TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

VALTTERI KENKKILÄ

ATLAS-OHJELMOINTIKIELEN LAAJENTAMISMAHDOLLISUUDET

Diplomityö

Tarkastaja: Hannu-Matti Järvinen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekunta-neuvoston kokouksessa 13. tammi-kuuta 2016

TIIVISTELMÄ

VALTTERI KENKKILÄ: ATLAS-ohjelmointikielen laajentamismahdollisuudet
Tampereen teknillinen yliopisto
Diplomityö, 45 sivua, 10 liitesivua
Tammikuu 2016
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Hannu-Matti Järvinen

Avainsanat: ATLAS, FEP, OpenVMS, ALPHA

Tässä diplomityössä selvitetään Insta Integrated Logistics Support (ILS) Oy:n mahdollisuudet tuottaa Abbreviated Test Language for All Systems (ATLAS) -ohjelmointikielen laajennusosia toisia ohjelmointikieliä käyttäen. ATLAS-ohjelmointikieltä käytetään eri Consolidated Automated Support System (CASS) -testausasemien laitteiden ohjaamiseen. CASS-asemia käytetään useiden sähköisten järjestelmien testaamiseen. ATLASin ominaisuudet testattavalta laitteelta saatavan datan analysointiin ovat heikot ja tästä syystä laajennusosien käyttö helpottaisi testausohjelmien laadintaa huomattavasti.

CASS asemia on käytössä kahta eri mallia ja samoja laajennusosia halutaan suorittaa molemmissa asemissa. Laajennusosien tuottaminen molemmille asemille pyritään tekemään mahdollisimman tehokkaasti ja samaa ohjelmointikieltä käyttäen.

Diplomityössä selvitetään myös toisen CASS-asematyypin virtualisoinnin mahdollisuuksia. Virtualisoinnilla saavutettaisiin huomattavaa etua, mikäli ALPHA-arkkitehtuurilla toteutettu CASS-asema voidaan korvata.

ABSTRACT

VALTTERI KENKKILÄ: Extensions of ATLAS programming language
Tampere University of Technology
Master of Science Thesis, 45 pages, 10 Appendix pages
January 2016
Master's Degree Program in Information Technology
Major: Software Engineering
Examiner: Professor Hannu-Matti Järvinen

Keywords: ATLAS, FEP, OpenVMS, ALPHA

This Master's thesis conducts research on the ability of Insta ILS Oy to produce extension applications for Abbreviated Test Language for All Systems (ATLAS). ATLAS is a programming language that is used for controlling devices on Consolidated Automated Support System (CASS). CASS-stations are used for testing electrical devices and systems in aviation. Abilities of ATLAS are weak when analysing data from tested devices and extensions applications would ease analysing process considerably.

Two types of CASS-stations are utilized and the use of same extension application is preferred for both stations. Extension applications are intended to be efficiently produced and use the same programming language for both CASS-stations.

This Master's thesis also analyses possibilities of virtualizing a CASS-station. Virtualization would bring huge advantage, if it can be used to replace ALPHA-architecture that is used in CASS-station.

ALKUSANAT

Tutkielmassa käytetyt ohjelmointikielet sekä käyttöjärjestelmä olivat minulle ennestään täysin tuntemattomia. Opin näistä aiheista suunnattoman paljon työtä tehdessäni ja onnistuin myös hyödyntämään oppimaani työssäni.

Haluan kiittää työnantajaani Insta ILS Oy:tä saamastani mahdollisuudesta tehdä diplomityöni yrityksessä. Kiitän myös työkavereitani tuesta ja antamastanne avusta. Erityisesti haluan kiittää työn ohjaajaa Hannu Pohjanpäätä sekä kaikkia huoneen H2136 henkilöitä.

Tampereella, 31.1.2016

Valtteri Kenkkilä

SISÄLLYSLUETTELO

KUVALUETTELO

TAULUKKOLUETTELO

LYHENTEET JA MERKINNÄT

ATLAS	Engl. Abbreviated Test Language for All Systems, Standardoitu ohjelmointikieli avioniikkalaitteiden testaamiseen.
C/ATLAS	Engl. Common/Abbreviated Test Language for All Systems, joissakin standardeissa käytetty nimitys ATLAS-ohjelmointikielestä.
CASS ATLAS	CASS-asemille kehitetty ATLAS-kieli.
CASS	Engl. Consolidated Automated Support System, avioniikkalaitteiden testaamisessa käytettävä testausasema.
CISC	Engl. Comlex Instruction Set Computing, ohjelmoinnissa käytettävä käskykanta.
DHCP	Engl. Dynamic Host Configuration Protocol, IP-osotteiden jakamiseen käytettävä protokolla.
Ethernet	Yleisesti käytössä oleva tietoliikennetekniikka.
FEP	Engl. Functional Extension Program, ATLAS-ohjelmointikielen laajennusosa.
FTP	Engl. File Transfer Protocol, sovelluskerroksen tiedonsiirtoprotokolla.
GPI	Engl. General Purpose Interface, liityntärajapintä CASS-asemaan.
GZIP	Tiedostojen paketointimenetelmä.
ID	Engl. Interface Device, liityntälaite CASS-aseman GPI:n ja testattavan laitteen välillä.
IDPC	Engl. Interface Device Personal Computer, ID:n sisään lisätty PC.
ILS	Engl. Integrated Logistics Support, yksi Instan osakeyhtiöistä.
LRL	Engl. Longest Record Length, pisin muuttuja järjestelmässä.
LTS	Engl. Long Time Support, Ubuntu-käyttöjärjestelmän versio, jonka tuki on taattu kolmeksi vuodeksi julkaisuajankohdasta eteenpäin.
MFCASS	Engl. Main Frame Consolidated Automated Support System, avioniikkalaitteiden testaamisessa käytettävä testausasema.
MIL-STD-1553	Yhdysvaltain puolustusministeriön kehittämä sotilasstandardi joka määrittelee sarjaliikenneprotokollan.
Mittalaite	CASS-aseman sisällä oleva laitteisto jolla mitataan testilaitteen ominaisuuksia tai syötetään testilaitteelle signaalia.
MRS	Engl. Maximum Record Size, muuttujan maksimipituus järjestelmässä.
MTPSI	Engl. Master Test Program Set Index, testiohjelmassa käytettävän laitteiston määrittelevä tiedosto.
PC	Engl. Personal Computer, tietokone.
RISC	Engl. Reduced Instruction Set Computing, ohjelmoinnissa käytettävä käskykanta.
RS232	Sarjaliikenneprotokolla.
RS422	Sarjaliikenneprotokolla.
RTCASS	engl. Reconfigurable Transportable Consolidated Automated Support System, avioniikkalaitteiden testaamisessa käytettävä testausasema.
SMBus	System Management Bus, yksinkertainen alhaisella kaistanleveydellä toimiva kommunikaatioväylä.
SSH	Engl. Secure Shell, kryptattu tietoliikenneprotokolla.

Tar-paketti	Yhdeksi tiedostoksi koottu tiedostokokoelma.
TCP/IP	Eengl. Transmission Control Protocol/Internet Protocol, yhteydellinen tietoliikenneprotokolla.
TELNET	Sovelluskerroksen tiedonsiirtoprotokolla.
Testilaite	CASS-asemalla testattava laite tai järjestelmä.
TGZ-paketti	Gzip pakattu Tar-paketti.
TPS	Engl. Test Program Set, testauksessa käytettävän laitteiston ja ohjelmiston kokonaisuus.
TTY	Tampereen teknillinen yliopisto.
USB	Engl. Universal Serial Bus, yleisesti käytössä oleva kommunikaatioprotokolla.
UUT	Engl. Unit Under Testing, CASS-asemalla testattava laitteisto.
XML	Engl. Extensible Markup Language.

1. JOHDANTO

CASS-asemia (engl. Consolidated Automated Support System) käytetään avioniikka-laitteiden testaamiseen ja mahdollisten vikojen paikantamiseen testattavassa laitteessa. Vikojen paikantamista voidaan suorittaa CASS-asemalla aina komponentin tasolle asti. CASS-asemalla voidaan syöttää testattavalle laitteelle esimerkiksi ennalta määriteltyjä jännitteitä sekä virtoja ja mitata testattavasta laitteesta saatavia vasteita. CASS-asemalla voidaan myös kommunikoida testattavan laitteen kanssa eri sarjaliikenneprotokollia käyttäen. CASS-asemien käyttämiä tietoliikenneyhteyksiä ovat esimerkiksi RS232, RS422 sekä MIL-1553. Nykyisin testattavista laitteista löytyy myös uudempia kommunikointiväyliä kuten Ethernet ja USB (engl. Universal Serial Bus), joille CASS-asemassa ei ole suoraa tukea. Lisääntynyt kommunikaatioväylien määrä lisää huomattavasti tarvetta käsitellä ja analysoida testattavalta laitteelta saapuvaa dataa.

CASS-aseman testauslaitteistoa ohjataan ATLAS-ohjelmointikielellä (engl. Abbreviated Test Language for All Systems) ja asemien käyttöjärjestelminä toimivat Windows sekä OpenVMS. ATLAS on kehitetty CASS-aseman sisäisten laitteiden hallintaan ja suoritusalustasta riippumattomaksi ohjelmointikieleksi. ATLAS soveltuu CASS-aseman sisäisten laitteiden ohjaamiseen hyvin, mutta testattavalta laitteelta saatujen tulosten käsittely on ATLASta käyttäen todella hankalaa. Lisääntyneen datan käsittelytarpeen johdosta ATLAS-ohjelmointikielen puutteet aiheuttavat suuren työkuorman ja hidastavat testausohjelmistojen tuottamista.

ATLAS-kieltä voidaan laajentaa kahdella eri menetelmällä. Ensimmäinen näistä menetelmistä on FEP (engl. Functional Extension Program). FEP on toisella ohjelmointikielellä toteutettu ohjelma, jota voidaan kutsua ATLASilla toteutetusta ohjelmasta. FEP voidaan toteuttaa käyttämällä esimerkiksi Fortran- tai C-ohjelmointikieliä, jolloin näiden kielten ominaisuudet saadaan käyttöön myös ATLAS-kielellä toteutetuissa ohjelmissa. Toinen mahdollinen ATLAS-kielen laajennusmenetelmä on skriptien käyttäminen. Skripteillä toteutettua toiminnallisuutta voidaan hyödyntää sellaisissa tapauksissa, joissa vaadittu toiminta on riittävän yksinkertaista.

FEPeja käytetään Insta ILS:n CASS-asemilla, mutta toistaiseksi niitä ei ole itse tuotettu. Tämän tutkielman tarkoituksena on selvittää, miten FEPeja saadaan tuotettua käytettävissä oleviin käyttöjärjestelmäympäristöihin sekä testilaitteistoihin. Tämän lisäksi tavoitteena on valita laitteiden testaamisessa tarvittava esimerkki-FEP ja toteutetaan se eri käyttöjärjestelmäympäristöissä.

Luvussa 2 esitellään käytettävä laitteisto ja laitteiden käyttöjärjestelmäympäristöt sekä käytettävien ohjelmointikielten ominaisuuksia ja rajoitteita. Luvuissa 3 ja 4 käsitellään FEPin toteuttamista erityyppisille CASS-asemille. Luvussa 5 käsitellään FEPien avulla saatuja hyötyjä eri laitteisto- ja käyttöjärjestelmäympäristöissä. Luvussa 6 käsitellään saatuja tuloksia ja asetetaan lähtökohdat FEPien jatkokehitykselle.

2. LAITTEISTO- JA JÄRJESTELMÄYMPÄRISTÖT

Tämä luku käsittelee käytössä olevia testauslaitteistoja, CASS-asemia sekä ATLAS-ohjelmointikielen ominaisuuksia. Testauslaitteistosta ja sillä testattavista laitteista tarkastellaan niitä ominaisuuksia, jotka aiheuttavat tarpeen ATLAS-ohjelmointikielen laajentamiselle. Tässä luvussa tarkastellaan myös sitä, miksi testauksessa käytettävä ATLAS-ohjelmointikieli luo tarpeen ulkoisille, toisella ohjelmointikielellä toteutetuille ohjelmille.

2.1 CASS-asemat

CASS-asemia käytetään erilaisten avioniikkalaitteiden ja -järjestelmien testaamiseen sekä mahdollisten vikojen paikantamiseen testattavissa laitteissa. Kuvassa 1 on esitettynä RTCASS-testausasema (engl. Reconfigurable Transportable Consolidated Automated Support System), joka on toinen tässä tutkielmassa käytettävistä CASS-asemista.

Kuva 1 RTCASS-asema

2.1.1 Käytettävä laitteisto

CASS-asemia on tällä hetkellä olemassa kuutta eri mallia. Eri CASS-asemien malleja on esiteltynä taulukossa 1. Insta ILS:ssä käytössä olevien CASS-asemien mallit ovat RTCASS sekä MFCASS (engl. Main Frame Consolidated Automated Support System).

Taulukko 1 CASS-asemien tyypit [1][2]

Malli	Selite, engl.
CNICASS	Communications, Navigation, and Identification CASS
EOCASS	Electro-Optical CASS
HPCASS	High Power CASS
HYBCASS	Hybrid CASS
MFCASS	Main Frame CASS
RFCASS	Radio Frequency CASS
RTCASS	Reconfigurable Transportable CASS

CASS-asemia käytetään monien eri lentolaitteiden järjestelmien laitteiden testaamiseen. Joitakin CASS-asemalla testattavien laitteiden alustoja on esiteltynä taulukossa 2. CASS-asemalla testattavista laitteista ovat esimerkiksi muistimoduulit, näytöt, radiot, tallentimet sekä tietokoneet. Testattavan laitteen ominaisuuksien ja testausvaatimusten perusteella voidaan valita sopiva CASS-asema testaamista varten.

Taulukko 2 Testattavien laitteiden alustoja [1][3]

Lentolaite
F/A-18 Hornet
MV-22 Osprey
AV-8B Harrier
EA-6B Provler
C-130 Hercules
AH-1Z Viper
UH-1Y Venom

Tässä työssä käytettävät CASS-asemat eroavat toisistaan esimerkiksi käyttöjärjestelmien sekä muutamien aseman sisäisten mittalaitteiden osalta. Taulukossa 3 on esitetty muutamia CASS-asemista löytyviä laitteita, joita käyttämällä testattavalle laitteelle voidaan syöttää erilaisia signaaleita, jännitteitä ja virtoja, sekä mitata testattavan laitteelta saatavia vasteita.

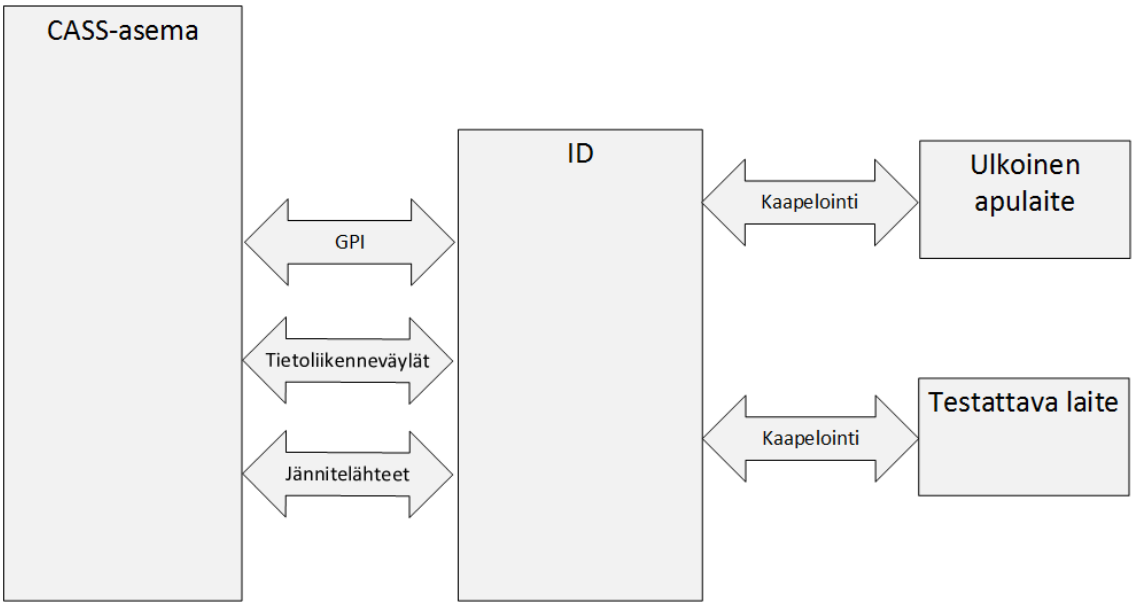
Käytettävissä olevalla testauslaitteistolla pyritään simuloimaan testattavan laitteen toimintaympäristöä niin normaali kuin vikatilanteissa ja tarkastelemaan vasteiden avulla testattavan laitteen toimintaa. Erilaisia tilanteita simuloimalla voidaan varmistaa testattavan laitteen oikea toiminta. Tämän lisäksi testaamalla pyritään paikantamaan vikaantunut osa, oli kyseessä sitten piirikortti tai komponentti. CASS-asemien sisäisten laitteiden toimintaa ohjataan ATLAS-ohjelmointikielen avulla.

Taulukko 3 CASS-aseman laitteistoja [1]

Mittalaite	Ominaisuudet
65 V tasajännitelähde	Tuottaa tasajännitettä testattavalle laitteelle
120 V tasajännitelähde	Tuottaa tasajännitettä testattavalle laitteelle
450 V tasajännitelähde	Tuottaa tasajännitettä testattavalle laitteelle
135 V vaihtovirtalähde	Tuottaa vaihtovirtaa testattavalle laitteelle
Digitaalinen yleismittari	Mittaa jännitettä, virtaa ja resistanssia.
Taajuusmittari	Mittaa testattavalta laitteelta signaalin taajuutta sekä aikajaksoja
Pussigeneraattori	Tuottaa testattavalle laitteelle haluttua signaalia.

Testattavan laitteen ja CASS-aseman välisenä rajapintana toimii erillinen kytkentälaitteisto, ID (engl. Interface Device). Jokaisella testattavalla laitteella on uniikki ID jonka avulla testattavan laitteen ja CASS-aseman väliset liitännät yhdistetään. Tällä menetelmällä CASS-aseman ja testattavan laitteen liitännät on saatu toisistaan riippumattomiksi ja mahdollistetaan uusien testattavien laitteiden testauskyvyn luominen asemalle. ID kiinnitetään CASS-asemasta löytyvään GPI:hin (engl. General Purpose Interface), jonka välityksellä testattavassa laitteessa saadaan käyttöön suurin osa CASS-asemasta löytyvistä laitteistoista. Loput laitteistot kuten CASS-aseman tietoliikenneyhteydet kytketään ID:hen erillisiä kaapeleita käyttäen. Testattava laitteisto sekä mahdollisesti muut testauksessa käytettävät apulaitteet saadaan kytkettyä ID:hen kaapeleiden avulla. Kuvassa 2 on esitettynä esimerkki CASS-aseman kytkentätavasta.

Mikäli testattavassa laitteessa on käytössä sellaisia tietoliikenneyhteyksiä tai muita liitäntöjä, joita testaukseen käytettävässä CASS-asemassa ei ole saatavilla, voidaan ID varustaa erillisellä laitteistolla, josta nämä yhteydet ja tarvittavat liitännät löytyvät. Tällainen laitteisto voi olla esimerkiksi teollisuus-PC (engl. Personal Computer), josta tarvittavat kommunikaatioväylät sekä liitännät löytyvät. ID:n sisään lisättyä teollisuus-PC:tä kutsutaan tässä työssä IDPC:ksi.



Kuva 2 Liitynnät

Taulukossa 4 on esitetty CASS-asemista valmiiksi löytyviä tietoliikenneväyliä, joiden kautta testattavat laitteet voivat kommunikoida CASS-aseman kanssa.

Taulukko 4 CASS-asemien tietoliikenneyhteyksiä [1]

Tietoliikenneyhteys
ARINC-429
IEEE-488
IEEE-802.3
MIL-STD-1553
RS-232
RS-422

Nykyisin testattavissa laitteissa käytetään usein kommunikaatioväylinä esimerkiksi Gbit/s Ethernetiä (IEEE 802.3ab), SMBus:ia (engl. System Management Bus), USB:tä ja valokuitua, joille ei CASS-asemista valmiiksi löydy tukea. Nämä kommunikaatioväylät saadaan laitteita testattaessa käyttöön IDPC:n välityksellä. CASS-asemasta puuttuvat tietoliikenneväylät ja niiden välittämä data saadaan käyttöön muuntamalla se IDPC:llä CASS-asemasta löytyvän tietoliikenneväylän muotoon. Tällainen muunnos voi olla esimerkiksi USB-väylältä saatavan datan muuntaminen sarjaliikenneväylälle sopivaksi.

2.1.2 Testattavan laitteiston asettamat vaatimukset

Uusia tietoliikenneväyliä ja liitäntöjä tarvitaan simuloimaan testattavan laitteen todellista käyttöympäristöä ja kaikkia siinä esiintyviä tilanteita. Uusien tietoliikenneväylien lisääminen järjestelmiin ja testattaviin laitteisiin lisää myös käsiteltävän datan määrää

huomattavasti. IDPC:n lisääminen testausjärjestelmään ei sinällään muuta testattavan laitteiston asettamien vaatimusten tilannetta parempaan suuntaan käsiteltävän datan määrän suhteen, sillä IDPC:tä tulisi käyttää ainoastaan datan muuntamiseen muodosta toiseen. Kaikki datan tulkinta sekä päätökset käytettävän datan perusteella tulee suorittaa CASS-asemassa ja näin IDPC:n tulisi toimia tässä järjestelmässä ainoastaan tiedon välittäjänä. Kaikkien päätösten suorittaminen CASS-asemassa on yksi testausohjelmistojen, TPS:ien (engl. Test Program Set), suunnitteluvaatimuksista, joskin siitä voidaan poiketa tapauskohtaisesti [1]. TPS on kokonaisuus joka sisältää testausohjelman lisäksi myös kaiken muun tietyn laitteen testaamisessa tarvittavan laitteiston sekä dokumentaation. TPS:n sisältöön kuuluvat esimerkiksi ATLASilla toteutettu testiohjelma, tarvittavat FEPit, ID sekä kaapelointi [5]. Suorittamalla kaikki testauksen aikaiset päätökset ja analysointi ainoastaan CASS-asemaa käyttäen, voidaan varmistua siitä, että päätökset suoritetaan aina kalibroidulla ja standardien mukaisella laitteistolla.

