vähentävät saatavilla olevan järjestelmämuistin määrää. Järjestelmämuistin saatavuus puolestaan rajoittaa ohjelmiston kokoa, jolloin muistin jalanjäljestä tulee yhä merkittävämpi ominaisuus. Järjestelmävaatimusten täyttämiseksi sekä dynaamisen että staattisen muistin käyttöä on tarkkailtava laiteohjelmistoa kehittäessä. [15]

Kuten valtaosassa TR4+ laiteohjelmistoa, SL-komentoparserissa käytetään ainoastaan staattista muistia. Ohjelmiston muistin tarpeen voi selvittää linkkerin tuottamasta .map tiedostosta (engl. linker map file), johon on kirjattu muun muassa käännetyn ohjelmiston koko. SL-komentoparserissa kaksi suurinta muistin käyttöön vaikuttavaa tekijää ovat porttikohtaiset puskurit sekä tunnettujen komentojen taulukko. Taulukossa 4 on vertailtu komentoparserien muistin käyttöä.

Taulukko 4. Muistin käytön vertailu SL-komentoparserien välillä.

Versio	Koodi (kB)	Data (kB)
Vanha	27.41	2.83
Testi	3.88	0.41
Uusi	3.41	0.51

Taulukoista 3 ja 4 voidaan päätellä, että suorituskyvyssä sekä muistin käytössä ei ilmennyt tässä vaiheessa merkittäviä muutoksia. Puuttuvien toimintojen lisäämisen jälkeen näitä arvoja on syytä tutkia uudestaan, jotta nähdään kuinka suorituskyky sekä muistin käyttö skaalautuu toimintoihin nähden, uudessa SL-komentoparserissa.

7 Testaus ja dokumentaatio

Hyvin suunniteltu sulautettu ohjelmisto on testattavissa sekä kohdelaitteistolla että sen ulkopuolella [30]. Itse asiassa vain muutamalla rivillä koodia on mahdollista saada aikaan omiin tarpeisiin mukautettu testikehys. Testauksessa tärkeintä on testien suorittaminen, eikä niiden rakentamiseen käytetty ohjelmisto. [31] Tässä luvussa käydään läpi SL-komentoparserin testausta sekä kuvaillaan uuden ohjelmistomoduulin dokumentaatioprosessia.

