OULUN AMMATTIKORKEAKOULU

Jari Kylmäoja

MODEEMILOKEISTA GENEROITUJEN OPERAATTORITESTITIEDOSTOJEN TESTAUSAUTOMAATIO

MODEEMILOKEISTA GENEROITUJEN OPERAATTORITESTITIEDOSTOJEN TESTAUSAUTOMAATIO

Jari Kylmäoja
Opinnäytetyö
Kevät 2022
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, laite- ja tuotesuunnittelun suuntautumisvaihtoehto

---

Tekijä: Jari Kylmäoja
Opinnäytetyön nimi: modeemilokeista generoitujen operaattoritestitiedostojen testausautomaatio
Työn ohjaajat: Teemu Korpela (Oamk), Marko Kukkohovi (MediaTek),
Kristian Lappalainen (MediaTek), Ari Simonen (MediaTek), Teemu Partanen (MediaTek)
Työn valmistumislukukausi ja -vuosi: Kevät 2022
Sivumäärä: 30

Opinnäytetyön aiheena oli toteuttaa testausautomaatio, jossa päivittäin ladatuista modeemilokeista generoidaan operaattoritestitiedostoja testattavaksi ja raportoidaan saadut testitulokset eteenpäin tapauskohtaisille vastuuhenkilöille. Generointiskriptin jatkokehittämisen lisäksi työhön sisältyi lokien ennakkotarkistusskriptin kirjoittaminen, joka analysoi lokit ja varmistaa vaadittujen tapahtumien löytymisen niistä. Puuttuvia tapahtumia pyritään löytämään ja yhdistelemään toisista, yhteensopivista lokitiedostoista ennen testitiedostojen generointivaiheeseen siirtymistä.

Testausautomaation varhaiset vaiheet ja tarvittavat eräajot toteutettiin Linux-ympäristössä Python- ja Bash-skriptejä käyttäen. Automaatiovaiheiden välitulokset kirjoitetaan CSV-tiedostoon. Generointiskriptin tuottamat testitiedostot ovat C-kielisiä, kuten niiden testausympäristökin.

Opinnäytetyö alkoi tutkimalla manuaalisesti ladattuja modeemilokeja ja kartoittamalla puuttuvien tapahtumien tyyppejä ja niiden puuttumisen todennäköisyyttä. Tämän perusteella ennakkotarkistus- ja generointiskriptiin suunniteltiin ja toteutettiin lokitiedostojen tietojen yhdistelyn mahdollistavaa älykkyyttä. Lisäksi tutkittiin päivittäin saatujen lokipakettien hakemistorakennetta ja niiden sisältämiä tiedostotyyppejä. Lokipakettien sisältöä optimoitiin tunnistamalla ja jättämällä niistä pois tarpeettomat hakemistot ja tiedostotyypit.

Asiasanat: Python, IMS, VoLTE, modeemiloki, testausautomaatio, Linux

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Device and Product Design

Author: Jari Kylmäoja
Title of thesis: test automation of test cases generated from modem logs
Supervisors: Teemu Korpela (Oamk), Marko Kukkohovi (MediaTek),
Kristian Lappalainen (MediaTek), Ari Simonen (MediaTek), Teemu Partanen (MediaTek)
Term and year when the thesis was submitted: Spring 2022
Number of pages: 30

The objective of the thesis was to develop testing automation, in which test case files are generated from modem log files. Generated test case files are tested in the test environment, and the results are reported to relevant persons. In addition to further developing the generation script, a log prechecker script was written for analyzing the logs in advance, ensuring that required events are present in them. Missing events are searched and combined from other, compatible log files, prior to advancing to the test case file generation phase.

The early phases and batch run tasks in the automation were implemented in Linux environment, using Bash- and Python-scripts. In middle phases, intermittent results are written out in a CSV-file. The test case files produced by the generation script are written in C-language, as is the testing environment.

The thesis work began by studying manually downloaded modem log files and examining the types and probability of missing events. Based on this, intelligence was developed in the prechecker and generation scripts, enabling combining information from separate log files. Additionally, the folder structure and the contained file types of daily log archives was examined, and their contents were optimized by identifying and omitting redundant directories and file types.