Vaikka testattavista laitteista löytyy edelleen valmiuksia myös vanhemmille viestintäväylille, niin uudet väylätekniikat kuten USB ja Ethernet toimivat nykyisin testattavissa laitteissa pääasiallisina viestintäväylinä. Kommunikaatioväylien sekä ominaisuuksien huomattava lisääntyminen testattavissa laitteissa lisää merkittävästi tarvetta käsitellä ja analysoida laitteiden lähettämää dataa. Aikaisemmin testattavista laitteista on mitattu sähköisiä suureita ja analysoitu yksinkertaisten sarjaliikenneviestin sisältöä, joihin CASS-aseman sekä ATLASin ominaisuudet ovat riittäneet. Nykyisin testattavat laitteet välittävät huomattavasti suuremman määrän dataa kuin vanhemmat laitteet. Laitteiden välittämää dataa joudutaan analysoimaan ja käsittelemään oikean toiminnan varmistamiseksi testattavassa laitteessa. Tällaisen suuren tietomäärän käsittely ja analysointi CASS-asemalla eivät ole helposti toteutettavissa, johtuen ATLAS-ohjelmontikielen rajoitteista.

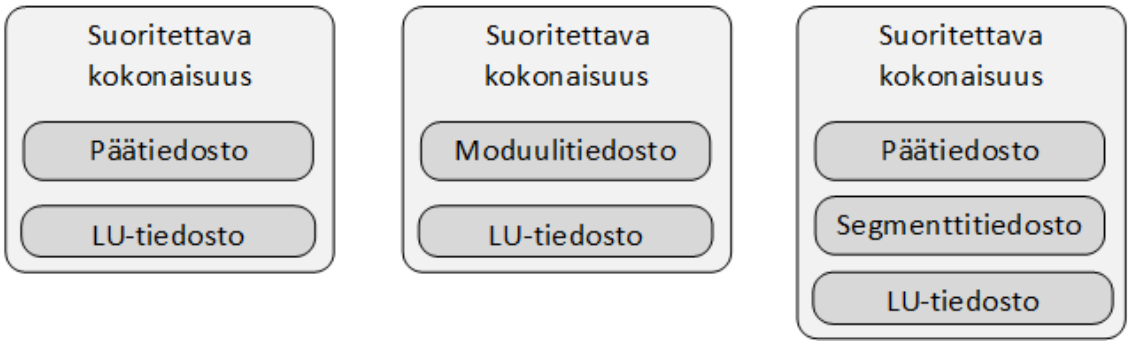
2.2 ATLAS

CASS ATLAS on ohjelmointikieli, jolla CASS-asemalla suoritettavat testiohjelmat on pääosin kirjoitettu. ATLAS on IEEE:n standardin 716-1995 määrittelemä korkean tason ohjelmointikieli, jonka pääasiallinen käyttö on testauksessa. Standardi IEEE 716-1995 on uusin ja samalla kolmas julkaistu versio ATLASista. Standardin määrittelemä kieli on suunniteltu riippumattomaksi testiä suorittavasta järjestelmästä ja tämä mahdollistaa sen implementoinnin erilaisiin automaattisiin testausjärjestelmiin [7]. ATLAS-kielen ominaisuudet eivät sellaisenaan riitä CASS-asemalla suoritettavan testauksen vaatimuksiin, joten CASS-asemalla suoritettavaa testausta varten luotiin oma kieli, CASS-ATLAS. CASS ATLASin pohjana on standardin IEEE 716-1985 määritelmä, jota on laajennettu standardien IEEE 416-1984 ja IEEE 716-1989 osilla. CASS-aseman valmistaja on myös valmistanut ja lisänny omia laajennusosia CASS ATLASiin [4]. CASS ATLASia ei tule sekoittaa joissakin standardeissa esiintyvään merkintään C/ATLAS (engl. Common/Abbreviated Test Language for All Systems).

IEEE:n määritelmä ATLASin toimimisesta korkealla tasolla tarkoittaa ainoastaan kykyä toimia useissa eri testausjärjestelmissä ja olla käytettävästä testauslaitteistosta riippumaton. Verrattaessa tämän tutkielman kirjoitushetkellä yleisesti käytössä oleviin ohjelmointikieliin, kuten C++ tai Java, ATLASia voidaan pitää matalan tasoisena ja laitteistonläheisenä ohjelmointikielenä. ATLASin avulla voidaan helposti ohjata käytettävän testauslaitteiston toimintaa yksittäisen laitteen ja muodostettavan signaalin tasolla. Pääpiirteissään CASS-aseman laitteita ohjataan ATLASilla siten, että ensin muodostetaan tarvittavat signaalipolut kytkemällä aseman sisäiset releet tarvittaviin asentoihin. Signaalipolkujen muodostamisen jälkeen annetaan käytettävälle laitteelle halutut parametrit joko mittauksen suorittamiseen tai signaalin muodostamiseen.

2.2.1 ATLAS ohjelman rakenne

Tässä työssä käytetään tästä eteenpäin CASS ATLASta lyhyempää nimitystä ATLAS. ATLASilla toteutettu testiohjelma voidaan jakaa useisiin eri tiedostoihin, tai hyvin yksinkertaisessa tapauksessa toteuttaa jopa vain yhdellä tiedostolla. Testiohjelma kuuluu aina vähintään niin sanottu päätiedosto, sekä haluttaessa erillisiä moduuli- tai segmenttitiedostoja. Moduulitiedosto on itsenäinen kokonaisuus, joka sisältää itsessään kaiken tarvittavan jotta se voidaan suorittaa. Segmenttitiedosto on osa kokonaisuutta ja siinä voidaan kutsua suoritettavaksi osia testiohjelman päätiedostosta. Segmenttitiedostot muodostavat yhdessä päätiedoston kanssa kokonaisuuden ja suoritettavan testiohjelman. Näiden tiedostojen lisäksi on olemassa erillinen LU-tiedosto (engl. Look Up), joka sisältää tiedot testiohjelmassa käytettävistä relekytkennöistä ja CASS-aseman sisäisistä mittalaitteista sekä muusta laitteistosta. Kuvassa 3 on esitettynä eri tavat muodostaa suoritettava kokonaisuus käytettävistä tiedostoista. Testiohjelma voidaan toteuttaa myös ilman LU-tiedostoa, mutta tämä on todella harvinaista. LU-tiedostossa määritellään esimerkiksi tulosteiden antamiseen käytettävä laitteisto, näyttö, tulostin tai tiedosto, joita ilman ohjelman suorittaminen on hankalaa.



Kuva 3 Suoritettavia kokonaisuuksia

ATLASilla toteutettuja testiohjelmia suoritetaan molemmissa CASS-asemissa graafisen käyttöliittymän välityksellä. Sovellus jolla testiohjelmia ohjataan, vaatii testiohjelman suorittamiseksi käännetyt testiohjelman koodit sekä MTPSI-tiedoston (engl. Master Test

Program Set Index). MTPSI-tiedostossa määritellään kaikki CASS-aseman laitteet joita testin aikana käytetään, käännetyn testiohjelman sijainti järjestelmälevyllä, sekä näiden lisäksi kaikki tarvittavat ulkoiset laitteet, kuten esimerkiksi ID:n. MTPSI-tiedosto tehdään omalla apuohjelmallaan MFCASS-asemalla. Sama testiohjelma voidaan myös käynnistää eri laitteistokonfiguraatioilla, luomalla useita eri MTPSI-tiedostoja [8]. Useille laitteistokonfiguraatioille tai MTPSI-tiedostoille ei kuitenkaan ole usein tarvetta.

ATLAS tukee aliohjelmien käyttöä testiohjelmien muodostamisessa. Aliohjelmia voidaan pitää esimerkiksi C++-kielen funktioita vastaavina ohjelmakoodin kokonaisuuksina. Aliohjelmille voidaan välittää parametreja ja ne voivat palauttaa paluuarvoja. Aliohjelmat tulee sijoittaa segmenttejä käytettäessä testiohjelman päätiedostoon tai moduulien tapauksessa siihen moduuliin, jossa niitä käytetään.

ATLASilla toteutetussa ohjelmassa jokainen suoritettava lause aloitetaan numeroimalla kyseinen rivi nousevassa järjestyksessä. Samaa menetelmää on käytetty myös esimerkiksi BASIC- sekä FORTRAN-ohjelmointikielten versioissa. Tätä menetelmää käyttämällä ohjelmassa voidaan hyödyntää GOTO-käskyjä, joita myös suositellaan käytettäväksi ATLASilla ohjelmoitaessa. Rivinumerointi ATLASilla toteutetuissa testiohjelmissa on rajoitettu välille 0-999999. Käytettävä rivinumeroiden alue ATLASilla toteutetussa ohjelmassa on jaettu taulukossa 5 esitettyjen vaatimusten mukaisesti. Rivinumeroinnille asetetut vaatimukset perustuvat TPS:n valmistamiselle asetettuihin yleisiin vaatimuksiin [5]. Fortranista tutun rivinumeroinnin lisäksi ATLASilla toteutetussa ohjelmassa käytetään Fortran 77-ohjelmointikielen tulostusformaattia kaikkien testiohjelman tulosteiden muotoiluihin [9].

Taulukko 5 Rivinumerointi

Rivinumero	Käyttötarkoitus
000100-099999	Esittelytoimenpiteet muuttujille, aliohjelmille ja muille testiohjelmassa käytetyille resursseille
100000-199999	Ensimmäinen käynnistyspiste
200000-299999	Toinen käynnistyspiste
x00000-x99999	x:s käynnistyspiste
900000-999999	Testiohjelman lopetustoimenpiteet

ATLASilla toteutetussa testiohjelmassa voi olla useita eri suorituskokonaisuuksia. Kun testattavasta laitteesta yritetään paikallistaa tiettyä vikaa, ei ole järkevää suorittaa sellaisia testejä, jotka eivät liity kyseiseen vikatilanteeseen millään tavalla. Tällöin ohjelma voidaan käynnistää sellaisesta käynnistyspisteestä, josta päästään suoraan etsittyä vikaa testaaviin kohtiin testiohjelmassa. Yleensä nämä suorituskokonaisuudet on jaettu omiksi segmentti- tai moduulitiedostoiksi.

ATLASissa käytettävät tietotyypit ovat esiteltynä taulukossa 6. Taulukossa esitettyjen tietotyyppien lisäksi ATLASissa voidaan käyttää näistä tietotyypeistä muodostettavia taulukoita.

Taulukko 6 ATLAS tietotyypit

Tietotyyppi	Ominaisuudet
integer	kokonaisluku ±32767
long integer	kokonaisluku ±2147483648
boolean	tosi / epätosi
decimal	reaaliluku ±0,1e+40
msgchar	char merkkijono
digital	binääri-, oktaali-, heksadesimaaliluku

2.2.2 ATLAS-kielen Rajoitteet

Yksi suurimmista ATLAS-kielen haasteista on dokumentaation vähäisyys. Vaikka ATLAS perustuu IEEE:n standardisoimaan kieleen, niin siihen mukaan otetut aikaisempien ATLASin versioiden lisäosat, sekä CASS-asemien valmistajan lisäämät ominaisuudet on dokumentoitu heikosti. Mikään saatavilla oleva dokumentaatio ei yksinään tarkasti kuvaa ATLASin ominaisuuksia ja dokumentaatiossa on myös huomattavia määriä virheitä. Näin ollen suurin osa ATLASin ominaisuuksista on selvitetty kokeilemalla ja esimerkkikoodeja tutkimalla.

Sähköisten suureiden tuottaminen ja mittaaminen CASS-asemalla on testausohjelmistojen tärkein tehtävä ja ominaisuus. Yksinkertaisten sähköisten suureiden tuottaminen, mittaaminen ja tarkastelu CASS-asemalla ovat helposti toteutettavissa ATLAS-kieltä käyttäen. Myös yksinkertaisten tietoliikenneviestien lähettäminen ja vastaanottaminen onnistuu ATLASta käyttäen helposti. Näille aliohjelmille on olemassa useita valmiita syntaksitaulukoita, joiden avulla CASS-aseman laitteiden ohjaaminen on toteutettavissa monille eri signaalityypeille. Kaikki näihin signaaleihin ja niiden analysointiin kohdistuvat toimenpiteet sen sijaan ovat hankalasti toteutettavissa ATLASta käyttäen.

CASS-asemalla signaalien mittaaminen sekä saatujen tulosten talletus voidaan suorittaa eri kantalukuformaateissa. ATLAS tukee binääri-, oktaali- kymmen- ja heksalukujärjestelmiä. Nämä kantalukuformaatit ovat kuitenkin täysin erillisiä lukuformaatteja, eikä niiden muunto toiseen kantalukumuotoon ole ATLASissa suoraan tuettu. Myöskään esimerkiksi C-kielestä tuttu char-tietotyypin tyyppimuunnos toiseksi tietotyypiksi ei ole mahdollinen ATLAS-kielessä. Käsiteltävän tietomäärän lisääntyessä eri kantalukujärjestelmissä olevien tulosten tarkastelun sekä vertailun tarve lisääntyvät. Myöskään esimerkiksi binääri- tai heksadesimaalilukujen aritmeettiset operaatiot eivät ole mahdollisia ATLAS-kieltä käytettäessä. ATLASista puuttuvia ominaisuuksia on listattuna taulukossa 7.

Taulukko 7 ATLASista puuttuvia ominaisuuksia

Ominaisuus
binäärilukujen aritmeettiset operaatiot
binäärilukujen muunto toiseen kantalukuformaattiin
char-merkkijonojen indeksointi
char-merkkijonojen konkatenointi
else if -rakenteen puuttuminen
rekursio
syötteen tyypin tunnistaminen
tietotyypin tyyppimuunnos

Jotkin halutuista, mutta puuttuvista, ATLAS-kielen ominaisuuksista olisi toteutettavissa todella pitkillä ja suurta käsityötä vaativilla ohjelmarakenteilla. Esimerkiksi binäärilukujen muuttaminen toiseen muotoon voitaisiin toteuttaa pitkillä *if else* -rakenteilla, mutta tämä ei ole järkevää. Käsin tällaisen ohjelmakoodin kirjoittaminen olisi liian työlästä eikä näin ollen ole järkevä vaihtoehto. ATLAS ei myöskään tue rekursiota, jolla voitaisiin helposti käsitellä tällaisia samaa kaavaa toistavia ohjelmarakenteita [10].

Muista ohjelmointikielistä tutun *else if* -rakenteen muodostaminen on ATLAS-kielessä hankalaa, sillä ATLAS ei tue kyseistä syntaksia. ATLAS-kielessä jokaista *if*-komentoa kohden voidaan käyttää vain yksi *else*-lause. Tällöin tehokkuutta lisäävä oikosulkuevaluoinnin tyyppinen *else if* -lauseen käyttö ei ole mahdollista, vaan jokainen *if*-haara käytäisiin ohjelmassa läpi, vaikka haluttu tulos olisi jo saavutettu. Useita vertailuja voidaan suorittaa sisäkkäisillä *if*-haaroilla. Sisäkkäisten *if*-haarojen käyttöä kuitenkin hankaloittaa se, että ATLAS-kääntäjä ei hyväksy yli 80-merkkiä pitkiä rivejä ja näin ollen lähdekoodin selkeydestä ja oikein tehdyistä sisennyksistä joudutaan hyvinkin helposti tinkimään, mikä tekee testiohjelmien rakenteesta todella epäselvän.

Kun ATLASilla toteutetulle ohjelmalle annetaan jokin syöte, on tämän syötteen oltava juuri oletettua tyyppiä tai aiheutuu poikkeus. ATLASissa ei ole mahdollista toteuttaa esimerkiksi C++-ohjelmointikielestä löytyvän poikkeuksien käsittelyn tapaista toimintaa. C++:lla toteutetussa ohjelmassa on mahdollista varautua tällaisiin virhetilanteisiin ja lopettaa ohjelman suorittaminen hallitusti tai jatkaa ohjelman suorittamista siten, että virhetilanne on otettu huomioon poikkeuksen jälkeen suoritettavissa toimenpiteissä. ATLASilla toteutetussa ohjelmassa, mikäli annetun syötteen tyyppi poikkeaa oletetusta, ei ohjelmaa voida jatkaa tai toimintaa lopettaa hallitusti. ATLASilla toteutettu ohjelma kyllä havaitsee väärän tyyppisen syötteen ja antaa tästä ilmoituksen käyttäjälle, mutta ohjelman toimintaan ei voida luoda käsittelyä, joka osaisi käsitellä tätä tilannetta normaalista suorituksesta poikkeavalla tavalla. Virhetilanteissa ATLASilla toteutettu ohjelma joko lopetetaan kokonaan tai suoritusta jatketaan kuin virhettä ei olisi tapahtunut. Mikäli ohjelman suoritusta jatketaan virhetilanteesta huolimatta, ei käytettävän syötteen tulkinnasta voida tietää mitään. Testiohjelman suorituksen jatkaminen virheellisen tiedon varassa on todella suuri riski käytettävälle ja testattavalle laitteistolle. Koska testiohjelmistot ovat perusrakenteeltaan sellaisia, että niillä syötetään testattaville laitteille

useita eri jännitteitä ja signaaleita, saattaa väärän tiedon käyttäminen aiheuttaa laitteistolle hyvin helposti pysyvää vauriota.

2.3 Functional Extension Program

ATLAS mahdollistaa ulkoisten ohjelmien kutsumisen sekä suorittamisen testiohjelman ajon aikana ja tällä tavalla uusien ominaisuuksien lisäämisen ohjelmiin, jotka on toteutettu ATLASilla. Näitä ulkoisia, toisella ohjelmointikielellä toteutettuja ohjelmia kutsutaan FEPeiksi. FEP on ulkoinen ohjelma, joka on kirjoitettu ohjelmointikielellä, jonka kääntämiselle ja suorittamiselle CASS-asemasta löytyy tuki. FEP on alun perin määritelty termillä Fortran External Program, mutta FEPin toteutus on mahdollista Fortranin lisäksi myös muilla ohjelmointikielillä. Taulukossa 8 on lueteltuna eri CASS-asemien tukemat ohjelmointikielet FEPin toteutukselle.

Taulukko 8 CASS-asemien tukemat ohjelmointikielet [6][11]

MFCASS	RTCASS
• FORTRAN • C • Pascal • Visual Basic	• C • C# • C++

CASS-aseman tukemalla ohjelmointikielellä tarkoitetaan sellaista ohjelmointikieltä, jolla kirjoitetun ohjelman suorittaminen CASS-asemassa on mahdollista. FEPin suorittamisen lisäksi CASS-asemalta vaaditaan kääntäjä, jolla FEPin toteuttaminen on mahdollista käytettävässä ympäristössä.

Tällä hetkellä käytössä oleva MFCASS-aseman käyttöjärjestelmä on Windows ja RTCASS-aseman käyttöjärjestelmä OpenVMS. Käytettävä Windowsin versio on kirjoittamishetkellä yleinen ja hyvin tuettu käyttöjärjestelmä. OpenVMS on huomattavasti Windowsia vanhempi ja kirjoitushetkellä tuntemattomampi käyttöjärjestelmä.

Sen lisäksi, että ATLASilla toteutettuja ohjelmia voidaan laajentaa käyttämällä FEPeja, voidaan FEPeja käyttää myös CASS-aseman omien toimintojen tarkkailuun ja laitteiston tarkastamiseen. FEP voi olla myös joko yksittäiselle TPS:lle toteutettu sovellus, tai kaikille CASS-asemassa suoritettaville TPS:lle yhteinen. TPS:lle toteutettua FEPiä kutsutaan TPS FEPiksi ja CASS-aseman omien järjestelmien käyttöön tarkoitettua FEPiä System FEPiksi. TPS FEP voidaan jakaa vielä kahteen eri ryhmään, common deployed TPS FEP sekä TPS deployed FEP. Common deployed TPS FEP on toteutettu kaikkien testiohjelmien yhteiseen käyttöön. TPS deployed FEP on ainoastaan jonkin tietyn TPS:n käyttöön toteutettu FEP. Eri tarkoituksiin toteutettujen FEPien toteutuksessa ei ole eroja, mutta niiden sijainti järjestelmässä riippuu FEPin käyttötarkoituksesta. Eri FEP-tyyppien sijainnit on tarkemmin määriteltynä luvuissa 3.4 ja 4.2. [4]

Taulukossa 9 on esitelty esimerkkejä jo valmiiksi käytössä olevista FEPeistä sekä niiden toiminnallisuudesta. Kaikki taulukossa olevat FEPit ovat Common deployed TPS FEPejä ja näin ollen niitä voidaan käyttää kaikissa eri ATLASilla toteutetuissa testiohjelmissa. Taulukossa esiteltävät FEPit on toteutettuna molemmille käytettävissä olevista CASS-asematyypeistä. Näistä valmiista FEPeistä ei ole tiedossa muuta kuin niiden rajapinta. Valmiiden FEPien lähdekoodi ei ollut työn aikana saatavilla kummallekaan CASS-asemalle.

Taulukko 9 Valmiit FEPit

FEP	Kuvaus
CVT_STR_INT	Muuntaa Char-jonon Int tyyppiseksi muuttujaksi. (+32767 – -32767)
CVT_STR_LG_INT	Muuntaa Char-jonon Long Int tyyppiseksi muuttujaksi.
GET_FILE_LENGTH	Palauttaa tiedoston merkkien määrän.
COMPARE_STRINGS	Vertaa kahden Char-jonon merkkejä määritellyllä merkkiväliltä.

2.3.1 OpenVMS

OpenVMS on yksi aikaisemmin laajassa käytössä olleista käyttöjärjestelmistä. OpenVMS perustuu aikaisempaan VAX/VMS käyttöjärjestelmään. Ensimmäinen VAX/VMS-käyttöjärjestelmä on julkaistu jo vuonna 1977 ja tämä käyttöjärjestelmä oli tarkoitettu laitteille, jotka on toteutettu VAX-suoritinarkkitehtuurilla. Vuonna 1992 VAX/VMS-järjestelmiä arveltiin olevan käytössä vielä noin puoli miljoonaa ja käyttäjiä noin 10 miljoonaa [13]. VAX arkkitehtuuria on käytetty myös aikaisemmassa Block I mallin MFCASS-asemassa. Siirryttäessä VAX-suoritinarkkitehtuurista ALPHA-suoritinarkkitehtuuriin tehtiin myös tietoteknisesti merkittävä muutos siirtymällä käyttämään RISC-käskykantaa (engl. Reduced Instruction Set Computer) CISC-käskykannan (engl. Complex Instruction Set Computer) sijaan [12]. Nykyisten MFCASS-asemien arkkitehtuurit ovat malliltaan Block II sekä Block III ja niissä käytetään ALPHA arkkitehtuuria [4]. Ensimmäinen OpenVMS:n versio on julkaistu vuonna 1992 ja järjestelmää päivitetään edelleen. Kirjoitushetkellä viimeisin julkaistu versio OpenVMS-käyttöjärjestelmästä on versio V8.4-1H1 ja se on julkaistu 9/2015 [14].