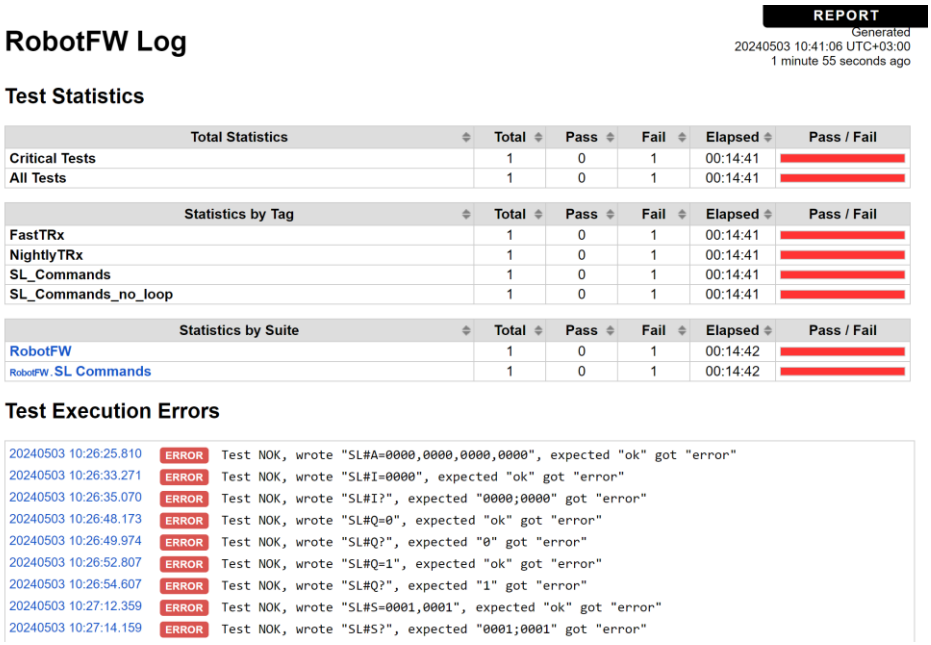
7.1 Testaus

Ennen kohdelaitteistoon siirtymistä, uuden SL-komentoparserin alustavaan testaukseen käytettiin muun muassa Unity testauskirjastoa, jota ei tule sekoittaa Unity-pelimoottoriin. Unity testitiedosto on tavallinen C-ohjelma, jossa käytetään assert-makroja varmistamaan ohjelmistokomponentin oikeanlainen toiminta. Unity pitää kirjaa ajettujen testien määrästä sekä lopputuloksesta, ja tuottaa näiden perusteella ihmisen luettavissa olevan testiraportin. [32]

Monien muiden ominaisuuksien tavoin, SL-komentoja käydään läpi myös radiomodeemien lopullisen toiminnallisuustestauksen aikana. Tällöin koko laiteohjelmistoa ajetaan testipenkeissä Robot Framework hallinnan alla. [33] Robot Framework on avoimen lähdekoodin automaatio- ja testauskehys, jonka keskeisin ominaisuus on sen oma käyttäjäystävällinen tiedostomuoto, jota käytetään prosessien määrittämiseen [34]. Näitä radiomodeemien testejä voidaan ajaa viikonloppuisin, öisin tai heti versionhallinnan muutosten jälkeen. Jenkins vastaa sopivan ajoituksen valinnasta, testin työmäärän mukaan. [33] Jenkins on avoimen lähdekoodin automaatiopalvelin, joka mahdollistaa sekä pienien että suurien ohjelmistoprojektien automaattisen rakentamisen sekä käyttöönoton [35].

Edellä mainittuja Robot Framework testejä ajettiin myös varsinaisten testipenkkien ulkopuolella tämän opinnäytetyön aikana. Radiomodeemin testit toimivat itsenäisesti, mutta ennen niiden suorittamista on

konfiguraatiotiedostoon määriteltävä muutamia testausympäristön parametrejä, kuten oikea sarjaportti ja sen alustavat asetukset. SL-komentotestit koostuvat lähinnä CSV-tiedostoista (Comma-Separated Values), joihin on määritelty testin kuvaus, suoritettava komento sekä odotettu vaste. Näitä taulukoita käy läpi mukautettu Robot Framework kirjasto, joka kirjoittaa sarjaväylälle oikean komennon ja tekee vasteen perusteella päätöksen testin tuloksesta. Tällaista työnkulkua kutsutaan datalähtöiseksi testaukseksi. Kuten kuvassa 12 näkyy, yleisempien SL-komentojen testi voi kestää jopa 15 minuuttia, joka ei täten sovi ajettavaksi jokaisen komennon toteutuksen välissä.



RobotFW Log

REPORT
Generated
20240503 10:41:06 UTC+03:00
1 minute 55 seconds ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	0	1	00:14:41	
All Tests	1	0	1	00:14:41	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
FastTRx	1	0	1	00:14:41	
NightlyTRx	1	0	1	00:14:41	
SL_Commands	1	0	1	00:14:41	
SL_Commands_no_loop	1	0	1	00:14:41	

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
RobotFW	1	0	1	00:14:42	
RobotFW.SL Commands	1	0	1	00:14:42	

Test Execution Errors

```
20240503 10:26:25.810 ERROR Test NOK, wrote "SL#A=0000,0000,0000,0000", expected "ok" got "error"
20240503 10:26:33.271 ERROR Test NOK, wrote "SL#I=0000", expected "ok" got "error"
20240503 10:26:35.070 ERROR Test NOK, wrote "SL#I?", expected "0000;0000" got "error"
20240503 10:26:48.173 ERROR Test NOK, wrote "SL#Q=0", expected "ok" got "error"
20240503 10:26:49.974 ERROR Test NOK, wrote "SL#Q?", expected "0" got "error"
20240503 10:26:52.807 ERROR Test NOK, wrote "SL#Q=1", expected "ok" got "error"
20240503 10:26:54.607 ERROR Test NOK, wrote "SL#Q?", expected "1" got "error"
20240503 10:27:12.359 ERROR Test NOK, wrote "SL#S=0001,0001", expected "ok" got "error"
20240503 10:27:14.159 ERROR Test NOK, wrote "SL#S?", expected "0001;0001" got "error"
```

Kuva 12. Robot Framework testiloki.