Keywords: Python, IMS, VoLTE, modem log, test automation, Linux

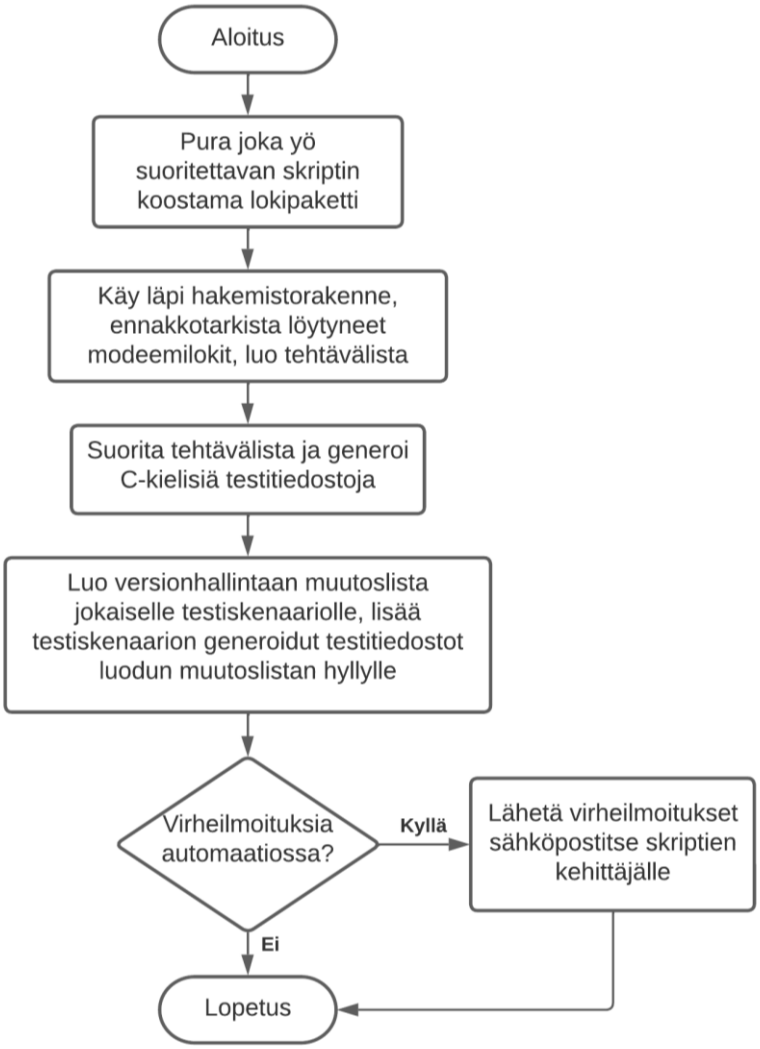
SISÄLLYS

LYHENTEET

CSV	Comma Separated Values, tekstitiedosto jossa yksittäiset arvot erotettu pilkuilla toisistaan
FTP	File Transfer Protocol, yleinen tiedonsiirtoprotokolla tiedostojen siirtoon tietokoneverkoissa
HTTP	Hypertext Transfer Protocol, selainten ja WWW-palvelimien käyttämä tiedonsiirtoprotokolla
IMS	IP Multimedia Subsystem, yleinen arkkitehtuuri IP-multimediapalveluiden välittämiseen
MCC	Mobile Country Code, kolminumeroinen koodi, joka yksilöi maan matkapuhelinverkoissa
MNC	Mobile Network Code, kaksi- tai kolminumeroinen koodi, joka yksilöi operaattorin matkapuhelinverkoissa
MO	Mobile Originating, lähtevä (puhelu)
MT	Mobile Terminating, saapuva (puhelu)
RegEx	Regular Expression, merkkijono, joka määrittelee hakulausekkeen, jolla voidaan hakea tuloksia esimerkiksi tekstistä
SDP	Session Description Protocol, esitysmuoto yksittäisen multimediayhteysistunnon kuvailemiseen
SIP	Session Initiation Protocol, reaaliaikaisissa istunnoissa käytetty tekstiperusteinen tietoliikenneprotokolla
SMTP	Simple Mail Transfer Protocol, Internetin yleinen sähköpostiviestien tiedonsiirtoprotokolla
SQL	Structured Query Language, yleinen tietokantojen hallinnoinnissa ja käsittelyssä käytetty ohjelmointikieli
URI	Uniform Resource Identifier, merkkijono joka yksilöi web-teknologioiden käyttämiä resursseja
VoIP	Voice over IP, menetelmä ääniyhteyksien ja multimediaistuntojen välittämiseen IP-verkkojen yli
VoLTE	Voice over LTE, matkapuhelimien ja datapäätteiden käyttämä nopea langattoman viestinnän standardi

1 JOHDANTO

Opinnäytetyön aiheena oli toteuttaa Linux-ympäristössä Python- ja Bash-skriptejä käyttäen testausautomaatio, jossa generoidaan päivittäin saaduista modeemilokeista operaattoritestitiedostoja testattavaksi (kuva 1). Koko automaation oli tarkoitus toimia hyvin itsenäisesti ja tarve käyttäjän puuttumiselle sen toimintaan tulisi olla minimaalista. Automaation alkuvaiheissa ennakkotarkistusskripti analysoi lokit ja varmistaa vaadittujen tapahtumien löytymisen niistä, kuten esimerkiksi puhelimen rekisteröitymisen verkkoon. Oleellisten tapahtumien puuttuessa pyritään niitä löytämään ja yhdistelemään toisista, yhteensopivista lokitiedostoista ennen generointivaiheeseen siirtymistä.



KUVA 1. Vuokaavio automaatiosta

Generointivaiheessa erillinen Python-skripti lukee edellisen vaiheen tulokset, joiden perusteella se tuottaa eräajona testitiedostoja, jotka ovat C-kielisiä, kuten niiden testausympäristökin. Testitiedostot sisältävät kaikki mukaan otettuihin tapahtumiin liittyvät SIP-viestit, testiajossa suoritettavat päätestisekvenssit ja testiajossa käytettävät, modeemilokitiedoston sisäisistä viesteistä luetut konfiguraatioarvot. Myös generointiskriptiä jatkokehitettiin työn aikana tarpeen mukaan.

Generoidut testitiedostot ajetaan testausohjelman läpi, jossa luodaan testitiedostosta luetun konfiguraation ja siitä löytyvien tapahtumien perusteella testitilanne, jossa simuloidaan puhelimen ja verkon välisiä tapahtumia. Testi läpäistään, jos virheellistä toimintaa ei havaita. Saadut testitulokset raportoidaan eteenpäin tapauskohtaisesti oikeille vastuuhenkilöille.

Työn alussa tutkittiin eri modeemilokeja ja kartoitettiin puuttuvien tapahtumien tyyppejä ja niiden puuttumisen todennäköisyyttä. Tämän perusteella ennakkotarkistus- ja generointiskriptiin suunniteltiin ja toteutettiin lokitiedostojen tietojen yhdistelyn mahdollistavaa älykkyyttä. Lisäksi tutkittiin päivittäin saatujen lokipakettien hakemistorakennetta ja niiden sisältämiä tiedostotyyppejä. Lokipakettien sisältöä optimoitiin tunnistamalla ja jättämällä niistä pois tarpeettomaksi havaitut hakemistot ja tiedostotyypit.