CASS-asemien käyttöjärjestelmät asettavat vaatimuksia ja rajoitteita tässä työssä toteutettaville FEPeille. Sen lisäksi, että toteutettavalle FEPille pitää löytyä käyttöjärjestelmässä toimiva kääntäjä, niin myös toteutuskielen tulisi olla sama. Tällöin FEPin lähdekoodin uudelleenkirjoituksen määrä saataisiin minimoitua, kun se toteutetaan molemmille ympäristöille samalla ohjelmointikielellä.

2.3.2 FEPin toteutustavat

RTCASS-asemalle toteutettavan FEPin pohjana voidaan käyttää aseman apuohjelmistoihin kuuluvaa FEP Wizard -ohjelmistoa. Tämä ohjelmisto on Windows-käyttöjärjestelmäympäristössä suoritettava .NET-sovelluskehystä hyödyntävä ohjelma. FEP Wizard luo C#-projektin käyttäjän syöttämän rajapinnan perusteella. FEPin toiminnallisuus luodaan tämän jälkeen Visual Studio -kehitysympäristöä käyttämällä. Tämä menetelmä ei kuitenkaan tue sitä ajatusta, että samalla ohjelmakoodilla saataisiin toteutettua FEP edes osittain molemmille työssä käytetyille CASS-asemille. RTCASS-asemalle FEP voidaan toteuttaa myös hyödyntämällä FEP Wizardin käyttämää DirectTest-kirjastoa. DirectTest-kirjasto tarjoaa rajapinnan, jonka avulla voidaan välittää parametreja ATLASilla toteutetun testiohjelman sekä FEPin välillä. Tämä kirjasto voitaisiin implementoida käyttöön myös C-kielellä toteutetussa FEPissä ja näin ollen hyödyntää ennalta luotua ATLAS-kielen rajapintaa.

MFCASS-asemalla FEPin toteuttaminen suoritetaan jollakin aseman tukemista ohjelmointikielistä. Mitään valmista apuohjelmistoa FEPin luomiseen MFCASS-asemalla ei ole olemassa. Tässä työssä käytettävissä olevissa MFCASS-ympäristöissä ei ole tällä hetkellä asennettuna mitään vaadituista käännöstyökaluista FEPin luomiseksi. Käännöstyökalujen kokeellinen asentaminen testimielessä MFCASS-asemaan ei riskien takia tule kysymykseen. Tästä syystä työ tulee keskittymään myös virtuaalisen käännösympäristön pystyttämiseen MFCASS-aseman osalta. Virtuaalisen kääntöympäristön pystyttäminen tulee olemaan yksi työn haasteellisimmista toimenpiteistä johtuen järjestelmän harvinaisuudesta ja tuntemattomuudesta kirjoittajalle.

FEP voidaan toteuttaa MFCASS-aseman käyttöjärjestelmässä, OpenVMS:ssä, myös DCL:a (engl. Digital Command Language) käyttämällä. DCL on OpenVMS:n komentorivillä käytettävä komentokieli, mutta samalla se on myös skriptikielen tavoin toimiva komentokieli. Samoja komentoja voidaan siis suorittaa suoraan komentoriviltä sekä kootusti komentoja sisältäviä tiedostoja suorittamalla [15]. Tällä menetelmällä tehdyt FEPit pystyvät siis niihin toimenpiteisiin, joita käyttöjärjestelmän komentorivillä pystytään suorittamaan. Skriptejä käytettäessä FEPin toiminta muodostuu siten, että ATLAS-kielisestä testiohjelmasta välitetään tietoa johonkin järjestelmässä olevaan tiedostoon. Tiedoston käsittelyn jälkeen testiohjelmassa käsketään skriptiä suorittamaan tiedostolle halutut toimenpiteet. Viimeisenä vaiheena testiohjelma lukee tiedostoon tehdyt muutokset ja käyttää niitä. Parametrien suora välitys testiohjelman ja skriptin välillä ei ole mahdollista.

RTCASS-aseman käyttämässä Windows-käyttöjärjestelmässä skriptien hyödyntäminen onnistuu Batch-skriptikieltä käyttämällä. Batch-kielen tapauksessa rajoitteet ja ominaisuudet FEPin tuottamisessa ovat suunnilleen samat kuin DCL:ää käytettäessä. Ominaisuuksiltaan DCL:n käskykanta on laajempi ja tästä syystä monipuolisempi kuin uudemman Batch-kielen käskykanta.

2.3.3 FEPille asetettavat vaatimukset

Mikäli molemmilla työssä käytetyillä asematyypeillä, RTCASS ja MFCASS, halutaan suorittaa sellainen TPS, joka käyttää suorituksen aikana FEPiä, niin käytettävän FEPin tulee olla saatavilla molemmille asemille. Näin ollen FEPin suunnittelussa pitää ottaa huomioon toteutusmahdollisuudet asematyypeillä. Suunnittelemalla FEP siten, että sen toteuttaminen, kääntäminen sekä suorittaminen onnistuvat molemmissa asematyypeissä, voidaan lisätä testiohjelmien riippumattomuutta laitteistosta, tietystä CASS-asemasta, ja parantaa näin ollen järjestelmän saatavuutta sekä testausvalmiutta.

FEPin toteutuksessa tulee ottaa huomioon, että FEP ei saa itsenäisesti tai suoraan käyttää CASS-aseman laitteistoa. FEPin tehtävä on ainoastaan käsitellä ATLASilla toteutetun testiohjelman sille välittämää tietoa ja vastata testiohjelmalle. Näillä toimenpiteillä voidaan varmistaa se, että testiohjelman suorituksen loputtua CASS-aseman laitteet palautetaan oikeaan tilaan ja esimerkiksi virtalähteet on sammutettu testaamisen loputtua. Estämällä FEPin pääsy käsiksi tiedostojärjestelmään estetään myös tiedostojärjestelmälle mahdollisesti aiheutuvat haitat erilaisissa vikatilanteissa. Tiedostojärjestelmään pääsyn estäminen tarkoittaa esimerkiksi sitä, että FEPin käyttämät parametrit tulevat ATLASilla toteutetulta testiohjelmalta, eivätkä CASS-asemassa sijaitsevista tiedostoista. [4]

FEPin toteutuksessa tulee huomioida ohjelmistossa mahdollisesti aiheutuvat poikkeukset sekä virhetilanteet, niihin reagointi ja niiden käsittely. ATLASilla toteutetut testiohjelmat eivät osaa käsitellä poikkeuksia ja toipua niistä. ATLASilla toteutettu testiohjelma havaitsee virhetilanteita kuten esimerkiksi väärän tyyppisen syötteen, mutta tietoa tästä virhetilanteen aiheutumisesta ei voida välittää testiohjelmalle millään tavalla. Mikäli virhetilanne siis tapahtuu, niin testiohjelma antaa tästä ilmoituksen käyttäjälle, jonka jälkeen testiohjelma joko keskeytetään tai ohjelman suoritusta jatketaan käyttäjän valinnan mukaan, aivan kuten virhetilannetta ei olisi tapahtunut. Muilla ohjelmointikielillä toteutetut FEPit voivat havaita poikkeuksia ja myös toipua niistä. Tieto FEPin suorituksen aikana tapahtuneesta poikkeuksesta voidaan välittää FEPin rajapinnan kautta ATLASilla toteutetulle testiohjelmalle ja tätä kautta jatkaa ohjelman suorittamista hallitusti myös virhetilanteiden jälkeen. FEPin rajapintaan tulee siis tämän vaatimuksen perusteella asettaa oma muuttuja mahdolliselle poikkeukselle, jotta testiohjelma osaa reagoida oikein näissä tilanteissa.

FEPin suunnittelussa tulee ottaa huomioon TPS:lle asetetut testausvaatimukset, joilla varmistetaan riittävän kattava UUT:n (engl. Unit Under Testing) testaaminen sekä testiohjelman oikea toiminta kaikissa tilanteissa. Yksi TPS:n testausmenetelmä, jolle on valmiiksi annettu läpäisykriteerit, on virheiden kylväminen testiohjelmaan. TPS:n testaamiselle asetetut kriteerit virheiden kylvämistä käytettäessä määrittelevät, että testiohjelman tulee havaita oikein 85 % kylvetyistä virheistä ja tunnistaa oikein 90 % kylvetyistä virheistä. Tämän lisäksi kriteerit määrittelevät, että havaittujen alkuperäisten vir-

heiden määrä on alle kaksi kertaa kylvettyjen virheiden määrästä. Mikäli nämä kriteerit eivät täyty, testiohjelma hylätään ja palautetaan korjattavaksi. FEPin käyttötarkoituksesta ja ominaisuuksista riippuu se, millainen testaus on sopiva ja järkevää toteuttaa kyseiselle FEPille. Suurin osa TPS:n testaamiselle asetettavista vaatimuksista käsittelee mitattaville suureille asetettuja toleransseja, eivätkä näin ollen vaikuta suoraan FEPin toteuttamiseen. [16]

2.4 Ongelman asettelu ja tavoiteltavat hyödyt

Suurimman haasteen tässä työssä tulee asettamaan MFCASS-ympäristössä olevan käännösympäristön puuttuminen. Käännösympäristön puuttumisen johdosta virtuaalisen käännösympäristön pystyttäminen ennestään tuntemattomalle järjestelmälle tulee olemaan yksi keskeisimpiä työn kohteita. C-kielen sekä Windows-ympäristössä toimimisen ei oleteta asettavan yhtä suuria haasteita FEPin valmistamiselle, kuin OpenVMS-alustaa käyttävässä virtuaalisessa ympäristössä toimimisen. Näistä seikoista johtuen työ tullaan aloittamaan selvittämällä, miten virtuaalinen ympäristö esimerkiksi C-ohjelmointikielen kääntämiseen saadaan käyttöön. C-kieltä käyttämällä voitaisiin samaa FEPin ohjelmakoodia ainakin osittain käyttää molemmissa CASS-asemissa. Käännösympäristö tullaan toteuttamaan MFCASS-aseman tapauksessa myös Fortran-kielelle, sillä tällä kielellä on tarjolla lähdekoodiesimerkkejä. Fortran-kääntäjän lisääminen järjestelmään ei uskota juurikaan lisäävän työmäärää, mikäli C-kääntäjä onnistutaan järjestelmään asentamaan. Mikäli samaa ohjelmakoodia voidaan käyttää molemmissa CASS-asemissa, on työkuorma FEPin valmistamiselle pienempi ja tästäkin syystä tutkielman teko on hyvä aloittaa selvittämällä C-kielen ja virtuaalisen järjestelmän käyttömahdollisuudet ensimmäisenä.

ATLASilla toteutetun testiohjelman kääntäminen voidaan suorittaa molemmilla CASS-asemilla, mutta MTPSI-tiedoston luominen onnistuu ainoastaan MFCASS-asemalla. Mikäli MFCASS-asema voidaan virtualisoida, voidaan tämäkin toiminta tehdä MFCASS-asemasta riippumattomaksi.

MFCASS-aseman ATLAS-kielen kääntäjän antamat virheilmoitukset ovat huomattavasti kattavampia kuin RTCASS-aseman kääntäjän. RTCASS-asemalla olevan ATLAS-kääntäjän antama virheilmoitus on pahimmassa tapauksessa vain ilmoitus virheiden määrästä, joten virheiden etsiminen koodista on todella työlästä. MFCASS-asemalla suoritettava käännös antaa huomattavasti selkeämmät tiedot epäonnistuneen käännöksen syistä ja näin ollen virheiden korjaaminen on helpompaa sekä nopeampaa. ATLASin kääntäminen RTCASS-asemalla onnistuu myös sellaisissa tilanteissa, joissa for-, while- tai if-lauseesta puuttuu koodilohkon päättävä merkintä [17]. Alla on esitettynä esimerkkikoodi ATLASilla toteutetusta rakenteesta, jossa näkyy tapa, jolla for-silmukan sekä if-vertailun aloitus ja lopetus suoritetaan. Esimerkkikoodista nähdään myös se, että ATLAS-koodi on aina toteutettu isoilla kirjaimilla.

```
1000000   DEFINE, INTEGER, STORE, 'COUNTER' $
1000010   DEFINE, INTEGER, STORE, 'MAX_VALUE' $
1000020   CALCULATE, 'MAX_VALUE' = 5 $
1000030   FOR, 'COUNTER' 1 THRU 10, THEN $
1000040      IF, 'COUNTER' EQ 'MAX' $
1000050         LEAVE, FOR $
1000060      END, IF $
1000070   END, FOR $
```

Mikäli koodi saadaan käännettyä ilman koodilohkon päättäviä merkintöjä, saadaan aikaiseksi todella virhealttiita sovelluksia. Kääntäjä antaa varoituksen, mikäli se havaitsee koodilohkon lopettavan merkinnän puuttuvan, mutta virheellisen testiohjelman suorittaminen on tästä huolimatta mahdollista ilman korjaustoimenpiteitä. MFCASS-asemalla ATLASta käännettäessä, tällaista virhettä ei pystytä käsittelemään ja käännös keskeytyy, eikä virheellistä sovellusta näin ollen pystytä edes suorittamaan. On myös olemassa sellaisia virhetilanteita, joissa RTCASS-aseman kääntäjä ei anna käyttäjälle mitään ilmoitusta, vaan mahdollistaa virheellisen ohjelman suorittamisen. Näitä virhetilanteita ovat CASS-aseman sisäiseen laitteistoon kohdistuvat toimenpiteet, kuten sellaiset muutokset releytyksissä, joita ei todellisuudessa ole mahdollista suorittaa. Tällaisen virheellisen testiohjelman suorittaminen saattaa aiheuttaa vahinkoa käytettävälle laitteistolle.

Osana työtä pyritään myös selvittämään mahdollisuuksia käyttää virtuaalista käännösympäristöä ATLAS-koodin kääntämiseen. Tällä tavalla voitaisiin mahdollisesti parantaa käännöstyökalujen saatavuutta ja jopa osittain korvata kääntämiseen käytettävää rautatason laitteistoa. ALPHA-arkkitehtuurin laitteistojen heikko saatavuus korostaa virtuaalisen ympäristön tutkimisen tärkeyttä, jotta riittävät resurssit ATLAS-koodin kääntämiselle sekä kehittämiselle voidaan taata myös jatkossa. MFCASS-aseman virtualisoinnin etuna olisi myös se, että virheilmoituksiltaan kattavampi kääntäjä saadaan tätä kautta käyttöön riippumatta itse MFCASS-asemasta. Myös MTPSI-tiedostojen luominen voidaan yrittää suorittaa virtuaalisessa ympäristössä ja näin ollen pelkän RTCASS-aseman käyttäminen tuotekehityksessä virtuaalisen ympäristön rinnalla olisi ominaisuuksiltaan riittävä kombinaatio.

Suurimpana hyötynä tässä työssä saadaan selvitettyä, miten FEP voidaan eri CASS-asemille valmistaa ja mitä tähän eri CASS-asemien tapauksissa vaaditaan. Tämän lisäksi tavoitteena on saada valmistettua molemmille asemille testauskäytössä hyödyllinen FEP.

3. MFCASS FEPIN TOTEUTUS

Tässä luvussa käsitellään FEPin luomiseksi vaadittavia toimenpiteitä MFCASS-aseman tapauksessa. Luvussa käsitellään sekä FEPin käännösympäristön pystyttämistä, että itse FEPin luomista. Tutkielman teko aloitetaan selvittämällä, onko mahdollista luoda emulaattorin avulla virtuaalinen käännösympäristö, joka vastaisi MFCASS-aseman ympäristöä. FEPin luominen MFCASS-asemalle pyritään toteuttamaan virtuaalisella käännösympäristöllä. Tällä tavalla saavutetaan riippumattomuutta laitteistosta, estetään kokeellisesta toiminnasta aiheutuvat haitat MFCASS-asemalle, sekä estetään laitteiston kuormittuminen ja tuotantotyön viivästyminen. On myös mahdollista toteuttaa MFCASS-asemassa käytettävän ALPHA-arkkitehtuuria käyttävän laitteiston korvaaminen pelkällä ALPHA-arkkitehtuuria käyttävällä palvelimella, mutta myös tämä menetelmä on laitteistotason toteutuksesta riippuvainen, eikä sitä hyödynnetä tässä tutkielmassa. Tässä luvussa käsitellään syitä käännösympäristön luomisessa suoritettuihin toimenpiteisiin sekä kohdattuja ongelmia ja ratkaisuja niihin. Käännösympäristön luomisen tarkempi ohjeistus ja käyttäjältä vaadittavat toimenpiteet löytyvät liitteestä 1.

3.1 Käännösympäristön luominen

Käännösympäristön luomiseksi MFCASS FEPille tarvitaan virtuaalinen toteutus ALPHA-arkkitehtuurille ja sen päällä suoritettavalle OpenVMS-käyttöjärjestelmälle. ALPHA-arkkitehtuurin emulointiin on tarjolla useita eri emulaattoreita. ALPHA-arkkitehtuurille saatavilla olevia emulaattoreita on esiteltynä taulukossa 10. On olemassa myös sellaisia emulaattoreita, joihin MFCASS-asemasta varmuuskopiona otettu käyttöjärjestelmä voidaan siirtää suoraan. Näiden emulaattorien ominaisuudet eivät kuitenkaan riitä ominaisuuksiltaan tämän työn toteuttamiseen. Varmuuskopiossa käytetty tietorakenne estää tämän menetelmän käyttämisen suurimmassa osassa emulaattoreista. Varmuuskopiossa käytettävää tiedostorakennetta pystyy hyödyntämään ainakin FreeAXP-emulaattori.

Taulukko 10 ALPHA emulaattorit

Emulaattori	Isäntäjärjestelmän käyttöjärjestelmä
AlphaVM	Windows ja Linux
Avanti	Windows
CharonAXP	Windows
ES40	Windows
FreeAXP	Windows

Isäntäjärjestelmällä tarkoitetaan sitä käyttöjärjestelmää, jonka päällä emulaattoria suoritetaan. Tutkielman teon aikana taulukosta 10 kokeiltiin testikäytössä emulaattoreita AlphaVM ja FreeAXP. Valituista emulaattoreista oli saatavilla ilmaisversiot rajoitetuilla ominaisuuksilla. Rajoitetut ominaisuudet koskevat esimerkiksi emuloitavan järjestelmän muistin määrää sekä emuloitavien massamuistien määrää. Tässä työssä selvitetään emulaattorille asetettavat vaatimukset, joita ilman emulaattorin käyttäminen on hyvin hankalaa tai mahdotonta.

Yksi emulaattorille asetettavista vaatimuksista on, että yhteys emulaattorin ja isäntäjärjestelmän välillä tulee voida muodostaa verkkoyhteyden välityksellä. Verkkoyhteyttä tarvitaan tiedostojen siirtämiseen isäntäjärjestelmän ja emulaattorissa suoritettavan käyttöjärjestelmän välillä. Tiedostojen siirtäminen emulaattorissa suoritettavaan käyttöjärjestelmään onnistuu myös luomalla levykuvatiedosto ja ottamalla tämä levykuvatiedosto käyttöön emuloitavassa järjestelmässä. Tätä menetelmää hyödyntämällä suoritetaan myös itse käyttöjärjestelmän käyttöönotto ja asentaminen emulaattoriin. Luotavat levykuvat ovat kuitenkin kirjoitussuojattuja tiedostoja ja niitä emuloidaan CD-asemana. Näin ollen niitä käyttämällä ei voida siirtää tiedostoja toiseen suuntaan eli emulaattorilta isäntäjärjestelmälle. Tiedostojen siirtoa verkkoyhteyden yli tarvitaan esimerkiksi kääntäjän asennustiedostojen siirtämiseen isäntäjärjestelmästä emulaattorilla suoritettavaan järjestelmään sekä käännetyn ohjelman (FEP) siirtämiseen emulaattorilta isäntäjärjestelmään.

Toinen emulaattorille asetettavista vaatimuksista on mahdollisuus luoda järjestelmään useita massamuisteja. Yksi vapaa massamuisti tarvitaan käyttöjärjestelmän, OpenVMS:n, asentamista varten. Tämän lisäksi on hyvä olla olemassa myös toinen massamuisti, johon käyttäjä voi tehdä omat tallennuksensa. Näin voidaan erottaa ja suojata käyttöjärjestelmän käyttämä osio käyttäjän suorittamilta toimilta.

Verkkoyhteyden muodostaminen isäntäjärjestelmän ja emulaattorin välille voidaan muodostaa ainakin kahdella eri tavalla. Verkkoliikennettä voidaan ohjata järjestelmien välillä käyttämällä hyväksi TCP/IP (engl. Transmission Control Protocol/Internet Protocol) mallin IP:n broadcast-osoitetta. Tätä osoitetta käytettäessä kaikki verkon laitteet saavat lähetettävän viestin vastaanotettua [18]. Tämän menetelmän käyttäminen ei kuitenkaan olisi järkevää, kun järjestelmä kytketään osaksi suurempaa verkkoa. Broadcast-osoitteen turhalla käytöllä aiheutettaisiin ylimääräistä kuormitusta verkolle sekä mahdollisia vikatilanteita.

Toinen vaihtoehto verkkoyhteyden muodostamiseen emulaattorissa suoritettavan järjestelmän ja isäntäjärjestelmän välillä on lisätä isäntäjärjestelmään myös ylimääräisen verkkolaitteen emulointi. Verkkolaitteen emulointi on helpommin toteutettavissa Linux-käyttöjärjestelmässä kuin Windows-käyttöjärjestelmässä. Tämä seikka johti siihen, että työtä jatkettiin käyttämällä AlphaVM-emulaattoria, sillä muut saatavilla olevista emulaattoreista eivät tue suorittamista Linux-ympäristössä. AlphaVM:stä löytyy myös tuki

toiselle asetetuista vaatimuksista, eli tuki useamman massamuistin luomiseen. Tutkielmassa käytetty versio AlphaVM-emulaattorista on 1.5.17. Tämä on kirjoitushetkellä viimeisin vakaa Linux-käyttöjärjestelmälle saatava versio emulaattorista.

Linux-jakeluksi, jonka päällä AlphaVM-emulaattoria ajetaan, valittiin Ubuntu 14.04 LTS 64-bit. Eri Linux-jakeluista Ubuntu valittiin siitä syystä, että se on kirjoittajalle tutuin ja laajasti tuettu. Tutkielmassa käytettäväksi valittu Ubuntun versio on LTS- (engl. Long Time Support) julkaisu, eli sille saatava tuki on taattu kolmeksi vuodeksi julkaisuajankohdasta 17.4.2014 eteenpäin.

Verkkoyhteyden emulointi Linux-ympäristössä suoritetaan käyttämällä hyväksi *tun/tap* (engl. tunnel/network tap) virtuaalista verkkoliityntää. *Tun/tap* verkkoliitynnän luominen voidaan tehdä esimerkiksi *tunctl*-apuohjelmalla.