Tällä hetkellä valtaosa testivirheistä johtuu puuttuvista komennoista, jolloin päätelaite vastaanottaa `ERROR` vasteen, kun komentoparseri ei kykene tunnistamaan lähetettyä komentoa. Kuvassa 12 Robot Framework raportoi sataprosenttisen epäonnistumisen, koska kaikki yleisimmät komennot tarkistetaan yhdellä testillä.

7.2 Dokumentaatio

Sulautetuissa järjestelmissä, sekä laitteiston että ohjelmiston puolella, on usein vakioituneita, uudelleenkäytettäviä komponentteja, jolloin suunnitteluprosessin ei tarvitse aina alkaa puhtaalta pöydältä. Tekniseen dokumentaatioon ei kuitenkaan aina kiinnitetä tarpeeksi paljon huomiota. Dokumentaatio voi olla liian monimutkaista tai se saattaa jäädä jälkeen projektin edetessä. Nämä molemmat voivat johtaa huomattaviin suunnitteluvirheisiin. [36]

Uuden SL-komentoparserin dokumentaatio toteutettiin Doxygen dokumentaatio generaattorilla, josta on muodostunut de facto -standardi työkalu varsinkin C-kieliperheen projekteille [37]. Doxygen soveltuu erityisesti dokumentaatio koodina -menetelmän käyttöön, jolla tarkoitetaan dokumentaation ja ohjelmistokoodin kirjoittamista samoilla työkaluilla [38]. Ohjelmistokomponenttien dokumentaatio voidaan poimia suoraan lähdekoodista, mikä helpottaa huomattavasti dokumentaation ajantasaistamista. Doxygen kykenee myös visualisoimaan koodielementtien yhteyksiä automaattisten riippuvuus- ja yhteistyökaavioiden avulla. [37]

Dokumentaation kaavioihin käytettiin PlantUML työkalua, joista osa on myös esitetty tässä opinnäytetyössä. Omaa täsmäkieltään hyödyntäen, PlantUML mahdollistaa monipuolisten kaavioiden nopean piirtämisen [39]. Useiden ulkoisten työkalujen tavoin, tuki PlantUML-kaavioille on sisäänrakennettu Doxygen dokumentaatio generaattoriin [37].

Myös SL-komennot vaativat dokumentointia, joka jaetaan usein liitteenä SATELin radiomodeemin ohjekirjan mukana, taulukko muodossa [3]. Useimmiten nämä taulukot ovat käsin täytettyjä, joka on varsinkin erilaisten tuotevarianttien kasvaessa hyvin virhealtis prosessi. Tämän vuoksi SL-komentojen dokumentaatio jää helposti jälkeen toteutuksen muuttuessa, johtaen sekä käyttäjien että kehittäjien turhautumiseen. SL-komentoparserin dokumentaatiota ei haluta julkaista kokonaisuudessaan, mutta sen osia voidaan silti hyödyntää SL-komentojen dokumentaation ajantasaistamisessa.

8 Pohdinta ja jatkokehitys

Tämän opinnäytetyön tavoitteena oli aloittaa uuden SL-komentoparserin kehitys. SL-komentoja käytetään SATELin radiomodeemien ohjelmalliseen sekä interaktiiviseen hallintaan. Nykyisessä komentoparserissa on kuitenkin useita ongelmia, jonka vuoksi se on päätetty kirjoittaa uudestaan.

Tässä opinnäytetyössä syntyi uusi SL-komentoparseri, jota voidaan jatkokehityksen jälkeen hyödyntää SATELin radiomodeemeissa. Perusominaisuuksien lisäksi, uuteen komentoparseriin toteutettiin noin 50 komentoa. Uusi komentoparseri integroitiin myös alustavasti yhteen SATELin radiomodeemeista. Integraation jälkeen vertailtiin uuden ja vanhan komentoparserin suorituskykyä. Lopuksi uuden komentoparserin toimintaa tarkasteltiin myös SATELin testiympäristössä.