Opinnäytetyön tilaaja on Oulussa toimiva MediaTek Wireless Finland Oy, joka keskittyy pääosin langattoman viestinnän ohjelmisto- ja algoritmikehitykseen ja osittain myös laitekehitykseen. Sen emoyhtiö, taiwanilainen MediaTek Inc. on eräs maailman johtavia mikropiiri- ja puolijohdevalmistajia, jolla ei ole omia tuotantolaitoksia, mutta jonka suunnittelemia, korkeatasoisia piirisarjoja käytetään monenlaisessa kuluttajaelektroniikassa ja tietokoneiden oheislaitteissa (1).

2 KÄYTETYT TEKNOLOGIAT

Tässä luvussa kerrotaan työssä käytetyistä teknologioista, joihin kuuluivat Python-, C- ja Perl-ohjelmointikielet, Linuxin Bash-skriptit ja FTP-tiedonsiirtomenetelmä.

2.1 Python-ohjelmointikieli

Python on helposti opittava ja monipuolinen ohjelmointikieli, joka tarjoaa tehokkaat korkean tason tietorakenteet ja helpon lähestymistavan olio-ohjelmointiin. Se sopii hyvin skriptaukseen ja nopeaan ohjelmistokehitykseen useimmilla sovellusalustoilla, aiheesta riippumatta. (2.)

Python-kielessä esimerkiksi koodilohkoja ei ympäröidä aaltosulkein kuten joissakin ohjelmointikielissä, vaan lohkot sisennetään välilyöntejä käyttäen (kuva 2). Muuttujien ja funktioparametrien tyyppejä ei tarvitse määritellä, ja ne voivat myös muuttua erityyppiseksi koska tahansa. Lisäksi aliohjelmille ei tarvitse erikseen määritellä palautustyyppiä ja sama aliohjelma voi jopa palauttaa eri muuttujatyyppejä.

Alkuperäiset skriptit olivat Python-kielellä kirjoitettuja, ja niitä oli jatkokehitetty myös edellisissä, tutkintoon sisältyvissä yrityslähtöisissä tuotekehitysprojekteissa. Työn tilaaja rohkaisi ohjelmointikielen vaihtamiseen sopivampaan tarvittaessa, mutta koska Python osoittautui vaatimustason kasvaessakin edelleen toimivaksi ratkaisuksi, jatkettiin sen käyttämistä. Yrityksen sisäisesti kehitetty Python-moduuli mahdollisti modeemilokien avaamisen ja monenlaisen käsittelyn skripteissä. Myös työn aikana kirjoitettua uutta koodia ryhmiteltiin tarpeen mukaan omiin moduuleihinsa selkeyden ja järjestelmällisyyden vuoksi sekä ylläpidon helpottamiseksi.

```
11
12      if apply_message_protocol_swaps:
13          protocol_swap_table = [
14              [ None,      1,     False,     'operator1',     'headers_deregistration_ue_desubscribe_req' ],
15              [ None,      2,     True,      'operator2',     '<PRACK_183_REQUESTS>'                      ],
16              [ 20154,     0,     False,     'operator2',     'headers_registration_ue_subscribe_req'     ],
17              [ 20204,     1,     False,     'operator3',     'headers_mt_call_ue_bye_req'                ],
18              [ 20290,     2,     True,      'operator2',     '<PRACK_183_REQUESTS>'                      ],
19              [ 20154,     2,     True,      'operator2',     'headers_registration_ue_subscribe_req'     ],
20          ]
21
22          swap_count = 0
23
24          for campaign_id, call_type, is_separate_reg, operator, swap_target in protocol_swap_table:
25              if campaign_id == tc_data['campaign_id']:
26                  if call_type == tc_data['call_type']:
27                      if operator == tc_data['operator']  and  is_separate_reg == tc_data['is_separate_reg']:
28                          if '<PRACK_183_REQUESTS>' == swap_target:
29                              for hdr in ['headers_mo_call_ue_prack_183_req', 'headers_mo_call_ue_prack_183_req']:
30                                  message_protocol_swap(message_list, hdr)
31                                  swap_count += 1
32                          else:
33                              message_protocol_swap(message_list, swap_target)
34                              swap_count += 1
35
36          print('Message protocol modification count: %d.' % swap_count)
37
```

KUVA 2. Python-koodia. Operaattorien nimet muutettu.

2.2 C-ohjelmointikieli

Testitiedoston generointivaiheen jälkeen työnkulussa siirrytään ohjelmointikielessä Pythonista C:hen, sillä generoidut testitiedostot ovat C-kielisiä. Ne sisältävät kaikki simuloitaviin tapahtumiin liittyvät SIP-referenssiviestien sisällöt merkkijonomuuttujina, testiajossa suoritettavat päätestisekvenssit ja testiajossa käytettävät, modeemilokitiedoston sisäisistä viesteistä luetut konfiguraatioarvot.

Mukaan otettujen SIP-viestien käsittelyssä ja ryhmittelyssä puhelutapahtuman mukaan on myös hyödynnetty C-kielen struktuureita (kuva 3), käyttäen viittauksia testitiedoston alussa määriteltyihin merkkijonomuuttujiin. Usein myös sisempiä struktuureita on käytetty. Yksittäinen tapahtuma voi olla esimerkiksi puhelun pitoon asettaminen, joka käsittää 4–8 SIP-viestiä.

```
static UT_UA_OP_CALL_STR call_messages_1 = {
    .dial_string_ptr =          "96711*****",
    .uri_scheme =               VOLTE_MO_CALL_URI_SCHEME_TYPE_NONE,
    .invite_initial_req =       {.protocol = PROTOCOL_TCP, .headers_ptr = headers_mo_call_ue_invite_initial_req, .body_ptr = body_mo_call_a_ue_invite_initial_req},
    .invite_100_resp =          {.protocol = PROTOCOL_TCP, .headers_ptr = headers_mo_call_nw_invite_100_resp},
    .invite_183_1_resp =        {.protocol = PROTOCOL_TCP, .headers_ptr = headers_mo_call_nw_invite_183_1_resp, .body_ptr = body_mo_call_a_nw_invite_183_1_resp, .check
    .prack_183_1_req =          {.protocol = PROTOCOL_UDP, .headers_ptr = headers_mo_call_ue_prack_183_1_req},
    .prack_183_1_resp =         {.protocol = PROTOCOL_UDP, .headers_ptr = headers_mo_call_nw_prack_200_183_1_resp},
    .invite_180_1_resp =        {.protocol = PROTOCOL_TCP, .headers_ptr = headers_mo_call_nw_invite_180_1_resp},
    .invite_200_resp =          {.protocol = PROTOCOL_TCP, .headers_ptr = headers_mo_call_nw_invite_200_resp},
    .ack_req =                  {.protocol = PROTOCOL_TCP, .headers_ptr = headers_mo_call_ue_ack_req}
};
```