Ennen emulaattorin käyttöönottoa tarvitaan OpenVMS-käyttöjärjestelmän asennusmedia. OpenVMS on lisenssin alainen ohjelmisto, samoin kuin käyttöjärjestelmässä toimivat kääntäjätkin. Tutkielmassa tarvittavat lisenssit ovat vapaasti saatavilla harrastelijoille HP:n (Hewlett-Packard) kautta ja tämä tutkielma voidaan suorittaa näillä lisensseillä. Mikäli ohjelmistoa luodaan tuotantokäyttöön, hankitaan tarvittavat kaupalliset lisenssit siinä vaiheessa. OpenVMS:n harrastelijalisenssin saamiseksi tulee rekisteröityä DECUServe-yhteisön jäseneksi. DECUServe on verkossa toimiva tukiyhteisö, jonka tietokanta tarjoaa artikkeleita sekä muistiinpanoja VMS:ään liittyvissä ongelmissa. Tietokannan vanhimmat artikkelit ovat jo vuodelta 1987

OpenVMS on mahdollista saada sekä ALPHA- että VAX-arkkitehtuurille. Lisäksi OpenVMS:stä on mahdollista saada ainakin kahta eri versiota, joista uudempi on uusin julkaistu versio. Myös kaikki tutkielman teossa tarvittavat lisäosat ovat saatavissa molemmille arkkitehtuureille ja kaikille saatavilla oleville versioille. OpenVMS-käyttöjärjestelmä toimitetaan käyttäjälle ISO-levykuvatiedostoina ja halutut apuohjelmat käyttäjä saa ladattua zip-paketteina.

3.2 AlphaVM emulaattorin asentaminen

Isäntäjärjestelmässä, johon emulaattori asennetaan, tulee olla riittävä määrä vapaata levytilaa OpenVMS-käyttöjärjestelmän asentamista varten. Tämän lisäksi tulee olla vapaata levytilaa myös käyttäjän asentamille apuohjelmille sekä järjestelmässä tehtäville sovelluksille (FEP). Riittävä levytila riippuu käytettävästä OpenVMS:n versiosta ja sen koosta. Käyttäjän omaan käyttöön varatun tilan kokoa on hankala määritellä ja se riippuu lähinnä siitä, kuinka paljon isäntäjärjestelmän levytilasta halutaan antaa emuloitavan järjestelmän käyttöön.

Emulaattorin käyttö tapahtuu vapaavalintaista pääteohjelmaa käyttämällä. Tutkielmassa käytettiin aluksi *Putty*-päätesovellusta, mutta tästä jouduttiin luopumaan merkkien kai-

uttamisesta johtuvien ongelmien vuoksi. Päätesovellusta käytettäessä, käyttäjällä täytyy olla mahdollisuus muokata ohjelman yhteyden tyyppiä sekä tapaa jolla merkistöä käsitellään. Emulaattorin käyttöön soveltuvat asetukset saadaan toteutettua esimerkiksi *socat* nimisellä sovelluksella. Socat ei ole valmiiksi asennettuna tutkielmassa käytetyssä Ubuntu -käyttöjärjestelmässä ja se tulee asentaa erikseen.

3.3 Kääntäjän asennus

Kääntäjän asentaminen OpenVMS-käyttöjärjestelmään vaatii kääntäjän asennustiedostojen siirtämisen käyttöjärjestelmään sekä tarvittavien lisenssitietojen syöttämisen. Lisenssitietoja voidaan syöttää järjestelmään joko käsin, siirtämällä ne tiedostossa verkon yli tai tekemällä levykuvatiedosto ja liittämällä se järjestelmään. Kääntäjän tarvitsemat asennustiedostot tulee siirtää järjestelmään verkon yli tai levykuvatiedoston avulla. Jotta tiedostoja voidaan siirtää OpenVMS-käyttöjärjestelmään verkon yli, on OpenVMS:ssä otettava käyttöön FTP-tiedonsiirtoprotokolla. FTP:n käyttöönotto vaatii sekin lisenssitietojen syöttämisen järjestelmään ja tässä tapauksessa se joudutaan tekemään käsin.

Verkkoyhteyden lisenssitiedot ovat käyttäjälle toimitetussa lisenssiluettelossa nimellä *UCX*. Tämä lisenssi sisältää tarvittavat valtuudet TCP/IP-palveluiden käyttöönottoon OpenVMS-käyttöjärjestelmässä. Lisenssin tiedot syötetään DCL-kielellä, joten niiden syöttäminen tiedostoa suorittamalla on oikeastaan DCL:llä toteutetun skriptin suorittamista [19].

Kun verkkoasetusten lisenssitiedot on syötetty järjestelmään, voidaan verkkoyhteys ottaa käyttöön. Verkkoasetuksiin syötettävät asetukset koostuvat esimerkiksi IP-osoitteesta, aliverkon peitteestä ja halutuista verkkopalveluista. Valittavana olevia verkkopalveluita ovat esimerkiksi DHCP (engl. Dynamic Host Configuration Protocol), FTP, SSH (engl. Secure Shell) ja TELNET. Tiedostojen siirtämiseen tarvittava verkkopalvelu on FTP.

Emulaattorin uudelleenkäynnistymisien yhteydessä verkkopalvelun asetukset pysyvät tallennettuina järjestelmään, mutta verkkopalvelut sammuvat. Mikäli verkkopalveluita halutaan käyttää uudelleenkäynnistämisen jälkeen, pitää ne käynnistää uudelleen. Verkkopalveluiden käynnistäminen voidaan myös automatisoida muokkaamalla käyttöjärjestelmän käynnistystiedostoja.

Kun OpenVMS:ään siirretään tiedostoja levykuvatiedoston avulla, halutut tiedostot tallennetaan ensin levykuvatiedostoon. Tässä tutkielmassa levykuvatiedostojen luomisessa käytettiin *Ubuntun* tarjoamaa *Brasero*-apuohjelmaa. Levykuvatiedosto voidaan ottaa OpenVMS:ssä käyttöön, lisäämällä se emulaattorin asetustiedostoon ja tämän jälkeen liittämällä levykuva asetustiedostossa käytetyn laitenimen perusteella. Tutkielmassa käytetyt kääntäjät siirretään järjestelmään levykuvatiedostoa käyttämällä. Käytetyt kääntäjät versioineen sekä kääntäjien asennustiedostot on esiteltynä taulukossa 11.

Taulukko 11 Kääntäjien versiot sekä asennustiedostot

Kääntäjä	Asennus tiedostot
C V7.3	*cc073.a* *cc073.b*
Fortran V8.2	*dec-axpvms-fortran-v0802-0-1.pcsi$compressed* *dec-axpvms-fortran-v0802-0-1.pcsi$compressed_esw*

Kun levykuvatiedosto liitetään OpenVMS-järjestelmään ja levyllä olevia tiedostoja pyritään käyttämään, saadaan esimerkiksi C-kääntäjän asennustiedoston *cc073.a* kohdalla virheilmoitus:

```
%BACKUP-F-NOTSAVESET, DKA100:[000000]CC073.A;1 is not a BACKUP
save set
%VMSINSTAL-E-NOSAVESET, Save set  A  cannot be restored.
```

Tämä virheilmoitus kertoo, että tiedosto on korruptoitunut eikä sitä voida käyttää. Tiedoston suorittaminen epäonnistuu, vaikka tiedosto siirretäisiin levykuvatiedostosta sellaiselle levylle, johon käyttäjällä on kirjoitusoikeudet. Fortranin kohdalla asennustiedostojen nimet ovat muuttuneet levykuvatiedostossa muotoon *dec_axpvms_fortran_v0802_00.pcs;1* ja *dec_axpvms_fortran_v0802_01.pcs;1*. Tästä syystä tiedostot nimetään uudelleen samalla, kun ne kopioidaan sellaiseen sijaintiin, johon käyttäjällä on kirjoitusoikeudet.

Tiedoston korruptoituminen voidaan yrittää korjata säätämällä esimerkiksi tiedoston MRS- (engl. Maximum Record Size) sekä LRL-parametreja (engl. Longest Record Length). C-kääntäjän tiedostot saadaan parametreja säätämällä korjattua ja kääntäjä voidaan asentaa järjestelmään. Fortran-kääntäjän kohdalla tiedoston parametrien säätämisestä huolimatta, tiedostoja ei saada suoritettua OpenVMS:ssä, vaan edelleen saadaan sama virheilmoitus.

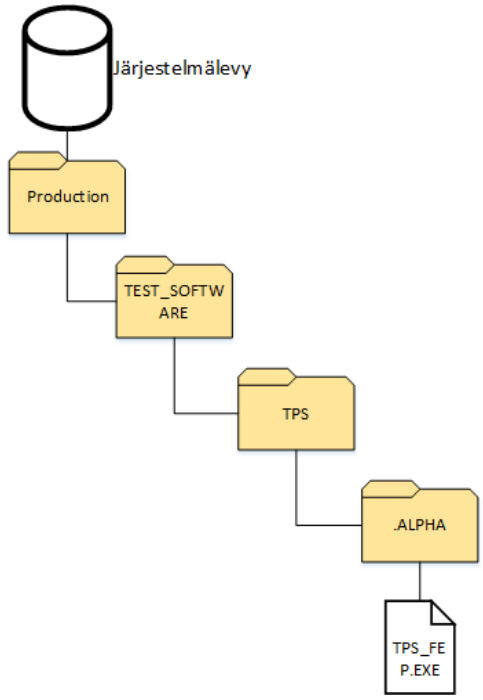
Tiedostojen korruptoitumista koskevat virheilmoitukset saadaan myös siinä tapauksessa, että Fortran-kääntäjän asennustiedostot siirretään OpenVMS:ään käyttämällä FTP:tä. Tämän siirron jälkeen tiedoston korruptoituminen saadaan kuitenkin korjattua komennolla, joka ei levykuvatiedostoa käytettäessä toiminut. Näin ollen molempien kääntäjien asentaminen saadaan suoritettua onnistuneesti. Siirrettäessä binääri- tai suoritettavia tiedostoja FTP:n yli, tulee tiedonsiirron tyyppi määritellä näiden tiedostojen tyypille sopivaksi [20].

Asennuksen jälkeen tarkistettiin molempien kääntäjien versiotiedot. Asennetun C-kääntäjän versio on *V7.3-009* ja Fortran kääntäjän versio *V8.2*. Kääntäjien asentamisen jälkeen voidaan tehdä yksinkertainen testiohjelma kääntäjän toiminnan varmistamiseksi. Tietotekniikassa perinteisellä Hello World! -testiohjelmalla saadaan varmistettua kääntäjän toiminta järjestelmässä.

Kun testiohjelma oli saatu onnistuneesti käännettyä sekä suoritettua emulaattorissa, se siirretään MFCASS-asemalle testattavaksi. Käännetyn testiohjelman siirto emulaattorista MFCASS-asemalle suoritetaan käyttäen FTP-yhteyttä. Jälleen siirrettäessä suoritettavaa tiedostoa tulee FTP-yhteyden tyyppi asettaa siirrettäville tiedostoille sopivaksi. Siirron jälkeen tehdyssä testiajossa voidaan testiohjelman todeta toimivan MFCASS-asemalla täysin samalla tavalla kuin emulaattorissa.

3.4 FEPin luominen

Kun ollaan luomassa MFCASS-asemalle FEPiä, niin aluksi tulee päättää, onko tämä FEP käytössä vain yhdessä testiohjelmassa, vai halutaanko tätä FEPiä käyttää laajemmin ja useissa eri testiohjelmissa. Mikäli FEP halutaan käyttöön vain yhdessä testiohjelmassa, niin valmistettavan FEPin sijainti järjestelmässä tulee olla kuvan 4 mukainen. Mikäli käytössä olisi aikaisempi Block I-mallin MFCASS-asema ja VAX-arkkitehtuuri, niin FEPin sijainti olisi muuten sama, mutta alimpana oleva kansio olisi *.VAX*. Kansiorakenteessa merkintä *TPS* viittaa jokaiselle testiohjelmalle annettavaan yksilölliseen tunnisteeseen.

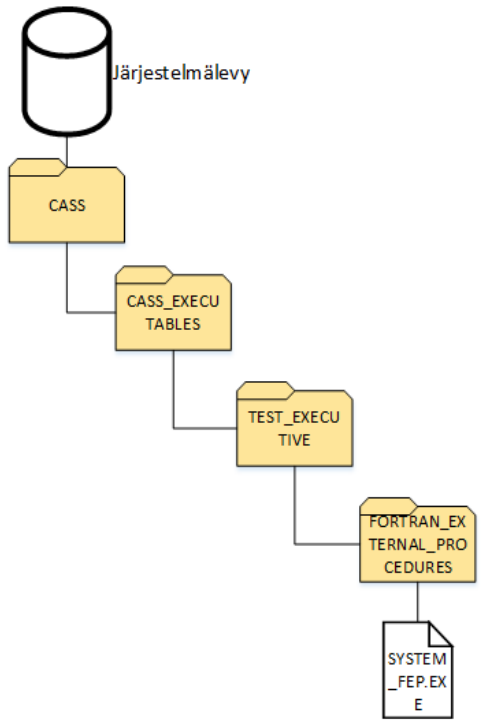


Kuva 4 TPS FEP MFCASS-asemalla

Mikäli valmistettava FEP on sellainen, että sitä halutaan käytettävän useassa eri TPS:ssä, niin FEPin sijainti järjestelmässä tulee olla kuvan 5 kaltainen. Vaikka FEPin toteutuskieli olisi jokin muu kuin Fortran, niin FEP sijoitetaan silti kansioon *FORTRAN_EXTERNAL_PROCEDURES*.

Lähdemateriaaliksi FEPin luomisessa MFCAS-asemalle on käytettävissä seitsemän lähdekoodiesimerkkiä, jotka on toteutettuna Fortran 77:lla. Yksi näistä valmiista esimerk-

kikoodeista siirretään emulaattoriin kääntämistä varten. Esimerkkikoodissa FEPille välitetään kaksi taulukkoa jotka sisältävät kokonaislukuja. Nämä kokonaisluvut lasketaan yhteen ja palautetaan yhdessä taulukossa. Valmista esimerkkikoodia on muokattava sisennysten osalta useilta eri riveiltä. Alkuperäisessä esimerkkikoodissa sisennykset ovat väärin ja tästä syystä kääntäjä antaa useita virheilmoituksia.



Kuva 5 System FEP MFCASS-asemalla

Fortranin syntaksissa on muutamia huomion arvoisia seikkoja, jotka käytettävissä olevissa lähdekoodeissa tulee huomioida. Pääsääntöisesti ensimmäiset kuusi saraketta lähdekoodissa jätetään tyhjiksi. Tästä poikkeuksena kuitenkin sellaiset rivit, joissa komento jatkuu usealle riville, tai käytetään sisäkkäisiä silmukkarakenteita. Mikäli komento jatkuu usealle eri riville, tulee tämä merkitä sarakkeeseen kuusi. Komennon jatkumisen merkkinä voidaan käyttää mitä tahansa merkkiä. Hyvä tapa merkitä komennon jatkuminen on esimerkiksi nouseva numerointi, &- tai +-merkki. Esimerkki komennon jatkumisesta seuraavalle riville sekä sisäkkäisten silmukoiden käytöstä on esitetty kohdassa Ohjelma 1.

```
          DO WHILE (.TRUE.)
2          IF (PROBLEM)
       1        CALL LIB$FUCN(PARAMETER1,%VAL(1),
4      2        'ARGUMENT ERROR')
           I = 1
6          DO 10 I = 1,100
              INT (I) = NUM1 (I) + NUM2 (I) + NUM1 (I)
8   10     END DO
          ENDDO
```

Ohjelma 1. *Fortran syntaksiesimerkki*

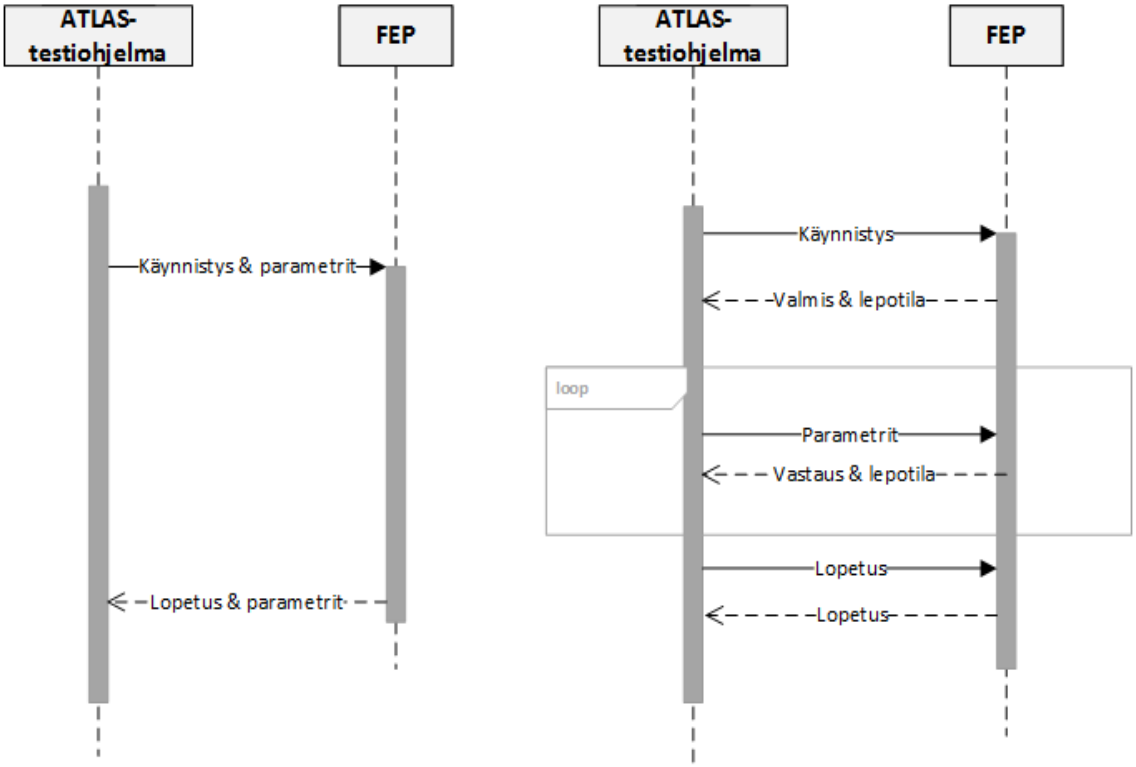
Do-silmukalle annetaan Fortranissa tunnistenumero, mikäli do-silmukoita on useita sisäkkäin. Numeroinnin tulee olla uniikki ja se sijoitetaan silmukan alkuun sekä loppuun. Do-silmukan lopetusehdon riville asetettava tunniste tulee olla sarakkeissa 2-5. Esimerkki tällaisesta Do-silmukan tunnisteesta nähdään ohjelma 1:n riveillä 6 ja 8. Riveiltä 8 ja 9 nähdään myös Fortranin ominaisuus, jonka mukaan avainsanojen välissä ei tarvitse olla välilyöntiä. Näin ollen lauseet *END DO* sekä *ENDDO* ovat täysin sama asia, vaikka kirjoitusasut poikkeavat toisistaan. Fortran 77:ssa lähdekoodista voidaan jopa poistaa kaikki välilyönnit, kunhan huolehditaan vain oikeista sisennyksistä, ja silti suorittaa onnistunut käännös [9].

Kun esimerkki-FEPin lähdekoodi on sisennetty oikein, se kääntyy Fortran-kääntäjällä ilman virheilmoituksia. Yritettäessä linkittää tämä koodi emulaattorissa, saadaan ohjelma 2:n kaltainen virheilmoitus. Virheilmoituksesta voidaan päätellä, että kaikkia tarvittavia kirjastoja ei ole saatavissa virtuaalisessa ympäristössä linkittämisen suorittamiseksi. *SCFEM*-alkuiset ulkoiset aliohjelmat kuuluvat kirjastoon *SCFE.OLB* [4]. Tietoa tämän kirjaston sijainnista ei ole, eikä sitä MFCASS-aseman järjestelmälevyltä löydy. *SCFE.OLB*-kirjasto tarjoaa rajapinnan FEPin ja ATLASilla toteutetun testiohjelman välillä. Tämän kirjaston olemassaolo on FEPin toteuttamiseksi kriittisen tärkeää ja sen puuttuminen tekee FEPin toteuttamisesta MFCASS-asemalle mahdotonta.

```
   %LINK-W-NUDFSYMS, 7 undefined symbols:
 2 %LINK-I-UDFSYM, LIB$ESTABLISH
   %LINK-I-UDFSYM, MSG_LITERAL
 4 %LINK-I-UDFSYM, SCFEM_GET_INT2
   %LINK-I-UDFSYM, SCFEM_PUT_INT2
 6 %LINK-I-UDFSYM, SCTEM_FECONDHNLR
   %LINK-I-UDFSYM, SCTEM_FE_INIT
 8 %LINK-I-UDFSYM, SCTEM_FE_PRLINE
   %LINK-W-USEUNDEF, undefined symbol SCTEM_FECONDHNLR referenced
10       in psect $LINK$ offset %X00000030
         in module INT_SAMPLE file DKA0:[FEP]SAMPLE_FEP.OBJ;9
12 %LINK-W-USEUNDEF, undefined symbol SCFEM_GET_INT2 referenced
         in psect $LINK$ offset %X00000040
14       in module INT_SAMPLE file DKA0:[FEP]SAMPLE_FEP.OBJ;9
   %LINK-W-USEUNDEF, undefined symbol SCTEM_FE_PRLINE referenced
16       in psect $LINK$ offset %X00000050
         in module INT_SAMPLE file DKA0:[FEP]SAMPLE_FEP.OBJ;9
18 %LINK-W-USEUNDEF, undefined symbol MSG_LITERAL referenced
         in psect $LINK$ offset %X00000060
20       in module INT_SAMPLE file DKA0:[FEP]SAMPLE_FEP.OBJ;9
   %LINK-W-USEUNDEF, undefined symbol LIB$ESTABLISH referenced
22       in psect $LINK$ offset %X000000C0
         in module INT_SAMPLE file DKA0:[FEP]SAMPLE_FEP.OBJ;9
24 %LINK-W-USEUNDEF, undefined symbol SCTEM_FE_INIT referenced
         in psect $LINK$ offset %X000000E0
26       in module INT_SAMPLE file DKA0:[FEP]SAMPLE_FEP.OBJ;9
   %LINK-W-USEUNDEF, undefined symbol SCFEM_PUT_INT2 referenced
30       in psect $LINK$ offset %X000000F0
         in module INT_SAMPLE file DKA0:[FEP]SAMPLE_FEP.OBJ;9
```

Ohjelma 2. *Virheilmoitus*

Käytettävissä olevan esimerkkiohjelman rakenteesta voidaan selvittää, millainen FEPin rakenteen tulisi olla, kun sitä suoritetaan MFCASS-asemalla. Lähdekoodissa määritellyt ulkoiset aliohjelmat *SYS$HIBER* ja *SYS$SETEF* ovat OpenVMS:n omia rutiineita. *SYS$SETEF* on rutiini, jolla asetetaan käyttöjärjestelmän lippu haluttuun arvoon ja ilmoitetaan tätä lippua odottavalle sovellukselle, että se voi jatkaa toimintaansa [21]. Tällä lipulla ilmoitetaan esimerkin tapauksessa ATLASilla toteutetulle testiohjelmalle, että FEP on käynnistetty ja valmiina suoritettavaksi. *SYS$HIBER*-rutiini asettaa suoritettavan sovelluksen lepotilaan ja odottamaan kutsua käyttöjärjestelmältä. Suoritettava sovellus eli FEP, odottaa lepotilassa kutsua käyttöjärjestelmältä ja ATLASilla toteutetulta testiohjelmalta, kunnes sille ollaan valmiita välittämään parametreja. Kuvassa 6 on vasemmalle esitettynä toimintalogiikka joka FEPillä oletettiin olevan ja kuvassa oikealla se logiikka, joka FEPillä todellisuudessa on.