Keskeisin aihe jatkokehitykselle on puuttuvien komentojen toteuttaminen. Vaadittujen toimintojen lisäämisen jälkeen on myös suoritettava kattavampaa toiminnallisuustestausta. Testauksella varmistetaan taaksepäin yhteensopivuus sekä uuden komentoparserin vaikutukset muihin ohjelmistomoduuleihin. Uuden komentoparserin on myös toimittava jo kentällä käytetyissä radiomodeemeissa. Käyttöönotto voidaan toteuttaa porrastetusti, aloittaen vähiten kriittisistä laitteista. Vanhimpien radiomodeemien ohjelmistoihin ei kuitenkaan tulla tekemään merkittäviä muutoksia.

Radioteknologialla on tärkeä osa sekä suurissa että pienissä kaupallisissa järjestelmissä. Tästä syystä, useimmat radiolaitteet tarvitsevat ohjelmallisen hallinnan mahdollistavan käyttöliittymän. Tämä opinnäytetyö havainnollistaa hyvin suunnitellun ja ylläpidettävän rajapinnan tärkeyttä.

9 Lähdeluettelo

[1] Federal Communications Commission, ”Industrial / Business,”. Saatavilla: https://www.fcc.gov/wireless/bureau-divisions/mobility-division/industrial-business [Haettu 23.4.2024].

[2] SATEL Oy, ”Radioteknologia varmistaa luotettavat yhteydet,”. Saatavilla: https://www.satel.com/fi/tuotteet/ [Haettu 10.4.2024].

[3] SATEL Oy, *SATEL-TR4+_R4+ Integration guide,* 2.4, 2022.

[4] SATEL Oy, ”Me olemme SATEL,”. Saatavilla: https://www.satel.com/fi/me-olemme-satel/ [Haettu 24.2.2024].

[5] SATEL Oy, ”Technological research in extreme harsh environment,”. Saatavilla: https://www.satel.com/wp-content/uploads/2022/12/SATEL-casestory-Mount-Everest-2.pdf [Haettu 19.5.2024].

[6] SATEL Oy, ”Safe and efficient airport operations,”. Saatavilla: https://www.satel.com/wp-content/uploads/2023/09/SATEL_casestory_Vaisala_lowres-1.pdf [Haettu 19.5.2024].

[7] E. Smith, ”The Hayes Code,” 22.2.2023.Saatavilla: https://tedium.co/2023/02/22/early-modem-technology-history/ [Haettu 3.3.2024].

[8] E. Smith, ”The Squeal of Data,” 28.6.2023.Saatavilla: https://tedium.co/2023/06/28/teletype-computer-evolution-history/ [Haettu 24.4.2024].

[9] Hayes Microcomputer Products. Inc., *Hayes Stack Smartmodem Owner's Manual,* 3, 1981.

[10] *A complete list of things to know about 2400bps modems.,*10, 1986, p. 27.

[11] ITU-T, *Serial asynchronous automatic dialling and control,*V.250, 2003.

[12] D. Tian, G. Hernandez, J. Choi, V. Frost, C. Ruales, K. Butler, P. Traynor, H. Vijayakumar, L. Harrison, A. Rahmati ja M. Grace, ”ATtention Spanned: Comprehensive Vulnerability Analysis of AT Commands Within the Android Ecosystem,” *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, 2018.

[13] J. Spolsky, ”Things You Should Never Do, Part I,” 6.4.2000.Saatavilla: https://www.joelonsoftware.com/2000/04/06/things-you-should-never-do-part-i/ [Haettu 21.4.2024].

[14] R. Oshana, ”Introduction to embedded and real-time systems,”. Saatavilla: https://profile.iiita.ac.in/bibhas.ghoshal/lecture_slides_embedded/embeddedsystemcomponentsnotes1.pdf [Haettu 19.5.2024].

[15] C. Yao ja Q. Li, Real-Time Concepts for Embedded Systems, CMP Books, 2003.