KUVA 3. Esimerkki struktuurista, jossa myös sisempi struktuuri. Puhelinnumero osittain piilotettu.

2.3 Perl-ohjelmointikieli

Projektikoordinaattorin toteuttamassa lokien keräilyvaiheessa on käytetty Perl-kielistä skriptiä. Skripti kerää alkuperäisistä kenttätestien SQL-tietokannoista dataa ja tallentaa sen yksinkertaistetussa muodossa erilliseen tietokantaan, jossa hakua tarkennetaan vielä tietylle ajanjaksolle ja tehdään muita tarkistuksia. Tuloksista poimitaan ainoastaan testiskenaariot, jotka sisältävät kaikki kolme eri puhelutapahtumatyyppiä.

2.4 File Transfer Protocol -tiedonsiirtomenetelmä

File Transfer Protocol (FTP) on yleinen tiedonsiirtomenetelmä, jota käytetään tiedostojen siirtoon tietoliikenneverkossa palvelimelta asiakkaalle. Se noudattaa asiakas–palvelin-mallista arkkitehtuuria, jossa hallinta ja datayhteydet asiakkaan ja palvelimen välillä on erotettu toisistaan. Käyttäjät todennetaan joko normaalilla sisäänkirjautumisella tai palvelimen mahdollistaessa anonyymisti. FTP:ssä tiedonsiirto on usein salattua kirjautumistietojen ja sisällön suojaamiseksi. (3.)

2.5 Bash-skriptit

Kuten Batch-tiedostojen skriptauksessa Windowsissa, Linuxin Bash-skriptauksessa komentojoukkoja luetaan ja suoritetaan järjestyksessä tekstitiedostosta tai niitä kirjoitetaan suoraan komentorivillä. Skriptaus Linuxissa on kuitenkin Windowsin Batch-skriptausta huomattavasti monipuolisempaa, ja kuten ohjelmoinnissa, skripteissä voidaan käyttää ehto- ja silmukkalauseita sekä funktioita. Useimmat Linuxin graafisessa käyttöliittymässä tehtävät toimet on mahdollista suorittaa komentoriviltä ja skriptata, mikä mahdollistaa tarvittaessa hyvin monimutkaisten tehtävien automatisoinnin (4).

Koska suurin osa automaation tehtävistä pystyttiin toteuttamaan Python-skripteillä, tarvittiin Bash-skriptausta vain muutaman yksinkertaisen komennon ajamiseen, kuten esimerkiksi joidenkin Python-skriptien kutsumiseen kiinteillä parametreilla ja niiden tuottamien tiedostojen siirtämiseen eri hakemistoon. Myös lokien keräilyvaiheessa käytettiin Bash-skriptiä, joka käy läpi mukaan poimittujen lokien hakemistopolut läpi rekursiivisesti ja poimii niistä vain oleelliset tiedostot. Kootut tiedostot pakataan ja siirretään lopuksi toiselle työalueelle FTP:tä käyttäen. Siirto on suojattua, yrityksen sisäistä tiedonsiirtoa.

3 MODEEMILOKIEN ANALYSOINTI JA KÄSITTELY

MediaTekin työntekijät suorittavat toisinaan eri maissa ja eri operaattorien verkoissa monenlaisia, monimutkaisuudeltaan ja pituudeltaan vaihtelevia testejä, joilla pyritään varmistamaan puhelinohjelmiston oikeanlainen toiminta sekä tavallisissa että poikkeuksellisissa tilanteissa. Yksittäisessä testauskampanjassa suoritetaan satoja erilaisia testejä, joista jokaisesta pyritään tallentamaan modeemilokitiedosto.

3.1 Session Initiation Protocol -tietoliikenneprotokolla

Session Initiation Protocol (SIP) on tekstiperusteinen tietoliikenneprotokolla, jolla voidaan aloittaa, ylläpitää ja päättää reaaliaikaisia istuntoja. Sitä käytetään multimediayhteyksien signalointiin ja hallintaan, ja sen käyttökohteita voivat olla esimerkiksi ääni- ja videopuhelut tai pikaviestikeskustelut. Kommunikoinnin toimiminen edellyttää, että osanottajien väliset viestit noudattavat protokollassa määriteltyä, tietynlaista muotoa ja viestijärjestystä. SIP:ssä on tunnistettavissa monia HTTP:ssä ja SMTP:ssä esiintyviä piirteitä. (5.) SIP-viestien lukeminen, tulkitseminen ja muu käsittely muodostaa merkittävän osan opinnäytetyön tehtävistä.