Kuva 6 FEPin toimintalogiikka

FEPin todellinen rakenne ja lähdekoodin riippuvuus käyttöjärjestelmäkohtaisista rutiineista aiheuttaa sen, että saman toiminnan toteuttaminen Windows-ympäristössä ja RTCASS-asemalla olisi hankalaa ja täysin erilaisella lähdekoodilla toteutettu. Kirjaston *SCFE.OLB* käyttöjärjestelmästä riippuvia rutiineita jouduttaisiin käyttämään MFCASS-asemalla toteutuskielestä riippumatta. Mikäli FEP yritettäisiin toteuttaa C-kielellä, joka on ainoa molempien CASS-asemien tukema ohjelmointikieli, käyttöjärjestelmästä riippuvia rutiineita jouduttaisiin käyttämään myös tässä tapauksessa. Tästä syystä ei ole järkevää yrittää toteuttaa FEPiä samalla lähdekoodilla molemmille CASS-asemille, sillä osa lähdekoodista jouduttaisiin joka tapauksessa toteuttamaan eri tavoilla.

Linkkeri on kääntäjästä täysin erillinen ohjelmisto ja se löytyy myös MFCASS-asemasta. Linkkerin päätehtävä on muodostaa suoritettavia kuvatiedostoja yhdistämällä dataa ja binäärikoodia kuvatiedostoksi [22]. Mikäli esimerkkilähdekoodi käännetään emulaattorissa olevalla Fortran-kääntäjällä ja siirretään tämän jälkeen linkitettäväksi MFCASS-asemaan, niin saadaan toisenlainen virheilmoitus kuin emulaattorissa. Liitteessä A on esitettynä MFCASS-asemalla linkittämisestä saatavat virheilmoitukset.

Saadut virheilmoitus aiheutuvat erilaisesta tiedostorakenteesta kuin mitä linkkeri odottaa saavansa. OpenVMS:n dokumentaatiossa tähän ongelmaan tarjotaan ratkaisuna lähdekoodin uudelleen kääntämistä [23]. Esimerkin tapauksessa uudelleen kääntämisestä ei kuitenkaan ole apua, vaan sama virheilmoitus saadaan myös uudelleen kääntämisen jälkeen suoritettaessa linkittäminen MFCASS-asemalla.

Johtuen kohdatuista ongelmista ja näiden selvittämisen asettamista työmääristä, lopetetaan FEPin luominen MFCASS-asemalle. Samalla tutkielmassa päätettiin asettaa RTCASS-asemalle kehitettävän FEPin lähtökohdaksi apuohjelma FEP Wizardin käyttäminen. FEP Wizardin käyttäminen on perusteltua, sillä samaa lähdekoodia ei voida hyödyntää molemmilla CASS-asemilla, vaikka MFCASS-asemalla kohdattuihin ongelmiin löydettäisiinkin ratkaisu.

ATLAS-koodin kääntämiseen tai MTPSI-tiedoston luomiseen käytettäviä työkaluja ei tässä vaiheessa yritetä asentaa virtuaaliseen ympäristöön. Näiden toimenpiteiden suorittaminen vaatisi huomattavasti lisää perehtymistä siihen, mitä kaikkea näiden työkalujen käyttöönotto virtuaalisessa järjestelmässä vaatii. Tämän tutkielman aikataulu ei riitä näiden toimenpiteiden riittävän syvälliseen perehtymiseen, joten tästä syystä nämä asiat jätetään myöhempää tutkimusta varten avoimiksi.

4. RTCASS FEPIN TOTEUTUS

Tässä luvussa käsitellään vaadittavia toimenpiteitä FEPin luomiseksi RTCASS-asemalle. Luvussa käsitellään sekä FEPin käännösympäristön pystyttämistä, että itse FEPin luomista. Johtuen FEPin luomisessa kohdatuista ongelmista MFCASS-asemalle, on järkevintä toteuttaa FEPin luominen RTCASS-asemalle käyttäen hyväksi FEP Wizard -apuohjelmaa ja toteutuskielenä C#-ohjelmointikieltä.

4.1 Käännösympäristön luominen

FEPin luomiseen käytettävät työkalut RTCASS-asemalla ovat osa apuohjelmakokonaisuutta, joilla suoritetaan myös esimerkiksi ATLAS-lähdekoodin kääntäminen RTCASS-asemalla. Kääntämiseen käytettävät työkalut eivät itsessään aseta suuria vaatimuksia ympäristölle, jossa niitä käytetään, mutta muut samassa asennuskokonaisuudessa tulevat työkalut sen sijaan toimivat paremmin tietyssä ympäristössä. Jotta myös kaikki muut kuin FEPin luomiseen käytettävät työkalut saadaan käyttöön, niin käännösympäristön luominen toteutetaan Windows XP -käyttöjärjestelmään. Taulukossa 12 on esiteltynä FEPin luomista varten asennettavat sovellukset ja järjestelmät.

Taulukko 12 Vaaditut työkalut FEPin toteutukseen RTCASS-asemalla [11]

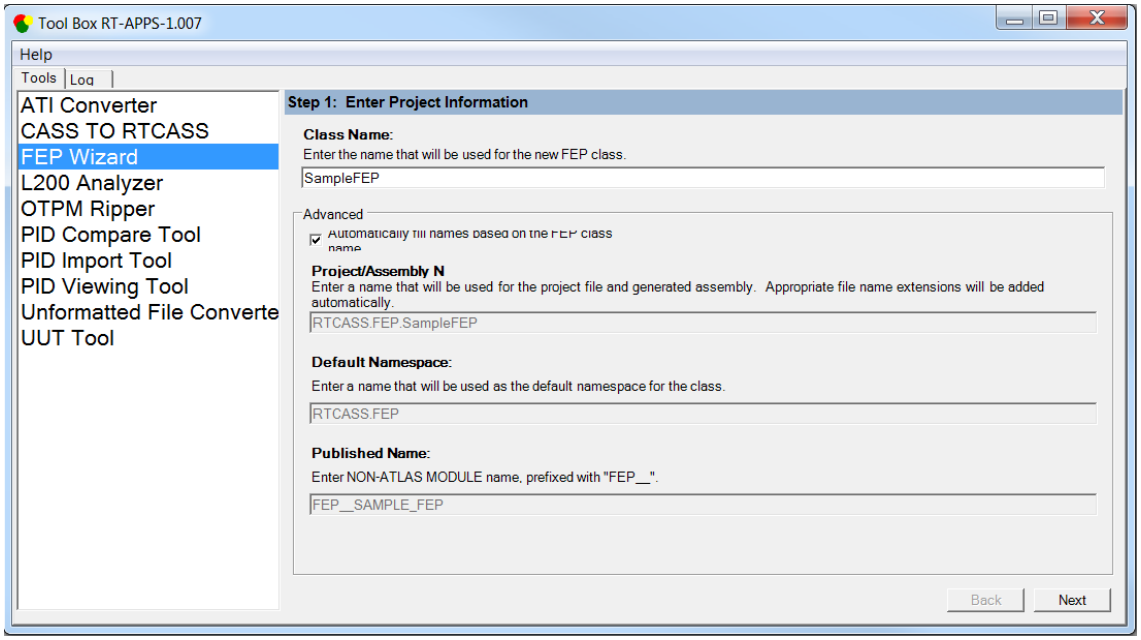
Asennettava työkalu	Selite
Microsoft Windows XP	Asennusympäristön käyttöjärjestelmä
Microsoft .NET Framework 1.0	Vaadittava sovelluskehys
Microsoft Visual Studio 2003	Käännösympäristö C#-projektille
RTCASS Tools	FEP Wizardin sisältävä ohjelmistokokonaisuus

FEPin luomiseen vaadittavien työkalujen asennusta ei käsitellä tässä dokumentissa erikseen, sillä niiden asennus ei vaadi erityisiä taitoja Windows-käyttöjärjestelmän käyttöön tottuneelta käyttäjältä. Asennusjärjestyksessä käyttäjän tulee huomioida, että .NET Framework 1.0 pitää asentaa ennen RTCASS Toolsia. Mikäli käyttäjä yrittää asennusta eri järjestyksessä, käyttäjälle annetaan virheilmoitus, josta oikea asennusjärjestys käy ilmi.

4.2 FEPin luominen

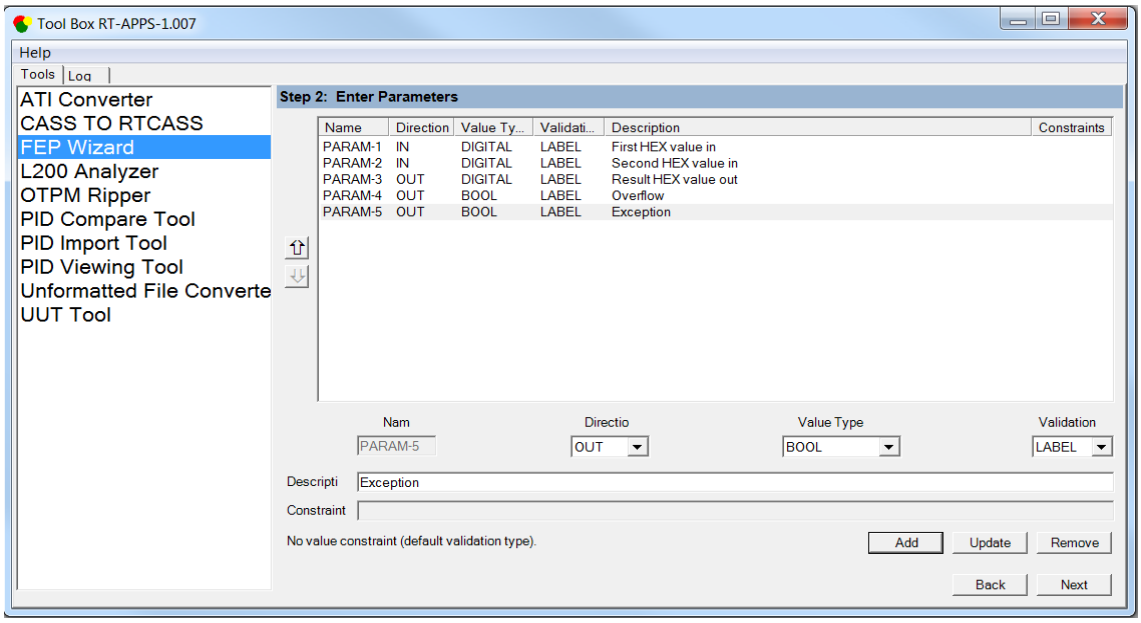
FEP Wizard on graafinen käyttöliittymä C#-projektin luomiseen ja sillä voidaan muodostaa FEPin ja ATLASilla toteutetun testiohjelman välinen rajapinta. FEP Wizardia päästään käyttämään käynnistämällä asennettu RTCASS Tools -sovellus. Tämän jälkeen käyttäjän tulee valita valikosta käytettäväksi FEP Wizard. FEPin luominen

RTCASS-asemalle aloitetaan FEP Wizardilla kuvassa 7 esitetyllä tavalla. Ensimmäisessä ikkunassa FEPin toteuttavalle luokalle annetaan haluttu nimi. Tässä tapauksessa FEPin luokan nimeksi on asetettu *SampleFEP*. Tässä vaiheessa muissa kohdissa käytetään oletusasetuksia.



Kuva 7 FEP Wizard, FEPin nimeäminen

Kun FEPin luokan nimi on asetettu, siirrytään seuraavaan vaiheeseen painamalla nappia *Next*. Seuraavassa vaiheessa käyttäjän tulee syöttää FEP Wizardille rajapinta, joka FEPin ja ATLASilla toteutetun testiohjelman välille halutaan toteuttaa. Kuvassa 8 on esitettynä esimerkki FEPin rajapinnasta.



Kuva 8 FEP Wizard, FEPin rajapinta

Esimerkin tapauksessa FEPin rajapinnassa välitetään kaksi parametria, jotka ovat tyypiltään DIGITAL. FEP palauttaa testiohjelmalle yhden *DIGITAL*-tyyppisen parametrin sekä kaksi *BOOLEAN*-tyyppistä parametria. Näillä parametreilla voidaan toteuttaa yksinkertainen FEP, joka laskee kaksi heksadesimaalilukua yhteen ja palauttaa yhteenlaskun summan testiohjelmalle. Testiohjelmalle palautettavien *BOOLEAN*-tyyppisten parametrien avulla voidaan välittää tieto siitä, onko FEPin suorituksen aikana tapahtunut ylivuoto tai jokin poikkeus. *BOOLEAN*-tyyppisten parametrien perusteella ATLASilla toteutettu testiohjelma osaa toimia oikein niissä tilanteissa, joissa virhe tapahtuu.

FEPin rajapinnalle voidaan FEP Wizardissa määritellä useita erityyppisiä muuttujia sekä näistä muuttujista muodostettavia listoja. Taulukossa 13 on esiteltynä FEP Wizardissa käytettävien muuttujien tyypit. Tyypin lisäksi muuttujalle määritellään suunta, joko FEPiltä ATLASilla toteutetulle testiohjelmalle päin (*OUT)* tai testiohjelmalta FEPille päin (*IN*).

Taulukko 13 FEP Wizard, rajapinnan muuttujien tyypit

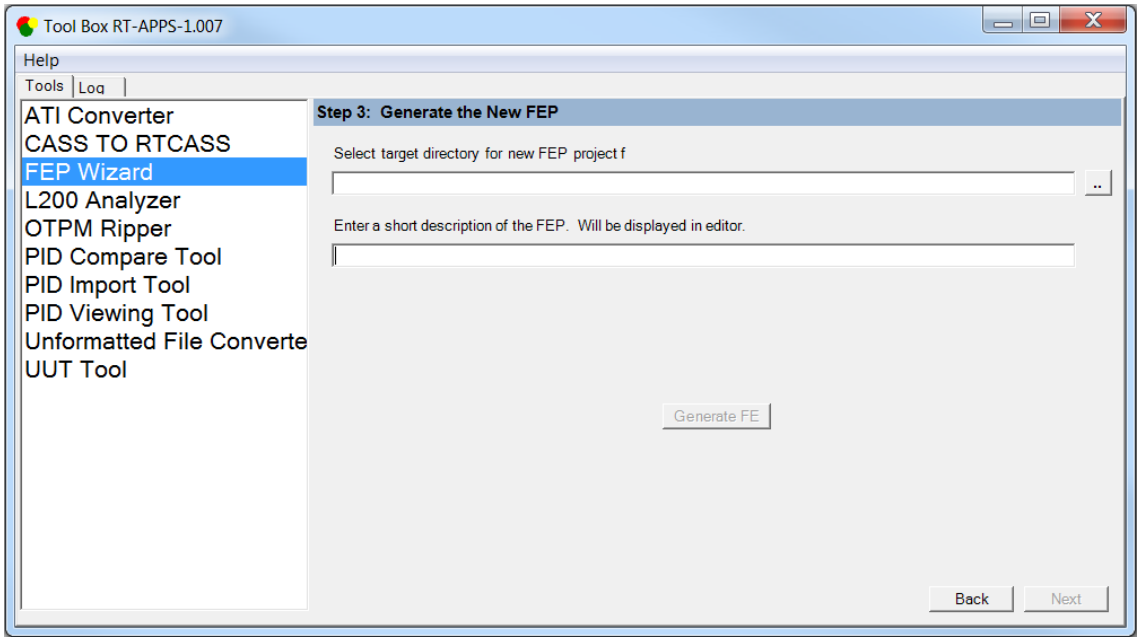
Parametrien tyypit
INT
LONG
BOOL
STRING
FLOAT
DIGITAL
INT_ARRAY
LONG_ARRAY
BOOL_ARRAY
STRING_ARRAY
FLOAT_ARRAY
DIGITAL_ARRAY

FEPin rajapintaan luotaville muuttujille annetaan myös validoinnin tyyppi, jonka avulla varmistetaan muuttujalta halutut ominaisuudet. Taulukossa 14 on esiteltynä eri validointityypit sekä niiden tarkoitus. Validoinnille annettavat parametrit syötetään FEP Wizardissa riville *Constraint*. FEP Wizard antaa myös ohjeistustekstiä eri validointiparametrien syöttämiseksi. Esimerkkitapauksessa parametreilta vaaditaan, että niillä on olemassa jokin arvo asettamalla validoinnin tyypiksi *LABEL*.

Taulukko 14 FEP Wizard, FEPin muuttujien validointi [6]

Validoinnin tyyppi	Selite
LABEL	Tämä validoinnin tyyppi varmistaa, että muuttujalla on jokin arvo. Tämä tyyppi on rajapinnassa oletus.
DIRECTIVE	Tämä validoinnin tyyppi varmistaa ainoastaan sen, että muuttuja on rajapinnassa. Muuttujalta ei vaadita mitään arvoa.
RANGE	Tämä validoinnin tyyppi määrittelee muuttujalle maksimi ja minimi raja-arvot. Raja-arvoista voidaan määritellä myös vain toinen. Muuttujalle annettavat vaihtoehdot riippuvat muuttujan tyypistä.
CHOICE	Tämä validoinnin tyyppi määrittelee muuttujan arvoksi sallitut vaihtoehdot. Sallitut vaihtoehdot riippuvat muuttujan tyypistä.

Kun FEPin rajapinnan parametrit on lisätty, siirrytään seuraavaan vaiheeseen. Rajapinnan luomisen viimeisessä vaiheessa luotavalle C#-projektille valitaan sijainti sekä annetaan kuvaus. Kuvassa 9 on esitettynä näiden tietojen syöttämiseen käytettävät kentät. FEPille annettava kuvaus lisätään kommenttina luotavan luokan tiedostoon. Kuvauksen antaminen ei ole pakollista, mutta helpottaa lähdekoodin tulkintaa myöhemmissä vaiheissa.



Kuva 9 FEP Wizard, FEPin kuvaus ja sijainti

FEP Wizard luo taulukossa 15 esitellyt tiedostot käyttäjän määrittelemään sijaintiin. Luoduista tiedostoista käyttäjälle tärkein on *SampleFEP.cs*. Tähän tiedostoon tullaan tekemään ne muutokset ja se toiminnallisuus, jota FEPin halutaan suorittavan.

Taulukko 15 FEP Wizard, automaattisesti luotavat tiedostot

Tiedoston nimi	Tarkoitus
AssemblyInfo.cs	Tähän tiedostoon käyttäjä voi lisätä esimerkiksi seuraavia FEPin metatietoja: • Configuration • Company • Copyright • Description
DirectTest.Interfaces.dll	ATLASilla toteutetun testiohjelman sekä FEPin välisen rajapinnan käsittelyyn luotu kirjasto. Kaikkien FEPien tulee käyttää samaa versiota tästä kirjastosta.
FEPBase.cs	Pääluokka, joka mahdollistaa FEPin nopean kehittämisen. Sisältää esimerkiksi parametrien validoinnin sekä virheiden kirjaamisen.
FEPConfiguration.cs	Määrittelee mahdolliset FEPissä käytettävät parametrit, niiden suunnat sekä validoinnin tyypit.
FEPEditWindow.cs FEPEditWindow.resx	Mahdollistavat myös graafisen käyttöliittymän käytön FEPissä.
RTCASS.FEP.SampleFEP.csproj	C#-projektitiedosto on XML-tiedosto, joka sisältää tietoa esimerkiksi käytettävistä projektin ulkoisista kirjastoista, projektin muiden tiedostojen nimistä sekä julkaistavan ja debug-version käännöksen eroista.
SampleFEP.cs	Tiedosto johon käyttäjä toteuttaa haluamansa toiminnallisuuden FEPille.

Esimerkin tapauksessa FEP Wizard luo luokkaan *FEPBase* perustuvan luokan, jonka sisältö on esitettynä liitteessä B. Esimerkkikoodissa nähdään FEPin rakentaja riviltä kuusi alkaen. FEPin rakentajassa määritellään rajapinnalle annetut muuttujat riveillä 15-25. FEPin toiminnallisuuden määrittelevä lohko *ExecuteFEP_Logic* alkaa riviltä 33.

Luodut tiedostot otetaan tämän jälkeen käyttöön Visual Studio 2003:ssa avaamalla luotu C#-projektitiedosto. Tämän jälkeen käyttäjän tulee avata tiedosto *SampleFEP.cs* muokattavaksi, jotta FEPin toiminnallisuus voidaan lisätä lähdekoodiin. *SampleFEP.cs*-tiedoston metodi *ExecuteFEP_Logic* muokataan muotoon, joka on esitelty kohdassa Ohjelma 3.

Varsinaisen heksadesimaalien yhteenlaskun lisäksi FEP Wizardin luomaan koodiin joudutaan tekemään joitakin lisäyksiä ja muutoksia. Riveillä 11 ja 13 olevat Boolean-tyyppiset muuttujat tulee alustaa jollakin arvolla, tai Visual Studion kääntäjä antaa näiden muuttujien käytöstä virheen. Rivillä 14 otetaan käyttöön poikkeuksia havaitseva *try*-haara. Kaikki mahdolliset poikkeukset otetaan kiinni riviltä 36 alkavassa *catch*-haarassa. Mikäli poikkeus tapahtuu, asetetaan Boolean-muuttuja *bPARAM_5* todeksi. Kun FEPin toiminta lopetetaan, välittyy tämä asetettu muuttuja ATLASilla toteutetulle testiohjelmalle ja siihen osataan reagoida oikein.

```
protected override object ExecuteFEP_Logic(System.Xml.XmlNode aDataNode)
{
      // Declare variables for input and output parameters.
      // This is an ATLAS DIGITAL type.
      DirectTest.Interfaces.Common.IDigital oPARAM_1;
      // This is an ATLAS DIGITAL type.
      DirectTest.Interfaces.Common.IDigital oPARAM_2;
      // This is an ATLAS DIGITAL type.
      DirectTest.Interfaces.Common.IDigital oPARAM_3;
      // This is an ATLAS BOOLEAN type.
      bool bPARAM_4 = false;
      // This is an ATLAS BOOLEAN type.
      bool bPARAM_5 = false;
      try
      {
            // Fetch the input variables.
            oPARAM_1 =
            (DirectTest.Interfaces.Common.IDigital)GetParameterValue(0);
            oPARAM_2 =
            (DirectTest.Interfaces.Common.IDigital)GetParameterValue(1);
            oPARAM_3 =
            (DirectTest.Interfaces.Common.IDigital)GetParameterValue(1);
            int temp1 = oPARAM_1.GetUInt16() + oPARAM_2.GetUInt16();
            //Biggest 16bit unsigned integer value
            if( temp1 <= 65535 )
            {
                  oPARAM_3.SetValue(temp1);
                  bPARAM_5 = true;
            }
            else
            {
                  bPARAM_4 = true;
                  bPARAM_5 = false;
            }
      }
      catch
      {
            LogError(M_FEP_ERROR, mName, "HEXFEP error", "VKEN",
            "FEP error:Exception throw test");
            bPARAM_5 = true;
            return null;
      }
      // Set output variables.
      SetParameterValue(2, oPARAM_3);
      SetParameterValue(3, bPARAM_4);
      SetParameterValue(4, bPARAM_5);
      return null;
}
```

Ohjelma 3. *Heksadesimaalien yhteenlasku*

LogError on yksi *FEPBase*-luokan funktioista ja sillä käsitellään FEPin suorituksen aikana tapahtuvat poikkeukset sekä käyttäjälle niistä annettavat ilmoitukset. *LogError*-funktion rajapinta on kuvattu taulukossa 16.