[16] M. S. Kirkpatrick, ”Parallelism vs. Concurrency,”. Saatavilla: https://w3.cs.jmu.edu/kirkpams/OpenCSF/Books/csf/html/ParVConc.html [Haettu 13.4.2024].

[17] Ttd65, ”Parallel vs concurrent,” 3.6.2021.Saatavilla: https://upload.wikimedia.org/wikipedia/commons/6/6f/Parallel-concurrent.png [Haettu 9.5.2024].

[18] R. Stallman ja Z. Weinberg, *The C Preprocessor,* 13.2.0

[19] I. Wienand, ”Implementing abstraction with C,” *Computer Science from the Bottom Up*, pp. 15-19.

[20] Y. Yang, ”The Secret World of Newline Characters,” 19.6.2018.Saatavilla: https://www.enigma.com/resources/blog/the-secret-world-of-newline-characters [Haettu 13.4.2024].

[21] SATEL Oy, *SATEL NMS PC User Manual,* 2.7

[22] SATEL Oy, *SATELLINE SaTerm Software User Guide,* 2.1

[23] SATEL Oy, ”SATELLINE SaTerm,”. Saatavilla: https://www.satel.com/wp-content/uploads/2021/02/Saterm.jpg [Haettu 14.4.2024].

[24] E. Poll, Secure Input Handling, 1.0, Nijmegen, 2023.

[25] MITRE Corporation, ”New to CWE,”. Saatavilla: https://cwe.mitre.org/about/new_to_cwe.html [Haettu 13.4.2024].

[26] ITU, ”Terrestrial Frequently Asked Questions (FAQ),”. Saatavilla: https://www.itu.int/en/ITU-R/terrestrial/Pages/by-categories-faq.aspx?categorizedby=15 [Haettu 27.4.2024].

[27] Embedded Artistry, ”Writing Portable Embedded Software,”. Saatavilla: https://embeddedartistry.com/wp-content/uploads/2020/08/Playbook-Writing-Portable-Software.pdf [Haettu 11.4.2024].

[28] J. L. Garbini, ”Finite State Machines in Embedded Applications,”. Saatavilla: http://courses.washington.edu/mengr477/resources/StateMachine.pdf [Haettu 19.5.2024].

[29] Saleae, ”Setup,”. Saatavilla: https://support.saleae.com/getting-started/setup [Haettu 24.4.2024].

[30] R. C. Martin, ”Clean Architecture: A Craftsman's Guide to Software Structure and Design,” Prentice Hall, 2018, p. 265.

[31] Jera Design LLC, ”JTN002 - MinUnit -- a minimal unit testing framework for C,”. Saatavilla: https://jera.com/techinfo/jtns/jtn002 [Haettu 5.4.2024].

[32] Throw The Switch, ”UNITY - Unit Testing for C (especially Embedded Software),”. Saatavilla: https://www.throwtheswitch.org/unity [Haettu 23.3.2024].

[33] J. Österholm, ”Radiomodeemin testiasetusten tuottaminen graafisen käyttöliittymän avulla,” 2021.Saatavilla: https://urn.fi/URN:NBN:fi:amk-2021052611352 [Haettu 19.5.2024].

[34] ”Robot Framework,”. Saatavilla: https://robotframework.org/ [Haettu 17.4.2024].

[35] ”Jenkins - Build great things at any scale,”. Saatavilla: https://www.jenkins.io/ [Haettu 17.4.2024].

[36] B. Muranko ja R. Drechsler, ”Technical Documentation of Software and Hardware,” *IFIP International Conference on Very Large Scale Integration*, 2006.

[37] D. v. Heesch, ”Doxygen - Manual for version 1.10.0,”. Saatavilla: https://www.doxygen.nl/files/doxygen_manual-1.10.0.pdf.zip [Haettu 29.3.2024].

[38] E. Holscher, ”Docs as Code,”. Saatavilla: https://www.writethedocs.org/guide/docs-as-code/ [Haettu 31.3.2024].

[39] ”PlantUML at a Glance,”. Saatavilla: https://plantuml.com/ [Haettu 17.4.2024].