SIP-protokollassa viestit ovat joko pyyntötyyppisiä (kuva 4) tai vastaustyyppisiä. Pyyntötyyppisten viestien ensimmäinen rivi sisältää URI:n, jolle pyyntö tulee lähettää, sekä metodin (taulukko 1), joka kuvaa pyynnön tarkoitusta. Vastaustyyppisten viestien ensimmäinen rivi sisältää vastauskoodin. (5.) Se on kolminumeroinen koodi, josta voidaan päätellä pyynnön onnistuminen. Hyväksytty pyyntö palauttaa yleensä koodin ”200”, ja muut palautuskoodit merkitsevät virhettä pyynnön käsittelyssä. Virhetapauksissa palautuskoodista voidaan virhekooditaulukkoa apuna käyttäen päätellä virheen tyyppi, minkä lisäksi palautuskoodin ohessa on yleensä annettu lyhyt, selkokielinen kuvaus ongelmasta.

```
REGISTER sip:ims.mnc005.mcc244.3gppnetwork.org SIP/2.0
Via: SIP/2.0/UDP [5555::a:b:c:d]:1400; branch=z9hG4bKnashds7
Max-Forwards: 70
P-Access-Network-Info: 3GPP-E-UTRAN-TDD; utran-cell-id-
     3gpp= 244005F3F5F7
From: <sip:kristiina@example.com>;tag=4fa3
To: <sip:kristiina@example.com >
Contact: <sip:[5555::a:b:c:d]:1400>;
expires=600000;
+sip.instance="<urn:gsma:imei:90420156-025763-0>";
+g.3gpp.smsip;
+g.3gpp.icsi-ref="urn%3Aurn-7%3A3gpp-service-ims.icsi.mmtel"[1]
Call-ID: apb03a0s09dkjdfglkj49111
Authorization: Digest username="private_user1@example.com",
        realm="ims.mnc005.mcc244.3gppnetwork.org", nonce="",
        uri="sip:ims.mnc005.mcc244.3gppnetwork.org",response=""
Security-Client: ipsec-3gpp; alg=hmac-sha-1-96; spi-c=1111;
     spi-s:=2222; port-c=9999; port-s=1400
Require: sec-agree
Proxy-Require: sec-agree
Supported: path
CSeq: 1 REGISTER
Content-Length: 0
```

KUVA 4. Esimerkki rekisteröintitapahtumaan liittyvästä SIP-viestistä (6)

TAULUKKO 1. Erilaisia SIP-viesteissä esiintyviä metodeja (5)

REGISTER	Rekisteröi To-kentässä annettu URI sijaintipalvelimelle, ja liitä siihen Contact-kentässä annettu verkko-osoite
INVITE	Aloita vuoropuhelu puhelun pystyttämiseksi
ACK	Vahvista, että alkuperäiseen INVITE-pyyntöön liittyvä päätösvastaus on vastaanotettu
BYE	Päätä vuoropuhelu ja lopeta puhelu
CANCEL	Peruuta vireillä oleva pyyntö
UPDATE	Muokkaa istunnon tilaa vuoropuhelun tilaa muuttamatta
REFER	Tee puhelunsiirtopyyntö vastaanottajalle
PRACK	Väliaikainen vahvistus
SUBSCRIBE	Tee tilauspyyntö ilmoitusviesteille tietyntyyppiseen tapahtumaan liittyen
NOTIFY	Ilmoita tilaajalle tietyntyyppiseen tapahtumaan liittyvistä muutoksista
PUBLISH	Julkaise ilmoitus tapahtumasta ilmoittajapalvelimelle
MESSAGE	Välitä tekstiviesti
INFO	Lähetä tietoa istunnon aikana muuttamatta sen tilaa
OPTIONS	Tiedustele toisen päätepisteen kyvykkyyttä

3.2 Session Description Protocol

Session Description Protocol (SDP) on esitysmuoto, jolla voidaan kuvailla yksittäistä multimediayhteysistuntoa. Sen pääkäyttötarkoitus on tukea suoratoistomediasovelluksia, kuten videokonferenssipuheluita ja VoIP:tä. Se ei itsessään välitä mediavirtaa, vaan sitä käytetään päätepisteiden välisissä neuvotteluissa, joissa sovitaan etukäteen esimerkiksi puhelun aikana käytettävistä mediatyypeistä ja muista ominaisuuksista. SDP muodostuu joukosta tekstimuotoisia kenttiä omilla riveillään (kuva 5). (7.) SDP voi sisältyä SIP-viesteihin niiden body-osassa.

```
v=0
o=jdoe 2890844526 2890842807 IN IP4 10.47.16.5
s=SDP Seminar
i=A Seminar on the session description protocol
u=http://www.example.com/seminars/sdp.pdf
e=j.doe@example.com (Jane Doe)
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
a=recvonly
m=audio 49170 RTP/AVP 0
m=video 51372 RTP/AVP 99
a=rtpmap:99 h263-1998/90000
```

KUVA 5. Esimerkki Session Description Protocol -osasta SIP-viestissä (8)