Taulukko 16 FEP Wizard, LogError-funktio

#	Tyyppi/Nimi	Tarkoitus
1	`int aSeverity`	Valmiiksi määritellyt vakavuusasteet erityyppisille virhetilanteille. Vakavuusasteet on määritelty taulukossa 17. Tämä näkyy testin suorittajalle.
2	`string aUserFriendlyName`	Käyttäjäystävällinen virheilmoituksen nimi, esimerkiksi FEPin nimi. Tämä viesti näytetään testin suorittajalle.
3	`string aUserFriendlyDescription`	Käyttäjäystävällinen kuvaus virheestä. Tämä viesti näytetään testin suorittajalle.
4	`string aDeveloperName`	FEPin kehittäjän nimi tai muu tunniste. Tätä viestiä ei näytetä testin suorittajalle.
5	`string aDeveloperDescription`	Kattavampi virheilmoitus kehittäjän käytettäväksi. Tätä viestiä ei näytetä testin suorittajalle.

Taulukko 17 FEP Wizard, virheiden vakavuusasteet

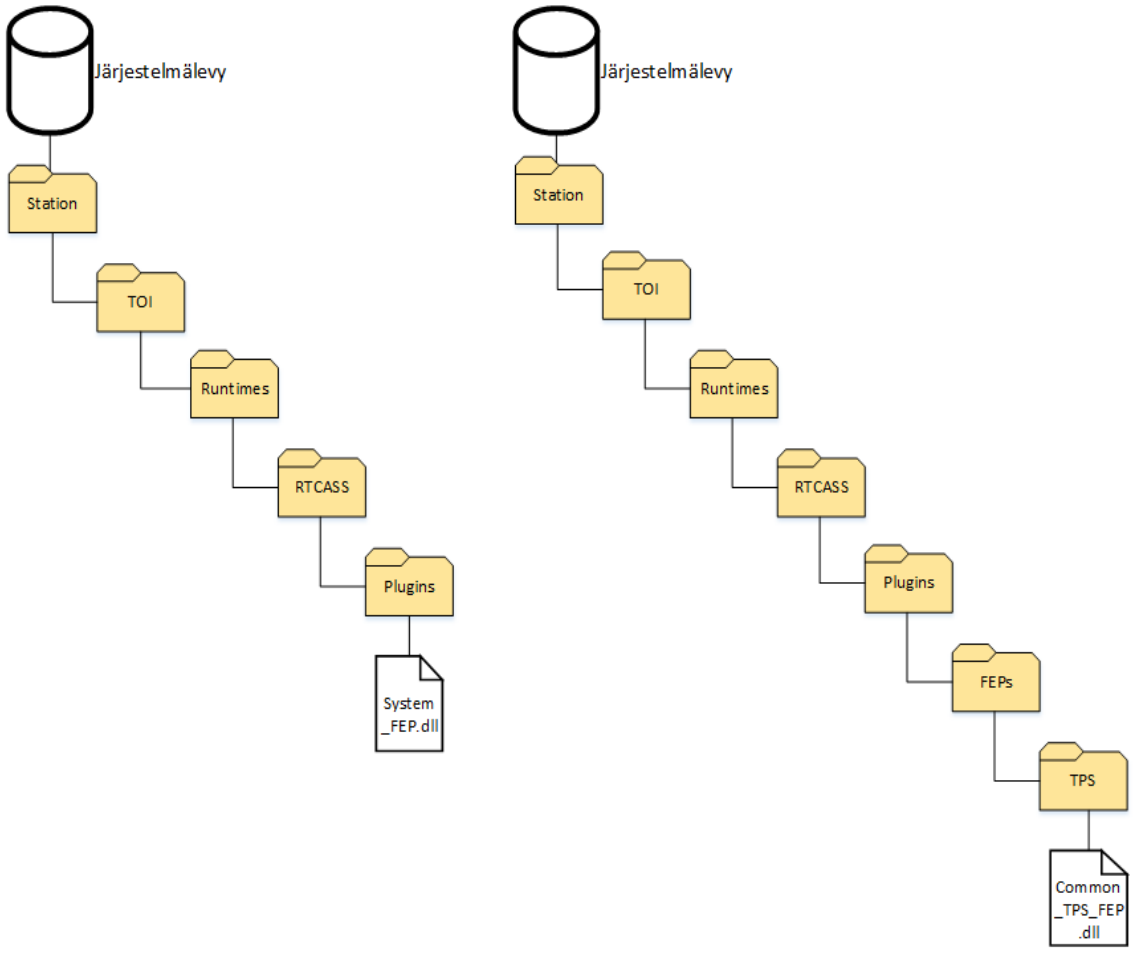
Virheiden vakavuus	Määritelty arvo
`protected const int M_FEP_INTERNAL_ERROR`	`-3000`
`protected const int M_FEP_INTERNAL_WARNING`	`-2000`
`protected const int M_FEP_INTERNAL_ADVISE`	`-1000`
`protected const int M_FEP_ADVISE`	`1000`
`protected const int M_FEP_WARNING`	`2000`
`protected const int M_FEP_ERROR`	`3000`
`protected const int M_FEP_CRITICAL_SHUTDOWN`	`4000`

Ohjelman 3 Riveillä 16-23 käsitellään testiohjelmalle välitettäviä heksadesimaalilukuja. Tässä esimerkissä suurin sallittu summa on 16-bittisen luvun positiivinen maksimi eli 65535. Mikäli yhteenlaskussa saatava summa on asetettua raja-arvoa suurempi, asetetaan muuttuja *bPARAM_4* todeksi ylivuodon merkiksi. Parametri *bPARAM_4* sekä kaikki muut käytettävät parametrit ovat FEP Wizardin nimeämiä eikä käyttäjä voi vaikuttaa tähän nimeämiskäytäntöön. Kun ATLASilla toteutetulle testiohjelmalle välitetään tieto ylivuodosta, osataan testiohjelmassa toimia oikein ja vältytään virheellisen tiedon käyttämiseltä. Vaikka riveillä 41 ja 47 FEPin suorittaminen lopetetaan komennolla *return null*, voisi lopetuksessa palautettava arvo olla mitä tahansa muutakin. Ohjelman paluuarvoa ei tarkastella FEPin suorituksen loputtua.

Kun muutokset FEPin lähdekoodiin on saatu tehtyä, tulee se kääntää Visual Studiossa ja luoda ATLASilla toteutetussa testiohjelmassa käytettävä dll-tiedosto. Visual Studio luo FEPistä käyttäjän valinnan mukaan joko Debug- tai Release-version. Luotu dll-tiedosto on nimetty sen luokan mukaan, jonka käyttäjä FEP Wizardissa loi. Esimerkin tapauksessa luotu dll-tiedosto on nimeltään *SampleFEP.dll*. Käännöksen yhteydessä samaan kansioon FEPin dll-tiedoston kanssa kopioidaan myös tiedosto *DirectTest.Interfaces.dll*.

DirectTest.Interfaces.dll-tiedosto sisältää FEPin ja ATLASilla toteutetun testiohjelman välisen rajapinnan käsittelyssä tarvittavat resurssit. Sama dll-tiedosto löytyy myös Station FEPeille tarkoitetusta kansiosta RTCASS-aseman järjestelmässä ja se on ladattuna käyttöön aseman käynnistyessä, joten tiedostoa ei tarvitse siirtää RTCASS-asemalle uudelleen.

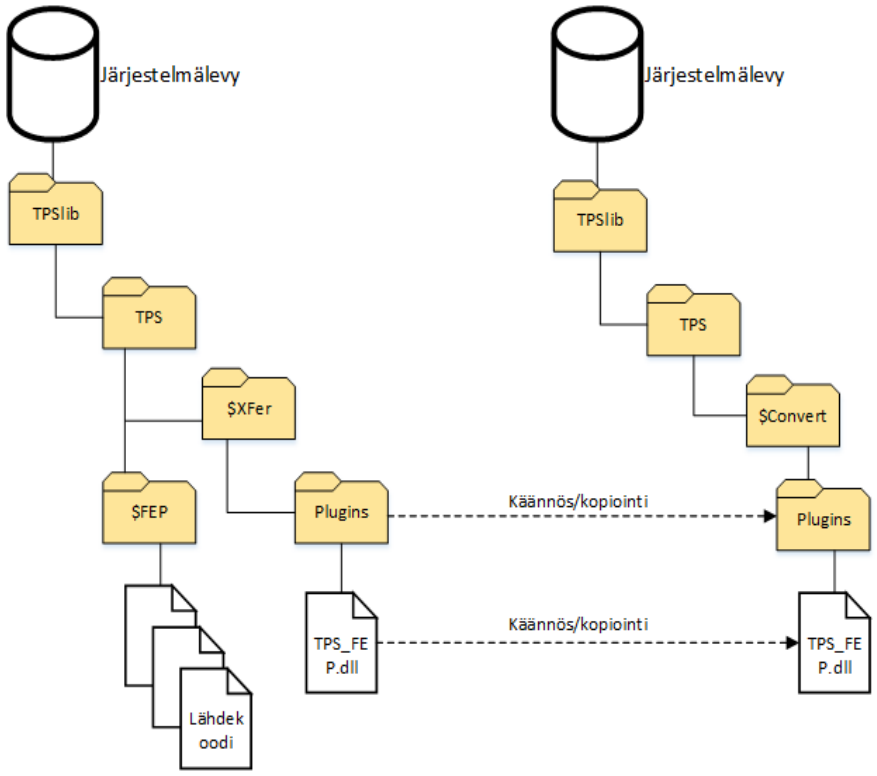
Luodun FEPin sijainti RTCASS-asemalla valitaan sen perusteella, onko kyseinen FEP tehty vain yhdelle vai usealle TPS:lle. Kuvassa 10 on esitettynä luodun tiedoston sijainti RTCASS-asemalla siinä tapauksessa, mikäli FEP on System FEP tai Common deployed TPS FEP.



Kuva 10 System FEPin kansiorakenne RTCASS-asemalla

Kuvassa 11 nähdään FEPin sijainti siinä tapauksessa, mikäli se on määritelty käytettäväksi vain yhdessä TPS:ssä, eli FEPin tyyppi on TPS deployed FEP. Kuvasta 11 nähdään myös ATLAS-lähdekoodille käännöksen aikana tapahtuvat toimenpiteet RTCASS-asemalla. Kuvassa 11 *TPS* on se yksilöllisellä tunnisteella nimetty kansio, jossa sijaitsee ATLASilla toteutettu testiohjelma. Tämän kansion alle luodaan kansio *$Xfer*, jonka alle kansio *Plugins*. Kansio *Plugins* on se kansio johon testiohjelman FEP tullaan sijoittamaan. Kansio *$FEP* on varattu FEPin lähdekoodille. FEPin lähdekoodia ei kuitenkaan TPS:n suorituksen aikana tarvita ja kansio on varattu ainoastaan sitä varten, että koko

TPS voidaan toimittaa yhtenä asennuspakettina sijainnista toiseen ja kehitystä voidaan jatkaa myös uudessa sijainnissa.



Kuva 11 TPS FEPin kansiorakenne RTCASS-asemalla käännöksen jälkeen

Kun ATLASilla toteutettu testiohjelma käännetään RTCASS-asemalla, niin samalla luodaan kuvassa 11 näkyvä kansio *$Convert*. Kansion *$Convert* alle kopioituu kansion *$Xfer* alla olevat kansiot sekä tiedostot. Samalla kun kansioiden sisältöä kopioidaan ja ATLAS-koodia käännetään, niin kääntäjä tarkistaa, että ATLAS-koodissa määritelty FEP on saatavilla ja että testiohjelma voi löytää sen. Myös käännetty ATLASilla toteutettu testiohjelma kopioituu kansioon *$Convert*.

Ohjelma 4 kuvaa ATLAS-kielessä tarpeelliset komennot FEPin käyttämiseksi. Esimerkin rivinumerointi on toteutettu ATLASissa käytettävällä rivinumerointimenetelmällä. Rivillä *000100* FEP liitetään osaksi käännettävää ATLAS-testiohjelmaa. Riveillä *000500* ja *000510* määritellään ATLAS-testiohjelmassa FEPin ulkopuolella käytettävät muuttujat, jotka välitetään FEPille. Rivien 005000 ja 005100 välinen alue on FEPin rajapinnan sekä FEPin sisäisten muuttujien määrittelevä alue. Rivillä 100100 on esimerkki siitä, miten FEPiä kutsutaan ATLAS-testiohjelmassa. Esimerkki ei esittele eri arvojen tallentamista FEPille välitettäviin muuttujiin, eikä FEPin suorituksen jälkeisiä toimenpiteitä eikä muuttujien tarkastamista. Ensimmäisessä sarakkeessa oleva *C*-kirjain on ATLAS-kielessä merkkinä kommenttirivistä.

```
C         Include FEP to the ATLAS program $
000100    INCLUDE, NON-ATLAS MODULE 'SAMPLEFEP' $

C         Declare needed parameters $
000500    DECLARE, DIGITAL, STORE, 'INPUT-HEX-1', 'INPUT-HEX-2',
          RESULT-HEX', FORMAT BNR 16 BITS $
000510    DECLARE, BOOLEAN, STORE, 'OVERFLOW', 'EXCEPTION' $

C         Declare the interface of the FEP $
005000    DEFINE, 'SAMPLEFEP', EXTERNAL, PROCEDURE ('FIRST-HEX',
          'SECOND-HEX') RESULT ('RESULT-HEX', 'OVERFLOW',
          'EXCEPTION') $
005010       DECLARE, DIGITAL, STORE, 'FIRST-HEX', FORMAT BNR 16
             BITS $
005020       DECLARE, DIGITAL, STORE, 'SECOND-HEX', FORMAT
             BNR 16 BITS $
005030       DECLARE, DIGITAL, STORE, 'RESULT-HEX', FORMAT
             BNR 16 BITS $
005040       DECLARE, BOOLEAN, STORE, 'OVERFLOW' $
005050       DECLARE, BOOLEAN, STORE, 'EXCEPTION' $
005100    END, 'SAMPLEFEP' $

C         Call the FEP from ATLAS program $
100100    PEFORM, 'SAMPLEFEP', 'INPUT-HEX-1', 'INPUT-HEX-2',
          'RESULT-HEX', 'EXCEPTION' $
```

Ohjelma 4. *FEPin käyttöönotto ATLAS-testiohjelmassa*

FEPiä käyttävä ATLASilla toteutettu testiohjelma siirretään RTCASS-asemalle yhdessä luodun *SampleFEP.dll*-tiedoston kanssa. Tiedostot siirretään niille varattuihin hakemistoihin, kuten kuvassa 11 on esitetty. Tämän jälkeen testiohjelma käännetään RTCASS-aseman käännöstyökalulla. Kääntämisen yhteydessä saadaan RTCASS-aeman käännöstyökalulta varoitus:

```
[CASS TO RTCASS]: TPS must supply external plugin for capability
FEP__SampleFEP
```

FEPin valmistamiseksi RTCASS-asemalle on olemassa oma ohjeistusdokumentaatio, jossa määritellään, että FEPin nimi tulee olla *FEP__* lisättynä luodun aliohjelman nimen eteen [6]. ATLAS-kielessä FEPiä kutsuttaessa tätä etuliitettä ei käytetä, mutta luodussa tiedostossa se kuuluisi olla. Kun etuliite lisätään luotuun dll-tiedostoon ja suoritetaan testiohjelman käännös uudelleen RTCASS-asemalla, niin varoitus pysyy samana. Tiedostot kopioituvat molemmissa tilanteissa kansioon *$Convert*. FEPin kopioiminen kansioon, jossa sitä voitaisiin käyttää common TPS deployed FEPinä eli useissa testiohjelmissa, aiheuttaa edelleen saman varoituksen riippumatta siitä, käytetäänkö tiedostossa etuliitettä vai ei.

Käännöksestä saadun virheilmoituksen johdosta oltiin yhteydessä henkilöihin, jotka kouluttavat FEPin valmistamisessa käytettävien sovellusten käyttöä. Ensimmäinen ehdotettu korjaustoimenpide oli yrittää kääntää FEP Visual Studiossa Debug-tilassa. FEP

käännettiin Visual Studiossa uudelleen Debug-tilassa ja siirrettiin tämän jälkeen RTCASS-asemalle, mutta virheilmoitus säilyi edelleen samana.

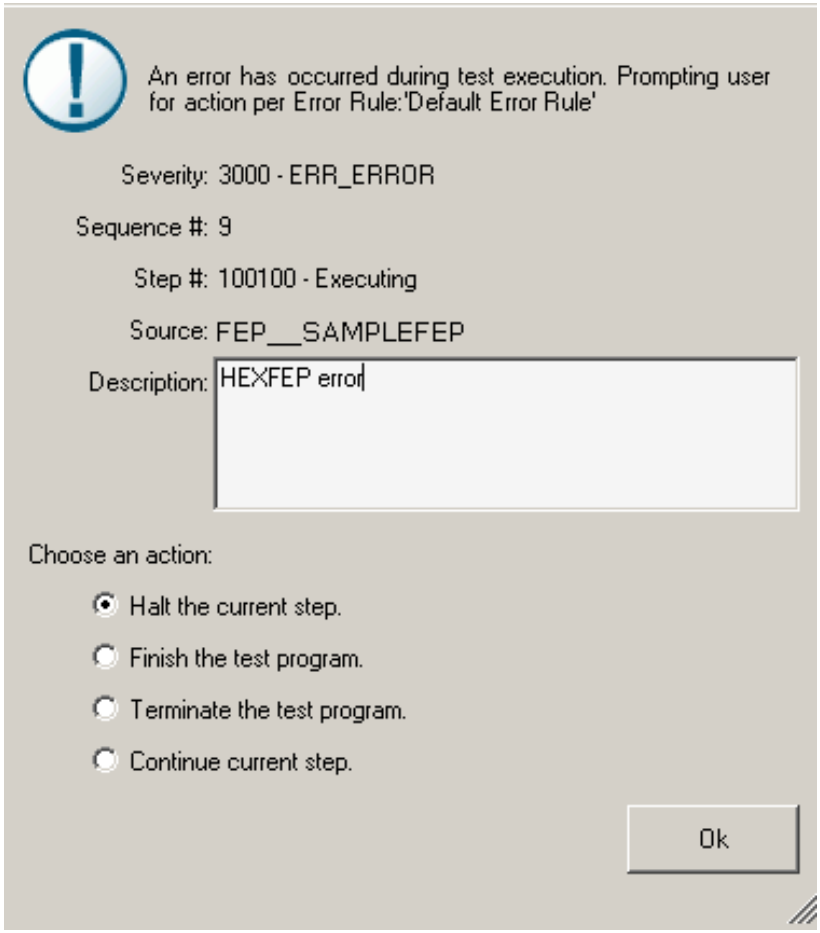
Koska varoitukseen ei löydetty ratkaisua käyttämällä Debug-tilaa, lähetettiin osa lähdekoodista FEP-työkalujen kouluttamisesta vastaavista henkilöille. Lähdekoodista saatiin selville FEP Wizardin luomassa C#-projektitiedostossa oleva virhe. C#-projektitiedostossa määritellään luotavan dll-tiedoston nimeäminen ja siellä pitäisi määritellä *FEP__*-etuliitteen lisääminen luotavaan dll-tiedostoon. Kun C#-projektitiedostoon tehdään muutos, joka lisää halutun etuliitteen automaattisesti, saatiin *FEP__*-etuliite dll-tiedostoon jo käännöksen yhteydessä. Tämän jälkeen ATLAS-testiohjelmaa käännettäessä saatiin edelleen sama varoitus kuin aikaisemmin. Jälleen oltaessa yhteydessä FEP-työkalujen kouluttajiin, he kertoivat tämän varoituksen kuuluvan asiaan ja olevan ilmoitus siitä, että käyttäjän tulee olla tietoinen FEPin tarpeellisuudesta testiohjelman suorituksessa.

Kun käännettyä ATLASilla toteutettua testiohjelmaa yritetään suorittaa saadusta virheilmoituksesta huolimatta, niin testiohjelman lataaminen suoritettavaksi RTCASS-aseman työkaluilla onnistuu ja myös itse testiohjelman suorittaminen onnistuu aivan kuten on suunniteltu. Mikäli ATLASilla toteutettua testiohjelmaa yritetään suorittaa varoituksesta huolimatta siten, että FEP ei ole saatavilla, keskeytyy testiohjelman lataus ennen kuin sitä pystytään suorittamaan.

ATLASilla toteutetun testiohjelman sekä FEPin testauksen yksityiskohtia ei tässä tutkielmassa käsitellä tarkasti. Testiohjelmaa kuitenkin testattiin siinä määrin, että sen oikeasta toiminnasta voitiin olla varmoja esimerkiksi ylivuotojen tapauksessa. Niin sanotun perustoiminnan varmistamisen jälkeen, testattiin FEPin toimintaa myös poikkeuksien tapauksessa. Poikkeuksen toimintaa voidaan testata lisäämällä FEPissä olevan *try*-koodilohkon sisään poikkeuksen aiheuttava käsky:

```
throw new System.ArgumentException("poikkeus");
```

Poikkeuksen toiminnan testaamisella varmistetaan se, että ATLASilla toteutettu testiohjelma saa ilmoituksen tapahtuneesta virhetilanteesta. Kuvassa 12 on nähtävissä käyttäjälle annettava ilmoitus poikkeuksen tapahtuessa. Aikaisemmin määriteltyjen poikkeuksen tietojen lisäksi ilmoituksesta nähdään myös koodirivi jolla poikkeus on tapahtunut. Tämä helpottaa vikatilanteen paikantamista, mikäli sama poikkeus voi aiheutua useissa eri kohdissa testiohjelmaa.