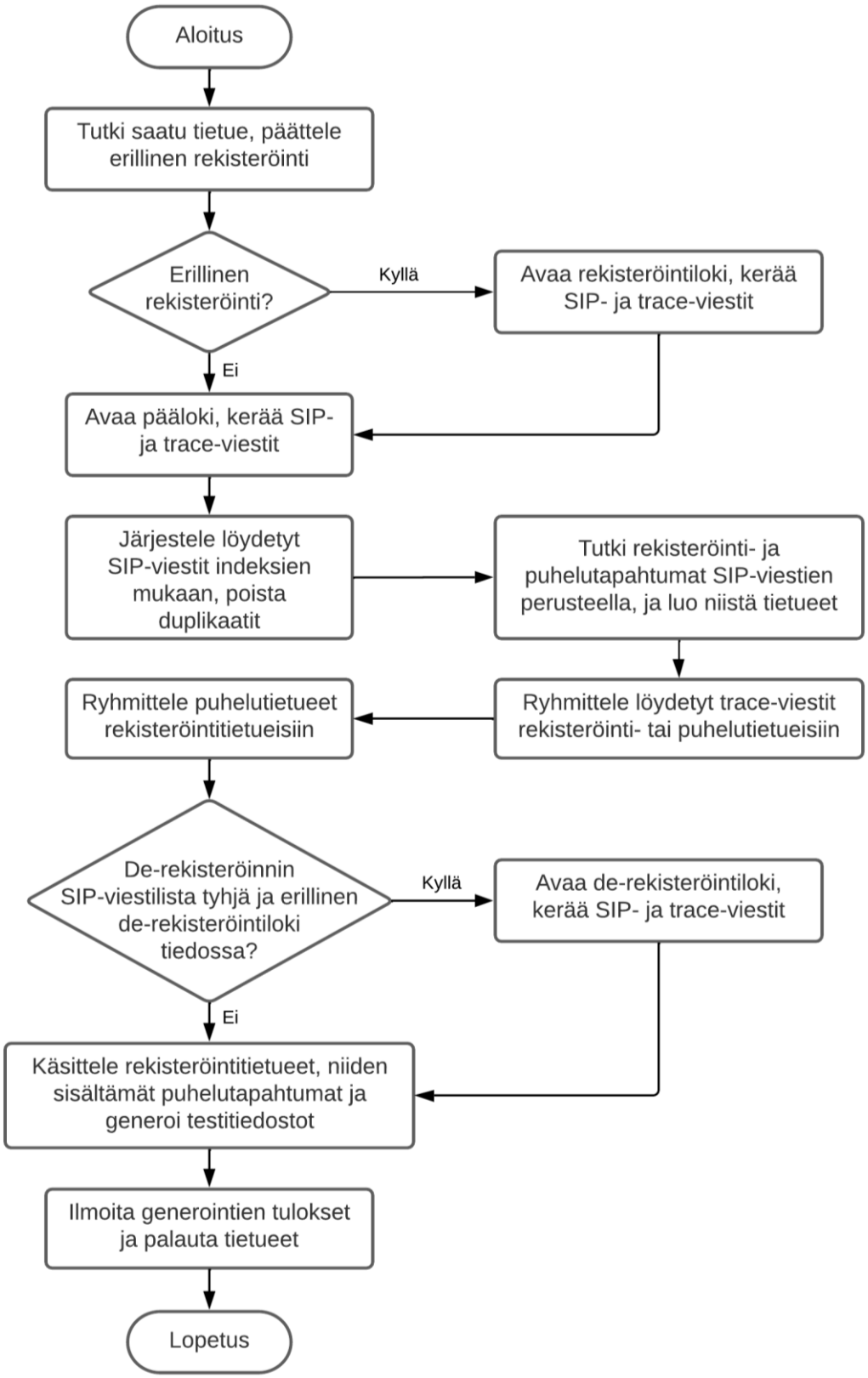
3.3 Modeemilokien analysointi ja testitiedoston kirjoitus

Modeemilokien analysointiin ja C-kielisten testitiedostojen kirjoittamiseen on kehitetty Python-skripti, joka MediaTekin kehittämää Python-kirjastoa hyödyntämällä osaa avata modeemilokitiedostoja. Lokitiedostot voivat olla kahdessa eri tiedostomuodossa. Skriptille annetaan komentoriviparametreina käsiteltävä hakemistopolku ja halutut tapahtumatyypit, kuten esimerkiksi puhelimen rekisteröityminen verkkoon, erilaiset peruspuhelut tai konferenssipuhelut. Vaihtoehtoisesti skripti voi saada nämä tiedot valmiissa tietueessa, jos skriptiä kutsuttiin toisesta skriptistä. Skriptin vuokaavio on esitetty kuvassa 6.

Skripti tutkii annetun tietueen ja jos se päättelee rekisteröintitapahtuman löytyvän erillisestä lokitiedostosta, se avataan ja siitä löytyvät SIP- ja trace-viestit tallennetaan muistiin. Trace-viestit sisältävät tietoa modeemin sisäisestä toiminnasta. Sama käsittely suoritetaan tietueessa määritellylle päälokille. Löydetyt SIP-viestit järjestellään indeksin mukaan pienimmästä suurimpaan ja niissä havaitut duplikaattiviestit poistetaan. Seuraavaksi kaikki löydetyt SIP-viestit käsitellään järjestyk-

sessä, jonka perusteella luodaan tietueet erillisistä rekisteröinti- ja puhelutapahtumista. Ryhmittelyvaiheessa kaikki trace-viestit ryhmitellään viestityypistä riippuen edellisessä vaiheessa luotuihin tietueisiin, jonka jälkeen vielä puhelutietueet ryhmitellään rekisteröintitietueisiin. Jos tässä vaiheessa rekisteröinnin poiston (de-rekisteröinti) viestilista on tyhjä ja tiedossa on erillinen de-rekisteröinnin sisältävä loki, poimitaan siitä vielä SIP- ja trace-viestit.

Lopuksi siirrytään testitiedostojen generointivaiheeseen, jossa jokainen rekisteröintitietue käsitellään toistolauseella. Niiden sisältämistä puhelutietueista tunnistetaan skriptin lähtötietueessa määritellyt halutut tapahtumat ja ainoastaan nämä tapahtumat ja niihin liittyvät tiedot kirjoitetaan lopulliseen generoitavaan testitiedostoon, joka on C-kielinen ohjelma.



KUVA 6. Modeemilokin käsittelyn ja testitiedostojen generoinnin vuokaavio

Yksittäisen testitiedoston generointi suoritetaan aliohjelmassa. Testitiedoston alkuun kirjoitetaan kaikki mukaan otettuihin tapahtumiin liittyvät SIP-viestit erillisinä char-tyyppisinä taulukkomuuttujina, joille skripti on päätellyt yksilölliset, moniosaiset nimet kunkin viestin tarkoituksen mukaan. Viestien sisältö vastaa pääosin modeemilokista luettujen viestien sisältöä, joitakin sisältökorvauksia ja pieniä muotoilumuutoksia lukuun ottamatta. Lisäksi viestin headers- ja body-osat on eroteltu omiin merkkijonomuuttujiinsa.

SIP-viestilistauksen jälkeen kirjoitettavaan aliohjelmaan sisällytetään testiajossa suoritettavat päätestisekvenssit, joissa jokin yksittäinen lokin tapahtuma simuloidaan (esimerkiksi verkkoon rekisteröityminen) ja joissa niille välitettyjä SIP-viestejä käytetään muun muassa viestien sisällön vertailemiseen testiajossa generoituvien, vastaavien viestien sisältöön. Testitiedoston loppuun kirjoitetaan pääohjelmakoodi, jossa asetetaan myös testiajossa käytettävät, modeemilokitiedoston sisäisistä viesteistä luetut konfiguraatioarvot. Ne voivat olla tyypiltään esimerkiksi merkkijonoja tai erityyppisiä lukuarvoja, joista osa muunnetaan skriptissä enum-vakiomuotoon.

Jokaisen testitiedoston generoinnin lopputulos tallennetaan skriptin tulostietueeseen. Jos testitiedoston generoinnissa tunnistetaan virhe, esimerkiksi tyhjä viestilista, palataan aliohjelmasta heti ulompaan funktioon, jossa virhe tallennetaan tulostietueeseen. Kaikkien generointien valmistuttua raportoidaan tulostietueen sisältämät lopputulokset, virheet, varoitukset ja muut ilmoitukset.

3.4 Modeemilokien tietojen yhdistely

Ihanteellisesti yksittäinen käsiteltävä modeemilokitiedosto sisältää kaikki testauksen kannalta vaaditut tapahtumat eli puhelimen rekisteröitymisen verkkoon, varsinaiset puhelutapahtumat ja puhelimen rekisteröinnin poistamisen verkosta. Yksittäisissä testeissä tuotetuista modeemilokitiedostoista puuttuvat kuitenkin usein puhelimen rekisteröitymiseen liittyvät tapahtumat, jotka ovat erillisiä testattavia tapahtumia ja joista on tallennettu erilliset modeemilokitiedostot. Koska eri maiden, verkkojen ja testiskenaarioiden yhdistelmiä on valtava määrä, lisäisi rekisteröitymisten sisällyttäminen kaikkiin testeihin merkittävästi tuotettujen lokitiedostojen kokoa ja testauksiin kuluvaa kokonaisaikaa.

Modeemilokeja analysoiva ja testitiedostot kirjoittava Python-skripti joutuukin usein yhdistelemään valikoidusti lokien sisältöä, mikäli yksittäinen loki ei sisällä kaikkia vaadittuja tapahtumia. Esimerkiksi erilaisten puhelutapahtumien onnistunut testaaminen edellyttää, että on löydetty puhelimen rekisteröityminen verkkoon, jotta siihen liittyvät viestit ja konfiguraatiotiedot voidaan analysoida lokista ja kirjoittaa lopulliseen testitiedostoon. Rekisteröitymisen puuttuessa puhelutapahtuman sisältävästä lokista pyritään lokitarjonnasta löytämään erillinen, mahdollisimman hyvin yhteensopiva lokitiedosto, joka sisältää rekisteröitymisen. Tällaisen lokitiedoston löytyessä skripti voi yhdistellä eri lokeista olennaiset tiedot ja kirjoittaa niiden avulla toimivan testitiedoston. Samalla se voi suorittaa lisäanalysointia ja varoittaa käyttäjää havaitsemistaan mahdollisista ongelmista, joita myöhemmin testiajossa saattaa ilmetä.

Koska yhdisteltävien lokien ajallinen ero voi olla tunteja tai jopa päiviä, saattavat lokeissa nähtävät puhelin- tai verkkokonfiguraatiot erota toisistaan osittain. Erot voivat olla joko yksittäisten koodilippujen arvoja tai suurempia kokonaisuuksia. Erityyppisten eroavaisuuksien kohdalla harkitaan tapauskohtaisesti, editoidaanko niitä skriptissä automaattisesti paremmin toisiaan vastaaviksi vai seuraako havaituista eroista vain virheilmoitus ilman muutosten tekemistä.