Kuva 12 FEP Wizard, poikkeusnäkymä

Käyttäjä voi poikkeuksen tapahtuessa valita, miten haluaa testiohjelman suoritusta jatkaa. Kuvassa 12 on esitettynä eri vaihtoehdot testiohjelman suorituksen jatkamiselle poikkeuksen tapahduttua. Oletuksena annettava vaihtoehto *Halt* ainoastaan pitää testiohjelman samassa kohdassa, eikä jatka testiohjelman suoritusta. Vaihtoehto *Finish* lopettaa testiohjelman suorituksen, mutta pitää testiohjelman ladattuna RTCASS-asemalla ja mahdollistaa sen suorittamisen välittömästi uudelleen. Vaihtoehto *Terminate* lopettaa ohjelman suorittamisen sekä poistaa testiohjelman suorituksesta. Vaihtoehdon *Terminate* jälkeen testiohjelma tulee ladata RTCASS-aseman työkaluilla uudelleen, mikäli se halutaan suorittaa uudelleen. Vaihtoehdot *Finish* ja *Terminate* asettavat RTCASS-aseman laitteet alkutilaan ja poistavat esimerkiksi kaikki jännitteet testattavista laitteista. Viimeinen vaihtoehto *Continue* jatkaa ohjelman suoritusta, aivan kuin FEP olisi saatu suoritettua onnistuneesti. Viimeisen vaihtoehdon valitseminen ei ole suotavaa, sillä testiohjelmassa käytetyt muuttujat saattavat aiheuttaa virheitä, joista ATLASilla toteutettu ohjelma ei osaa toipua. Pahimmassa tapauksessa testattava laitteisto tai RTCASS-aseman laitteisto saattaa rikkoutua tämän vaihtoehdon valinnasta johtuen.

Poikkeuksen aiheuttava koodirivi poistetaan lopullisesta lähdekoodista, sillä sen tarkoituksena oli ainoastaan tutkia poikkeuksen aikana tapahtuvaa toimintaa. Tämän jälkeen FEP voidaan todeta olevan valmis käytettäväksi. Tämä FEP sekä sen luomiseksi suoritetut toimenpiteet toimivat tämän jälkeen pohjana tuleville FEPeille. Seuraava toimenpide on arvioida, mitä ominaisuuksia tulevilla FEPeillä halutaan toteuttaa missäkin testiohjelmissa.

5. TULOKSET

Tässä luvussa käsitellään tutkielman aikana saavutettua hyötyä eri CASS-asemille sekä saavutettuja tuloksia. Tulosten käsittely on eritelty erikseen MFCASS-asemalle sekä RTCASS-asemalle. Luvussa käsitellään myös ehdotukset mahdollisiksi jatkotoimenpiteiksi.

5.1 Tulokset MFCASS-asemalle

MFCASS-aseman tapauksessa, valmista FEPiä ei saatu luotua, vaikka se oli yksi suurimmista tavoitteista tässä tutkielmassa. FEPin luomisessa tarvittavat menetelmät saatiin kuitenkin selvitettyä ja näitä menetelmiä voidaan tulevaisuudessa käyttää useilla eri ohjelmointikielillä toteutettavien FEPien pohjana. Fortran-ohjelmointikielen tuntemus lisääntyi työn aikana siinä määrin, että sen avulla voidaan muodostaa yksinkertaista, mutta toimivaa ohjelmakoodia. Fortranin kohdalla ongelmia aiheuttivat kielen syntaksisäännöistä johtuvat virheilmoitukset, joihin tästä eteenpäin tiedetään ratkaisut.

FEPin toteutuksen mahdollistamiseksi tärkein toimenpide on saada selvyys testiohjelman rajapinnassa käytettävän kirjaston *SCFE.OLB* saatavuuteen. Mikäli kyseinen kirjasto saadaan käyttöön, voidaan C- ja Fortran-kielellä toteutettujen FEPien toteuttamista yrittää uudelleen.

OpenVMS-käyttöjärjestelmän tuntemus lisääntyi työn aikana huomattavasti. Käyttöjärjestelmän tunnettavuus auttaa tulevaisuudessa testiohjelmien kehittäjiä kaikissa käyttöjärjestelmään liittyvissä ongelmissa. Emulaattorissa suoritettavaan OpenVMS-käyttöjärjestelmään voidaan tämän jälkeen yrittää toteuttaa esimerkiksi ATLAS-koodin kääntämisessä sekä MTPSI-tiedostojen luomisessa käytettäviä työkaluja.

Mikäli laitetason ALPHA-toteutus päätetään joskus korvata emulaattorilla tai järjestelmän rinnalle halutaan toinen järjestelmä saatavuuden turvaamiseksi, on emulaattorille asetettavat vaatimukset saatu valmiiksi selvitettyä. Saatujen vaatimusten pohjalta osataan vaatia oikeita resursseja ja ominaisuuksia valittavalta emulaattorilta. Koska ALPHA-arkkitehtuurin saatavuus ja tuki eivät tulevaisuudessa tule luultavasti kasvamaan, on hyvä, että on olemassa valmiit lähtökohdat ALPHA-arkkitehtuurin korvaamiseksi muilla menetelmillä.

ATLAS-kielellä toteutettavien testiohjelmien tuntemus kasvoi tutkielman teon aikana hieman. Mahdollisuus useiden eri MTPSI-tiedostojen käyttöön sekä TPS:n suorittami-

nen näin ollen eri konfiguraatioilla tuli kirjoittajalle uutena asiana esille. Tätä tietoa voidaan hyödyntää suunniteltaessa tulevien TPS:ien konfiguraatioita.

5.2 Tulokset RTCASS-asemalle

RTCASS-asemalle saatiin toteutettua toimiva FEP, vaikka se ei ollutkaan toteutettu C-kielellä, kuten alun perin oli ajateltu. Samalla saatiin kokemusta käytettävästä FEPin luomiseen tehdystä työkalusta, FEP Wizardista. Oletus FEPin luomisen onnistumisesta RTCASS-asemalle huomattavasti helpommin kuin MFCASS-asemalle osoittautui oikeaksi. Tutkielman suorituksen aikana saatiin selvitettyä ongelmia, jotka olisi kohdattu myöhemmin, mikäli FEPien käyttöä päätetään yleistää ja luomista lisätä. Tämän tutkielman jälkeen FEPin luomisen valmius sekä tarvittava ohjeistus on olemassa RTCASS-asemalle.

Yleisesti molempia CASS-asemia koskevaan TPS:n siirtomahdollisuuteen ympäristöstä toiseen osataan tämän tutkielman jälkeen varautua paremmin, sillä tiedetään TPS:n mahdollinen riippuvuus FEPin saatavuudesta. Myös uusien ATLASilla toteutettavien testiohjelmien kohdalla osataan varautua asemien väliseen eroavaisuuteen, mikäli testiohjelmassa päätetään käyttää FEPiä.

Insta ILS:ssä jossa työ tehtiin, on myös tärkeää tietää, mikä on sen kyky ja valmius näiden ohjelmistojen ja testiohjelmien tuottamisessa. Näin ollen osataan tehdä suunnitelmia ja varautua ennalta sellaisiin tilanteisiin, joissa testiohjelman tuottaminen ei olisi mahdollista ilman FEPin toteuttamista.

6. YHTEENVETO

Tutkielman tarkoituksena oli luoda FEP eli ATLAS-ohjelmointikielen laajennus sekä MFCASS- että RTCASS-asemalle. FEPien käyttö on Insta ILS:ssä tehtyjen testiohjelmien pohjalta tuttua, mutta itse näitä testiohjelmien laajennuksia ei ole vielä toteutettu. ATLAS-kielen laajennusten luominen helpottaisi testiohjelmien valmistamista nykyisin CASS-asemilla testattaville laitteille. FEPin avulla ATLASilla toteutettuun testiohjelmaan voidaan sisällyttää jollakin toisella ohjelmointikielellä toteutettu osio, kuten datan käsittely. Nykyaikaiset testattavat laitteistot hyödyntävät yhä enemmän eri tietoliikenneväyliä ja tuottavat suuria määriä dataa. ATLAS-kielen ominaisuudet soveltuvat huonosti lisääntyneen datamäärän analysointiin ja jollakin toisella ohjelmointikielellä luotavat sovelluslaajennukset helpottaisivat tätä tilannetta huomattavasti.

Tutkielman tavoitteena oli luoda toteutettavat laajennukset ainakin osittain samalla ohjelmointikielellä molemmille CASS-asemille, jotta uudelleenkirjoitettavan koodin määrä olisi mahdollisimman pieni. C-kieli on ainoa molempien CASS-asemien tukema ohjelmointikieli. FEPin toteuttaminen samalla ohjelmointikielellä molempiin CASS-asemiin helpottaisi myös ATLASilla toteutettujen testiohjelmien siirtoa CASS-asemien välillä. Jotta testiohjelma voidaan suorittaa molemmissa CASS-asemissa, tulee tehdyn laajennuksen olla saatavilla molemmille CASS-asematyypeille.

Tutkielman tarkoituksena oli myös selvittää MFCASS-aseman virtuaalisen vaihtoehdon toteutusmahdollisuuksia. Virtuaalisella ympäristöllä voitaisiin jopa korvata MFCASS-aseman tarjoamat ohjelmointityökalut, mikäli ympäristö saadaan toteutettua riittävän kattavasti. MFCASS-aseman tärkeys tällä hetkellä ATLASilla tuotettujen testiohjelmien luomiseksi on todella suuri. Virtuaalinen ympäristö tulee tutkielman teon aikana toimimaan myös FEPin toteutusympäristönä MFCASS-asemalle.

Tuotekehitys ja uusien TPS:ien tuottaminen on hyvin hankalaa pelkällä RTCASS-asemalla, joten tuotekehityksessä ollaan osittain riippuvaisia MFCASS-asemista. Riippuvuus MFCASS-asemasta johtuu siitä, että laitteistotietoa sisältävän MTPSI-tiedoston luominen ilman MFCASS-asemaa on mahdotonta. Tämän lisäksi RTCASS-asemalla ATLAS-koodin kääntämisessä syntyvien virheilmoitusten tulkitseminen on hyvin hankalaa, sillä virheilmoitus saattaa pahimmassa tapauksessa olla vain tieto virheiden lukumäärästä. MFCASS-asemalla ATLAS-koodin kääntämisestä saatavat virheilmoitukset ilmoittavat esimerkiksi rivin, jolla virhe ilmenee, joten virheiden korjaaminen nopeutuu huomattavasti. Näistä syistä johtuen MFCASS-asemalle vaihtoehtoisen ympäristön tutkiminen on tärkeää ja osa tätä tutkielmaa.

Tutkielman teko aloitettiin selvittämällä virtuaalisen ympäristön luontimahdollisuudet ja edellytykset halutulle ympäristölle. Kokeilemalla muutamia eri emulaattoreita, saatiin virtuaalisen ympäristön vaatimukset selvitettyä ja MFCASS-asemassa käytettävä OpenVMS-käyttöjärjestelmä emulaattorissa suoritettavaksi. Oikean emulaattorin valinta on äärimmäisen tärkeää, sillä virtuaalisen ympäristön luominen esimerkiksi sellaiseen järjestelmään, josta puuttuu verkkoyhteys, on täysin turhaa ja FEPin luomisen kannalta hyödytöntä. Tutkielman teon aikana saatiin selville ne vaatimukset mitkä emulaattorille tulee asettaa, jotta sen käyttö FEPin luomisessa on mahdollista.

Tutkielmassa luotuun virtuaaliseen ympäristöön saatiin asennettua C- ja Fortran-kielen kääntäjät. C- sekä Fortran-kielellä toteutetut esimerkkiohjelmat saatiin myös käännettyä virtuaalisessa ympäristössä ja tämän jälkeen siirrettyä MFCASS-asemaan onnistuneesti. Virtuaalisessa ympäristössä luotujen C- ja Fortran-kielellä toteutettujen testiohjelmien suoritus onnistui myös todellisessa ympäristössä, eli MFCASS-asemalla.

FEPin luomista MFCASS-asemalle ei tässä tutkielmassa saatu lopulta toteutettua. FE-Pin luomisen yhteydessä havaittiin, että käytetystä järjestelmästä puuttuu FEPin sekä ATLASilla toteutetun testiohjelman rajapinnan muodostava kirjasto. Puuttuvasta kirjastosta johtuen ei FEPin luomista voitu jatkaa tästä eteenpäin.

Tutkielmassa saatiin kuitenkin selvitettyä Fortranilla toteutettujen esimerkkikoodien perusteella FEPin rakenne sekä toimintaperiaate MFCASS-asemalla. Toimintaperiaatteen perusteella voitiin todeta, että MFCASS-aseman FEPissä käytetyn ohjelmarakenteen käyttäminen myös RTCASS-asemalle toteutettavassa FEPissä olisi huomattavan hankalaa verrattuna RTCASS-aseman apuohjelmalla tuotetun FEPin valmistamiseen. MFCASS-asemalla FEPin lähdekoodissa käytetään myös useita käyttöjärjestelmästä riippuvaisia komentoja, joiden käyttö RTCASS-asemalla ei ole mahdollista. Havaituista eroista johtuen alkuperäinen tavoite valmistaa FEP samalla ohjelmointikielellä molemmille CASS-asemille ei enää ollut järkevä siirryttäessä tutkielmassa RTCASS-aseman FEPin toteutukseen.

Tutkielman teon aikana Fortran-kielen tuntemusta saatiin parannettua ja FEPin luomista varten olevien esimerkkikoodien virheitä korjattua. Fortran-kielen tuntemuksen parantaminen ja esimerkkikoodien korjaamisen perusteella voidaan todeta, että mikäli FEP päätetään MFCASS-asemalle vielä toteuttaa, niin toteutuskielenä tulee luultavasti olemaan Fortran. Tämä siitäkin huolimatta, että kielen tuntemus toiseen toteutusvaihtoehtoon eli C-kieleen, on hieman heikompi.

RTCASS-asemalle FEPin toteuttaminen voidaan tehdä käyttämällä hyväksi FEP Wizard -apuohjelmisto. Apuohjelmistoa käyttämällä RTCASS-asemalle saatiin luotua toimiva FEP. FEPin luomisessa FEP Wizardin avulla havaittiin, että FEP Wizard tuottaa oletusasetuksilla väärin muotoillun C#-projektitiedoston. Projektitiedostossa olevan virheen korjauksen jälkeen havaittiin myös testiohjelman antama varoitus käyttäjälle. Tes-

tiohjelman antaman virheilmoituksen syy oli täysin dokumentoimaton. Käyttäjälle annettavan varoituksen syyksi paljastui ainoastaan ilmoitus siitä, että suoritettava testiohjelma vaatii FEPin käyttöä ja käyttäjän tulee varmistua FEPin oikeasta sijainnista järjestelmässä.

Vaikka FEPiä ei saatu luotua molemmille CASS-asemille, voidaan tutkielman onnistumiseen olla tyytyväisiä. Tällä hetkellä on hyvin tiedossa se, mitä pitää selvittää, jotta FEPin luominen MFCASS-asemalla on mahdollista. Lisäksi FEPin esimerkkikoodit ovat valmiina käännöskokeilua varten ja syntaksiltaan oikeiksi korjattu.

Emulaattorin käyttö MFCASS-aseman käyttämän ALPHA-arkkitehtuurin korvaajana selkeytyi ja vaatimukset emulaattorille saatiin selvitettyä. Tämän jälkeen mahdollisten emulaattorien vertailua voidaan suorittaa kattavasti jo ennen kaupallisia hankintoja. Emulaattorin käyttöä ATLASilla toteutetun koodin kääntämiseen sekä MTPSI-tiedostojen luomiseen tulee myös tutkia lisää.

Insta ILS Oy johon tutkielma tehtiin, sai selkeyttä omista kyvyistään testiohjelmille tuotettavien laajennusten luomisessa. Omien kykyjen tunnistaminen auttaa varautumaan tulevaisuudessa testiohjelmia suunniteltaessa. Yritys sai myös lisää tietoa käytettävistä järjestelmistä ja niiden ominaisuuksista.

Saatavilla olevien esimerkkikoodien rakennetta olisi voitu selvittää jo ennen kuin virtuaalista ympäristöä alettiin pystyttää. Tällä tavalla olisi saatettu päästä selvyyteen puuttuvasta kirjastosta jo ennen emulaattorin käyttöönottoa. Johtuen Fortran-kielen tuntemattomuudesta kirjoittajalle näin ei kuitenkaan tehty. Virtuaalisen ympäristön merkitys on myös siinä määrin tärkeä osa tutkielmaa, että sen käyttömahdollisuuksien selvitys oli joka tapauksessa todella hyödyllistä ja kannatti tehdä. Tutkielman teon aloittaminen MFCASS-aseman puolelta osoittautui oikeaksi valinnaksi, sillä FEPin toteutusta C-kielellä ei näin ollen yritetty RTCASS-asemalle. FEPin toteutus C-kielellä RTCASS-asemalle olisi mahdollista, mutta huomattavasti hitaampaa kuin valmiin apuohjelmiston käyttäminen.

LÄHTEET

[1] NAVY ATE and TPS Acquisition handbook, COMNAVAIRSYSCOM PMA260, 31 March 2005

[2] The CASS Family OTPS Sustainment Process, Version 1.1, September 2009

[3] DoD Automatic Test Systems Master Plan, Department of Defense, 2012

[4] User's Guide for TPS Developers (a.k.a. T00K), version 12, 30 August 2012

[5] Test Program Sets Performance Specification, Department of Defense, United States of America, MIL-PRF-32070A, 10 January 2012.

[6] RTCASS FEP Developer's Guide, The Boeing Company, St. Louis, Missouri, version 1.1.0, 12.06.2009

[7] Standard Test Language for All Systems – Common/Abbreviated Test Language for All Systems (C/ATLAS), 716 C/ATLAS Subcommittee of IEEE Standards Coordinating Committee 20, 03.03.1995, IEEE Standards Board

[8] Master Test Program Set Index, Core Topics

[9] Programming Language FORTRAN, American National Standard, ANSI® X3.9-1978, Revision of ANSI X3.9-1966, Saatavana(viitattu 30.1.2016): http://www.fortran.com/fortran/F77_std/rjcnf.html

[10] Standard Test Language for All Systems – Common/Abbreaviated Test Language for All Systems (C/ATLAS), The Institute of Electrical and Electronics Engineers, Inc, March 16 1995

[11] RTCASS Development Tools User Guide, The Boeing Company, St. Louis, Missouri, March 2 2015

[12] G. Bell, W.D. Strecker, What Have We Learned from the PDF'-11 - What We Have Learned from VAX and Alpha

[13] B. DuCharme, The Operating System Handbook or, fake your way through minis and mainframes, 2001

[14] Hewlett Packard Enterprise, OpenVMS Systems, Saatavissa (viitattu 31.1.2016): http://www8.hp.com/us/en/products/servers/openvms/what.html#!&pd1=1&pd2=1

[15] P. C. Anagnostopoulos, S. Hoffman, Writing programs in DCL, second edition, 1999

[16] Intermediate Level Test Program Sets (TPS) / Operational Test Program Sets (OTPS), General Acceptance Procedures, 12 Nov 2012

[17] RTCASS Development Tools Training Part II, Navair, In-Service Support Center, Jacksonville, 1/30/2016

[18] J.F. Curose, K.W. Ross, Computer Networking A top-down approach, Pearson international edition, 4th edition, March 23 2007

[19] OpenVMS License Management Utility Manual, Compaq Computer Corporation, Houston, Texas, April 2001

[20] HP TCP/IP Services for OpenVMS, Users guide, Hewlett-Packard, Palo Alto, California, September 2003

[21] HP OpenVMS System Services Reference Manual: GETUC-Z, Hewlett-Packard Company, Palo Alto, California

[22] OpenVMS Linker Utility Manual, Compac Computer Corporation, Houston, Texas, April 2001

[23] OpenVMS System Messages and Recovery Procedures Reference Manual: M-Z, Digital Equipment Corporation, Maynard, Massachusetts, May 1993

LIITE 1 KÄÄNNÖSYMPÄRISTÖN LUOMINEN MFCASS-ASEMALLE

VERKKOASETUKSET

Kun isäntäjärjestelmästä löytyy verkkoliityntä *eth0*, niin emulaattorin verkkoliitynnän luominen saadaan tehtyä syöttämällä Linux-järjestelmän komentoriville seuraavat komennot:

```
sudo tunctl -t tap0 -u <käyttäjätunnus>
sudo ifconfig tap0 up
sudo brctl addbr br0
sudo brctl addif br0 eth0
sudo brctl setfd br0 0
sudo ifconfig eth0 0.0.0.0
sudo ifconfig br0 192.168.1.2 netmask 255.255.255.0 broadcast
192.168.1.255 up
sudo route add -net 0.0.0.0/0 gw 192.168.1.1
sudo brctl addif br0 tap0
sudo ifconfig tap0 0.0.0.0
```

Taulukossa 18 on selitettynä isäntäjärjestelmän komentoriville syötettyjen komentojen suorittamat toimenpiteet.

Taulukko 18 Linux verkkoasetukset

Rivi nro.	Toiminto
1.	Tunctl:lle annetuilla parametreilla määritellään virtuaaliselle verkkoliitännälle annettava nimi *tap0* sekä käyttäjä jolla on oikeus käyttää tätä liityntää.
2.	Aktivoidaan parametrina annettu verkkoliitäntä *tap0*.
3.	Luodaan Ethernet-silta *br0*.
4.	Asetetaan Ethernet-silta *br0*:lle portti *eth0*.
5.	Asetetaan *br0*:n viive (engl. bridge forward delay) arvoon 0.
6.	Asetetaan verkkoliitäntä eth0:n IP-osoitteeksi 0.0.0.0.
7.	Asetetaan Ethernet-silta br0:n IP-osoite 192.168.1.2, verkkomaski 255.255.255.0, broadcast-osoite 192.168.1.255 sekä aktivoidaan verkkoliityntä.
8.	Lisätään reititys verkkoon 0.0.0.0/0 yhdyskäytävän 192.168.1.1 kautta.
9.	Luodaan Ethernet-silta *br0* annetulle verkkoliitäntä *tap0*:lle.
10.	Asetetaan verkkoliityntä *tap0*:n IP-osoite 0.0.0.0

Näiden asetusten jälkeen verkkoliityntä *tap0* voidaan antaa kokonaan emulaattorissa suoritettavan järjestelmän käyttöön. Isäntäjärjestelmässä tulee tämän jälkeen asettaa verkkoliitynnälle *eth0* IP-osoite, joka kuuluu samaan aliverkkoon osoitteen 192.168.1.1/255.255.255.0 kanssa. Asetusten syöttämisen jälkeen verkkoyhteys voidaan muodostaa suoraan isäntäjärjestelmän ja emulaattorin välille.

OPENVMS:N HANKINTA

OpenVMS:n harrastelijalisenssin saamiseksi tulee rekisteröityä DECUServe-yhteisön jäseneksi. DECUServen jäseneksi liittyminen tapahtuu ottamalla SSH-yhteys osoitteeseen:

eisner.encompasserve.org (viitattu 31.7.2015)

Käyttäjä antaa palvelussa tarvittavat tiedot itsestään ja edustamastaan tahosta. Tämän jälkeen käyttäjä saa itselleen oman DECUServe-tunnuksen, jota käytetään lisenssitietojen anomiseen. Lisenssitietoja anotaan täyttämällä selaimella hakukaavake osoitteessa:

http://plato.ccsscorp.com/hobbyist_registration.php. (viitattu 31.7.2015)

Hakukaavakkeen täytön jälkeen käyttäjä saa antamaansa sähköpostiosoitteeseen tarvittavat lisenssitiedot sekä OpenVMS-käyttöjärjestelmälle että monille OpenVMS:n lisäosille ja apuohjelmille. Samassa sähköpostissa on myös latausosoitteet OpenVMS-käyttöjärjestelmälle ja pyydetyille lisäosille. Latausosoitteet ovat kirjautumisen vaativia ja myös tarvittavat kirjautumistiedot toimitetaan käyttäjälle sähköpostin välityksellä.

ALPHAVM-EMULAATTORIN ASETUKSET

Käyttäjä saa ladattua AlphaVM:n itselleen tgz-pakettina ja voi purkaa paketin siihen kansioon, jossa haluaa emulaattoria käyttää. AlphaVM:n mukana tulee apuohjelma *emumv-mkdisk*, jolla käyttäjä voi luoda emuloitavalle järjestelmälle massamuisteja. Tarvittavat massamuistit luodaan kirjoittamalla tyhjiä levykuvatiedostoja *emumv-mkdisk*:n avulla. Taulukossa 19 on lueteltuna *emumv-mkdisk:*n tukemia massamuisteja sekä eri massamuistien kokoja.

Taulukko 19 massamuistityypit

Tyyppi	Koko
DEC RZ26	1 Gt
DEC RZ28D	2 Gt
DEC RZ29B	4 Gt
DEC RZ59	8,5 Gt

Taulukosta 19 voidaan valita kaksi haluttua massamuistia otettavaksi käyttöön emuloitavassa järjestelmässä. Massamuisti tehdään kirjoittamalla Linux-käyttöjärjestelmän komentoriville esimerkiksi komento:

```
./emumv-mkdisk rz29b os_disk.dd
```

Tämä komento kirjoittaa isäntäjärjestelmän kovalevylle tiedoston *os_disk.dd*, jonka koko on 4 Gt ja tyyppi *rz29b*. DEC RZ29B on Digital Equipment Corporationin valmis-

tama SCSI (engl. Small Computer System Interface) standardin mukainen ALPHA-järjestelmissä käytettävä massamuisti.

Puretusta AlphaVM-paketista löytyy emulaattorin asetuksien esimerkkitiedosto. Asetustiedosto sisältää tiedot käyttöön otettavista laitteistokomponenteista sekä näiden laitteiden asetuksista. Esimerkkitiedosto löytyy purettujen tiedostojen joukosta nimellä *example.emu*. Asetustiedosto sisältää taulukossa 20 esitellyt asetukset.

Taulukko 20 AlphaMV:n asetukset

#	Asetus	Selite
1	Tyyppi	Halutun ALPHA-arkkitehtuurin tyyppi, esimerkiksi es40 833.
2	Prosessorien määrä	Valitun ALPHA-arkkitehtuurin käyttämien prosessorien määrä.
3	Sarjanumero	Järjestelmän sarjanumero
4	Keskeytysten taajuus	Järjestelmän keskeytysten kellotaajuus
5	Muistin koko	Järjestelmän muistin koko megatavuissa
6	Prosessorin tyyppi	Käytettävän prosessorin tyyppi
7	Sarjaportit	Sarjaporttien asetukset kuten nopeus ja yhteysportit
8	SCSI-ohjaimet	SCSI-ohjaimen asetukset
9	SCSI-massamuistit	SCSI-massamuistien tunnistetiedot sekä levykuvatiedostojen sijainnit
10	Verkkosovitin	Laitetason verkkoasetukset kuten MAC-osoite
11	Käynnistys	Käynnistettävä sovellus

Taulukossa 20 esitetyistä asetuksista käyttäjän tulee muokata ainakin rivejä 9 ja 10 koskevia tietoja, jotta emulaattori saadaan toimimaan halutulla tavalla. Muiden rivien asetuksia tulee muokata tarpeen mukaan.

Massamuistien asetukset tulee asettaa viittaamaan luotuihin levynkuvatiedostoihin, joille OpenVMS-käyttöjärjestelmä tullaan asentamaan ja jolle käyttäjä tulee tekemään omat tallennuksensa. Alla on esitettynä käyttöjärjestelmän asentamista varten valitun levykuvatiedoston asetukset:

```
scsi_disk dka0
{
          scsi_bus = 0;
          scsi_id = 0;
          scsi_lun = 0;
          file   = '/home/alpha_emulator/os_disk';
          server = default;
          async  = yes;
          removable = no;
          caching = no;
          write_through = no;
          trace_cmd = no;
          trace_sense = no;
          trace_io = no;
}
```

CD-asemaa koskeva SCSI-ohjain tulee laittaa osoittamaan OpenVMS-käyttöjärjestelmän asennusmediaan. CD-aseman asetukset voidaan tehdä esimerkiksi seuraavalla tavalla:

```
scsi_cdrom dka400
{
          # The default server is aio for images, basic for
          # CDROM devices.
          server = default;
          scsi_id = 4;
          file    = '/home/alpha_emulator/os.iso';
          # file = '/dev/cdrom';
          # file = '/dev/cd0';
}
```

Yllä olevassa esimerkissä määritellään CD-aseman tapa toimia palvelimena, annetaan asemalle tunnistenumero sekä levykuvatiedoston sijainti. Esimerkin ensimmäisellä rivillä määritellään tunniste *dka400,* jolla valittu levykuvatiedosto saadaan käyttöön emulaattoria käytettäessä.

Verkkosovittimen asetukset tulee muokata siten, että käytettävä MAC-osoite on käytettävässä järjestelmässä uniikki. Verkkosovittimen asetukset voidaan asettaa esimerkiksi seuraavilla komennoilla:

```
ether tap0
{
          type = dec21040;
          server = tap;
          interface = 'tap0';
          mac_mode = user;
          mac_address = 0xE20E1C5465C2;
}
```

OPENVMS:N ASENNUS

OpenVMS käyttöjärjestelmän asentaminen emulaattoriin voidaan edellisten toimenpiteiden jälkeen aloittaa. Emulaattori käynnistetään antamalle emulaattorille parametriksi käytettävä asetustiedosto:

```
./alphavm_free example.emu
```

Tässä tutkielmassa käytetty *socatin* versio on 1.7.2.3. *Socat* tulee käynnistää seuraavilla parametreilla isäntäjärjestelmässä, jotta emulaattorin käyttö on mahdollista:

```
socat -,raw,echo=0,escape=0x1c tcp:127.0.0.1:20000
```

Asettamalla yhteyden tyyppi muotoon *raw*, emulaattorille lähetettävää merkistöä ei käsitellä tai muokata. Asetus *echo=0* estää käytettävää sovellusta kaiuttamasta lähetettäviä merkkejä käyttäjälle. Tällä asetuksella estetään salasanojen syöttämisessä syntyvät virhetilanteet, joita ilmeni kaiuttamisen ollessa käytössä. Asetuksella *escape=0x1c* asete-

taan käytettävä EOF (engl. End Of Line) koodaus. Tämä asetus on tärkeä jotta rivin loppuminen tulkitaan oikein käytettäessä *raw* tyyppiä datan lähetyksessä. Näillä asetuksilla voidaan varmistua siitä, että näppäimistön enter-näppäimen painaminen tuottaa halutun tuloksen. Kun tarvittavat asetukset on tehty, OpenVMS käyttöjärjestelmän asennus voidaan käynnistää syöttämällä emulaattorille komento:

```
boot dka400
```

Tällä komennolla emulaattori käynnistyy *dka400* nimiseltä järjestelmälevyltä. Käyttöjärjestelmän asentamisen aikana käyttäjälle esitetään joukko kysymyksiä, joihin käyttäjä vastaa haluamallaan tavalla. Kysymykset koostuvat esimerkiksi käyttäjänimeen, päivämäärään sekä käytettävään kieleen liittyvistä tiedoista. Käyttäjää myös ohjeistetaan asennuksen aikana riittävästi, jotta tietotekniikkaan perehtynyt käyttäjä pystyy käyttöjärjestelmän asentamaan. Käyttäjältä kysytään asennuksen aikana myös eri ohjelmistojen lisenssitietojen syöttämisestä, mutta nämä tiedot suositellaan syötettäväksi vasta käyttöjärjestelmän asennuksen jälkeen. OpenVMS:n asentamisen jälkeen käyttöjärjestelmä sijaitsee levyllä nimeltä *dka0*. Uudelleenkäynnistyksen yhteydessä asennettu OpenVMS-käyttöjärjestelmä saadaan käynnistettyä syöttämällä komento:

```
boot dka0
```

Tällä komennolla OpenVMS käynnistetään emulaattorin asetustiedostossa määritellyltä *dka0* levyltä.

KÄÄNTÄJIEN ASENNUS

Lisenssitietoja käsin syötettäessä tulee huomioida, että mikäli käyttää hyväksi kopiointia ja liittämistä, niin tietoa kannattaa syöttää vain enintään kaksi riviä kerrallaan, tai OpenVMS saattaa antaa virheilmoituksen ylivuodosta. Syötettävän lisenssitiedon esimerkki on esiteltynä alla siten muokattuna, että lisenssin autentikoinnin sekä tarkistussumman tiedot on muutettu väärinkäytösten estämiseksi.

```
LICENSE REGISTER UCX -
/ISSUER=DEC -
/AUTHORIZATION=HOBBYIST-VA-KEYXXXXX-XXXXX -
/PRODUCER=DEC -
/UNITS=0 -
/TERMINATION_DATE=31-AUG-2016 -
/ACTIVITY=CONSTANT=100 -
/CHECKSUM=2-CDOE-DANF-XXXX-XXXX
LICENSE DISABLE UCX/LOG/PRODUCER=DEC/ALL
LICENSE UNLOAD  UCX/LOG/PRODUCER=DEC
LICENSE ENABLE  UCX/LOG/PRODUCER=DEC/AUTH=HOBBYIST-VA-KEYXXXXX-
XXXXX
LICENSE LOAD    UCX/LOG/PRODUCER=DEC
```

Lisenssitietojen lisäksi yllä olevat komennot poistavat mahdollisen aikaisemman lisenssin käytöstä ja ottavat uuden lisenssin käyttöön. Lisenssi sisältää tiedon siitä, kuinka

pitkään se on voimassa. Näillä komennoilla voidaan siis varmistaa oikea toiminta myös siinä tapauksessa, että jo olemassa olevaa lisenssiä ollaan päivittämässä uudempaan. Lisenssitiedot sisältävä tekstitiedosto voidaan suorittaa muuttamalla tiedoston *TXT*-tiedostopääte *COM*-tiedostopäätteeksi. Tämän jälkeen tiedoston suorittaminen voidaan suorittaa esimerkiksi komennolla:

```
@lisenssit.com
```

Verkkopalveluiden käynnistäminen voidaan myös automatisoida syöttämällä järjestelmän käynnistystiedostoon *sys$common:[sysmgr]systartup_vms.com* komento:

```
@SYS$STARTUP:TCPIP$STARTUP.COM
```

Siirrettäessä kääntäjien asennustiedostoja emulaattoriin levykuvatiedoston käyttöönotto vaatii emulaattorin uudelleenkäynnistämisen. Siirron yhteydessä korruptoituvia tiedostoja voidaan yrittää korjata OpenVMS:n komennoilla. C-kääntäjän asennustiedoston *cc073.a* parametreja voidaan säätää komennolla:

```
SET FILE/ATTRIBUTES=(RFM:FIX,MRS:32256,LRL:32256,RAT:NONE)
cc073.a
```

Kun tiedostoja siirretään järjestelmien välillä verkon yli, tulee siirron tyyppi määritellä tiedostotyypin mukaan. Käytettäessä FTP-yhteyttä Ubuntusta OpenVMS:ään päin siirron tyypin määritys tapahtuu antamalla siirto-ohjelmalle komento *binary*. Mikäli FTP-yhteys muodostetaan OpenVMS järjestelmästä Ubuntuun, niin yhteyden tyyppi määritellään antamalla siirto-ohjelmalle komento *set type image* [20]. C-kääntäjän asentaminen saadaan suoritettua syöttämällä seuraava komento siinä hakemistossa, johon kääntäjän asennustiedostot on sijoitettuna:

```
@sys$update:vmsinstal CC073
```

Fortran kääntäjän asentaminen suoritetaan syöttämällä seuraava komento siinä hakemistossa, johon kääntäjän asennustiedosto on sijoitettuna:

```
product install fortran
```

Asennettu C-kääntäjä sekä sen versio voidaan varmistaa asennuksen jälkeen syöttämällä komento:

```
@sys$system:decc$show_versions.com
```

Fortran kääntäjän versio voidaan tarkistaa asennusohjelman luomasta dokumentaatiosta tiedostosta *sys$common:[syshlp]fortran.release_notes*.

Testiohjelman luomista varten käyttäjän tulee siirtyä sellaiseen sijaintiin OpenVMS-käyttöjärjestelmässä, johon hänellä on kirjoitusoikeus. Kun OpenVMS:ssä kirjoitetaan

tiedostoon, kannattaa päätteen emuloinnin tyyppi säätää komennolla *set/term vt100*. Asettamalla päätteen tyyppi oikein, vältytään tekstin syöttämisessä merkistöjen tulkintaerojen aiheuttamilta ongelmilta. OpenVMS-käyttöjärjestelmässä saadaan luotua käännettävä C-kielinen koodi syöttämällä seuraavat komennot:

```
create hello.c
#include <stdio.h>
#include <stdlib.h>
main()
  {
  printf("Hello, World!\n" );
  exit( EXIT_SUCCESS );
}
```

Luodun testiohjelman suorittamiseen käytetty komento sekä oletettu tuloste ovat:

```
run hello
Hello, World!
```

Fortranilla toteutetun Hello World! -testiohjelman luomisessa on helpompi käyttää editoria, kuin *create*-komentoa. Tämä johtuu siitä, että Fortran edellyttää juuri oikealla tavalla tehtyjä sisennyksiä, jotka on helpompi varmistaa editorissa.

Create on DCL:n komento, jonka avulla voidaan luoda parametrina annetun nimisiä tiedostoja. Rivit jotka kirjoitetaan *create*-komennon jälkeen, lisätään luotuun tiedostoon. Tiedoston luominen lopetetaan ja tiedosto tallennetaan painamalla näppäimiä Ctrl+Z. Tiedoston luomisen jälkeen, testiohjelma tulee kääntää ja linkittää syöttämällä seuraavat komennot:

```
cc hello
link hello
```

OpenVMS:n oma editori voidaan käynnistää komennolla *edit* ja antaa sille parametriksi luotavan tiedoston nimi *hello.f*. Tiedostoon kirjoitettava lähdekoodi on esitetty alla. Tässä Fortranilla toteutetussa lähdekoodissa ensimmäiset kuusi saraketta tulee jättää tyhjäksi. Editoria käytettäessä tiedoston tallennus tapahtuu painamalla näppäimiä Ctrl+Z. Mikäli editorista halutaan poistua tallentamatta, niin käyttäjän tulee painaa F10-näppäintä.

```
program hello
 print *, ”Hello Wordl!”
end program hello
```

LIITE 2 LINKITYKSEN VIRHEILMOITUS MFCASS-ASEMALLA

```
%LINK-W-ILLRECTYP, illegal record type (240.)
     in module SAMPLE_FEP record 2 file
     CASS:[PRODUCTION.TEST_SOFTWARE.FEP]SAMPLE_FEP.OBJ;1
%LINK-W-ILLRECTYP, illegal record type (16384.)
     in module SAMPLE_FEP record 3 file
     CASS:[PRODUCTION.TEST_SOFTWARE.FEP]SAMPLE_FEP.OBJ;1
%LINK-W-ILLRECTYP, illegal record type (30260.)
     in module SAMPLE_FEP record 4 file
     CASS:[PRODUCTION.TEST_SOFTWARE.FEP]SAMPLE_FEP.OBJ;1
%LINK-W-ILLRECTYP, illegal record type (5921.)
     in module SAMPLE_FEP record 5 file
     CASS:[PRODUCTION.TEST_SOFTWARE.FEP]SAMPLE_FEP.OBJ;1
%LINK-W-ILLRECTYP, illegal record type (0.)
     in module SAMPLE_FEP record 6 file
     CASS:[PRODUCTION.TEST_SOFTWARE.FEP]SAMPLE_FEP.OBJ;1
%LINK-W-ILLRECTYP, illegal record type (17427.)
     in module SAMPLE_FEP record 7 file
     CASS:[PRODUCTION.TEST_SOFTWARE.FEP]SAMPLE_FEP.OBJ;1
%LINK-W-SEQNCE, illegal record sequence
     in module SAMPLE_FEP file
     CASS:[PRODUCTION.TEST_SOFTWARE.FEP]SAMPLE_FEP.OBJ;1
%LINK-W-ILLRECTYP, illegal record type (0.)
     in module SAMPLE_FEP record 9 file
     CASS:[PRODUCTION.TEST_SOFTWARE.FEP]SAMPLE_FEP.OBJ;1
%LINK-W-ILLRECTYP, illegal record type (0.)
     in module SAMPLE_FEP record 10 file
     CASS:[PRODUCTION.TEST_SOFTWARE.FEP]SAMPLE_FEP.OBJ;1
%LINK-W-ILLRECTYP, illegal record type (0.)
     in module SAMPLE_FEP record 11 file
     CASS:[PRODUCTION.TEST_SOFTWARE.FEP]SAMPLE_FEP.OBJ;1
%LINK-W-SEQNCE, illegal record sequence
     in module SAMPLE_FEP file
     CASS:[PRODUCTION.TEST_SOFTWARE.FEP]SAMPLE_FEP.OBJ;1
%LINK-W-ILLRECTYP, illegal record type (5044.)
     in module SAMPLE_FEP record 13 file
     CASS:[PRODUCTION.TEST_SOFTWARE.FEP]SAMPLE_FEP.OBJ;1
%LINK-W-ILLRECTYP, illegal record type (0.)
     in module SAMPLE_FEP record 14 file
     CASS:[PRODUCTION.TEST_SOFTWARE.FEP]SAMPLE_FEP.OBJ;1
%LINK-W-NOEOM, no end-of-module record found
     in module CASS:[PRODUCTION.TEST_SOFTWARE.FEP]SAMPLE_FEP.OBJ;1
%LINK-W-SEQNCE, illegal record sequence
     in module SAMPLE_FEP file
     CASS:[PRODUCTION.TEST_SOFTWARE.FEP]SAMPLE_FEP.OBJ;1
%LINK-I-OPENIN, error opening SYS$COMMON:[SYSLIB]IMAGELIB.OLB; as
input
-RMS-E-FNF, file not found
%LINK-I-OPENIN, error opening SYS$COMMON:[SYSLIB]STARLET.OLB; as
input
-RMS-E-FNF, file not found
%LINK-W-EMPTYFILE, no modules found in file
CASS:[PRODUCTION.TEST_SOFTWARE.FEP]SAMPLE_FEP.OBJ;1
%LINK-W-USRTFR, image
CASS:[PRODUCTION.TEST_SOFTWARE.FEP]SAMPLE_FEP.OBJ;1
has no user transfer address
```

LIITE 3 FEP WIZARDIN LUOMA ESIMERKKILUOKKA

```
public class SampleFEP : FEPBase
{
/// <summary>
/// Default constructor
/// </summary>
public SampleFEP()
{
	// Set the published application name.
	mName = "FEP__SAMPLE_FEP";

	// Specify the configuration information.
	SetupConfiguration(mName, "Add two hex values
	together and return the result hex");

	AddConfigurationParameter("PARAM-1", FEP_ParamDirection.IN,
	FEP_ParamType.DIGITAL, "First HEX value in");
	AddConfigurationParameter("PARAM-2", FEP_ParamDirection.IN,
	FEP_ParamType.DIGITAL, "Second HEX value in");
	AddConfigurationParameter("PARAM-3", FEP_ParamDirection.OUT,
	FEP_ParamType.DIGITAL, "Result HEX value out");
	AddConfigurationParameter("PARAM-4", FEP_ParamDirection.OUT,
	FEP_ParamType.BOOL, "Overflow");
	AddConfigurationParameter("PARAM-5", FEP_ParamDirection.OUT,
	FEP_ParamType.BOOL, "Exception");
	ActivateConfiguration();
}

/// <summary>
/// Implement the specific FEP logic.
/// </summary>
/// <param name="aDataNode">TPML node being processed.</param>
/// <returns></returns>
protected override object ExecuteFEP_Logic(System.Xml.XmlNode
aDataNode)
{
	// Declare variables for input and output parameters.
	// This is an ATLAS DIGITAL type.
	DirectTest.Interfaces.Common.IDigital oPARAM_1;
	// This is an ATLAS DIGITAL type.
	DirectTest.Interfaces.Common.IDigital oPARAM_2;
	// This is an ATLAS DIGITAL type.
	DirectTest.Interfaces.Common.IDigital oPARAM_3;
	// This is an ATLAS BOOLEAN type.
	bool bPARAM_4;
	// This is an ATLAS BOOLEAN type.
	bool bPARAM_5;

	// Fetch the input variables.
	oPARAM_1 =
	(DirectTest.Interfaces.Common.IDigital)GetParameterValue(0);

	oPARAM_2 =
	(DirectTest.Interfaces.Common.IDigital)GetParameterValue(1);
```

```
54              // The following block of code is a sample of variable usage.
                // Replace this with your own FEP logic as needed.
56              // --------- Sample begin ---------
                string sOutput = string.Empty;
58              sOutput += "Input value of PARAM-1 is " + oPARAM_1.ToString()
                + System.Environment.NewLine;
60              sOutput += "Input value of PARAM-2 is " + oPARAM_2.ToString()
                + System.Environment.NewLine;
62              System.Windows.Forms.MessageBox.Show(sOutput, mName);
                // ---------- Sample end ----------
64
                // Set output variables.
66              SetParameterValue(2, oPARAM_3);
                SetParameterValue(3, bPARAM_4);
68              SetParameterValue(4, bPARAM_5);

70              return null;
            }
72  }
```