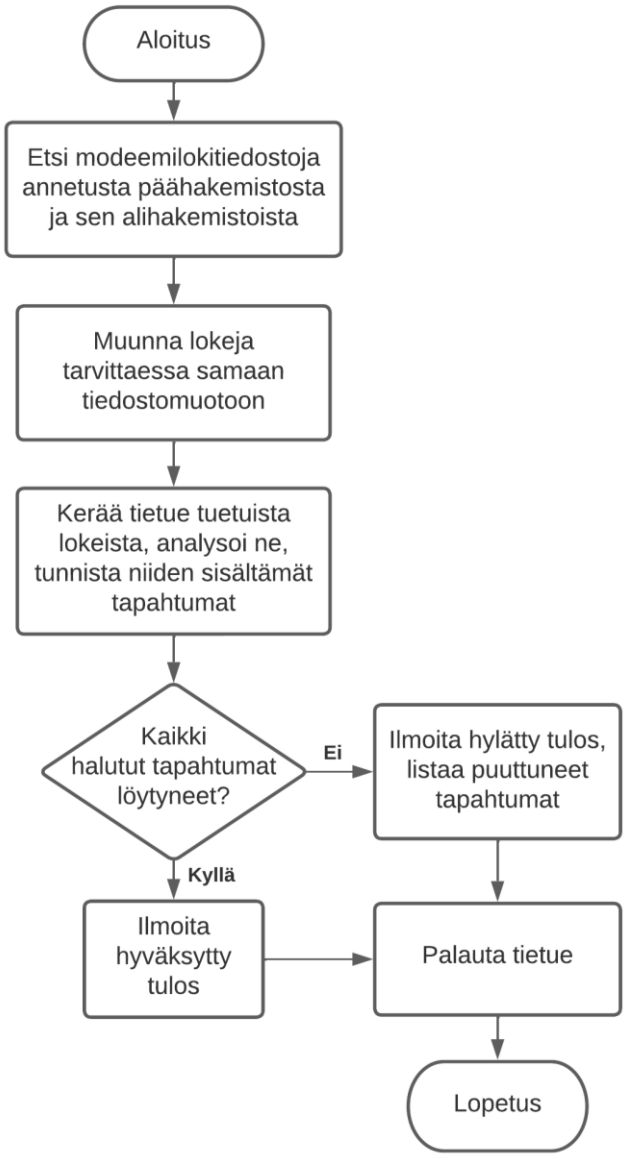
3.5 Modeemilokien ennakkotarkistus

Modeemilokit analysoidaan ja testitiedostot kirjoitetaan eri työalueella kuin missä testitiedostojen varsinainen testaaminen suoritetaan. Koska työalueiden välistä liikennettä pyritään minimoimaan tietoturvasyistä johtuen, ei testiajossa ilmenneistä virheistä ja ongelmista ole mahdollista välittää tietoa takaisin edellisen vaiheen työalueelle. Tämän vuoksi oli tarpeen kirjoittaa erillinen Python-skripti, joka tarkistaa analysoitavia lokeja ennakkoon vähintäänkin korkealla tasolla, jotta kirjoitetuilla testitiedostoilla olisi riittävän hyvät edellytykset läpäistä seuraavassa vaiheessa ajetut testit.

Ennakkotarkistusskriptiä voidaan kutsua funktiona toisesta skriptistä tai se voidaan suorittaa komentoriviltä. Molemmissa tapauksissa sille annetaan parametrina hakemistopolku, josta lokien etsintä aloitetaan. Skriptin koodissa polku välitetään funktiolle, joka tutkii toistolauseella yksittäisen hakemiston sisällön. Kohdatessaan hakemiston kohteissa alihakemistoja kutsuu funktio itseään rekursiivisesti, jolloin myös kaikki lähdehakemiston alikansiot tulee käytyä läpi. Samalla tallennetaan muistiin listaa löydetyistä modeemilokeista, joiden tiedostomuoto on tuettu.

Etsinnän tuloksena saadun listan lokit avataan ja niistä tunnistetaan puhelutapahtumat, verrataan löydettyjen ja haluttujen tapahtumien listaa keskenään, ilmoitetaan tutkinnan lopputulos ja palautetaan kerätty tietue, jos ulompi funktio kutsui skriptiä (kuva 7). Tarvittaessa löydettyjä lokeja muunnetaan samalla MediaTekin omaan lokitiedostomuotoon, mikä yhtenäistää niiden käsittelyä myöhemmissä vaiheissa. Olennaisin niistä tarkistettava tieto on puhelimen rekisteröitymistapahtuman löytyminen jostakin lokitiedostosta, koska rekisteröitymiseen liittyvistä SIP- ja muista sisäisistä viesteistä analysoidaan testausympäristössä myöhemmin asetettava puhelimen konfiguraatio testiajoa varten. Jos kaikki halutut puhelutapahtumat löydetään lokeista, tarkistus läpäistään hyväksytysti.

Modeemilokeista tarkistetaan myös muita tietoja ennakkoon. Useita lokitiedostoja yhdisteltäessä on hyödyllistä selvittää lokitiedoston tallentamisen päivämäärä ja käytetyn puhelinohjelmiston versio tai ohjelmistohaara, vertailla näiden tietojen vastaavuutta löydettyjen lokien välillä ja raportoida mahdollisista eroista. Mitä suurempia nämä erot ovat, sitä todennäköisemmin virheitä ja ongelmia esiintyy testiajossa tällaisten lokien tietoja yhdisteltäessä. Näiden erojen lisäksi ilmoitetaan yksilöllisten Call-ID-tunnisteiden lukumäärä, rekisteröinnin poiston löytyminen ja lukumäärälistaus valituista sisäisistä ja SIP-viesteistä, viestityypin tai SIP-metodin perusteella jaoteltuna.



KUVA 7. Ennakkotarkistusskriptin vuokaavio

4 TYÖN ETENEMINEN

Opinnäytetyön aihe oli jatkoa edellisissä, tutkintoon kuuluvissa yrityslähtöisissä tuotekehitysprojekteissa tehdylle kehitystyölle, jossa jo alustavasti toteutettiin muun muassa rekisteröinnin ja puhelutapahtuman yhdistämistä erillisistä modeemilokeista.

Työ alkoi määrittelemällä ja rajaamalla realistiset tavoitteet, jotka olisi mahdollista saavuttaa opinnäytetyölle varatussa ajassa. Ensimmäisissä vaiheissa selvitettiin, minkälaisia puhelutapahtumia tyypillisessä lokipaketissa oli löydettävissä, jotta tiedettäisiin, miten yleistä kaikkien vaadittujen tapahtumien löytyminen niistä on, vai onko lokipaketteja laajennettava liittämällä niihin tarpeen mukaan sisältöä muista testiskenaarioista.

Osastopäällikön ja projektikoordinaattorin kanssa sovittiin, että projektikoordinaattori toteuttaa automaatiossa Python-skriptejä edeltävän lokien keräilyvaiheen, joka automatisoidaan suoritettavaksi joka yö Jenkins-tehtävänä. Siinä Perl-kielinen skripti kerää alkuperäisistä kenttätestien tietokannoista olennaista dataa ja rajaa sitä eri kriteereillä, esimerkiksi päivämäärän mukaan. Jatkokäsittelyssä esimerkiksi poistetaan vielä turhia tiedostotyyppejä. Lopulta hyväksytyt tiedostot pakataan ja siirretään FTP-menetelmää käyttäen toiselle työalueelle, josta skripti lataa ja purkaa sen ja aloittaa sen käsittelemisen.

4.1 Lokipakettien sisällön analysointi ja ennakkotarkistusskripti

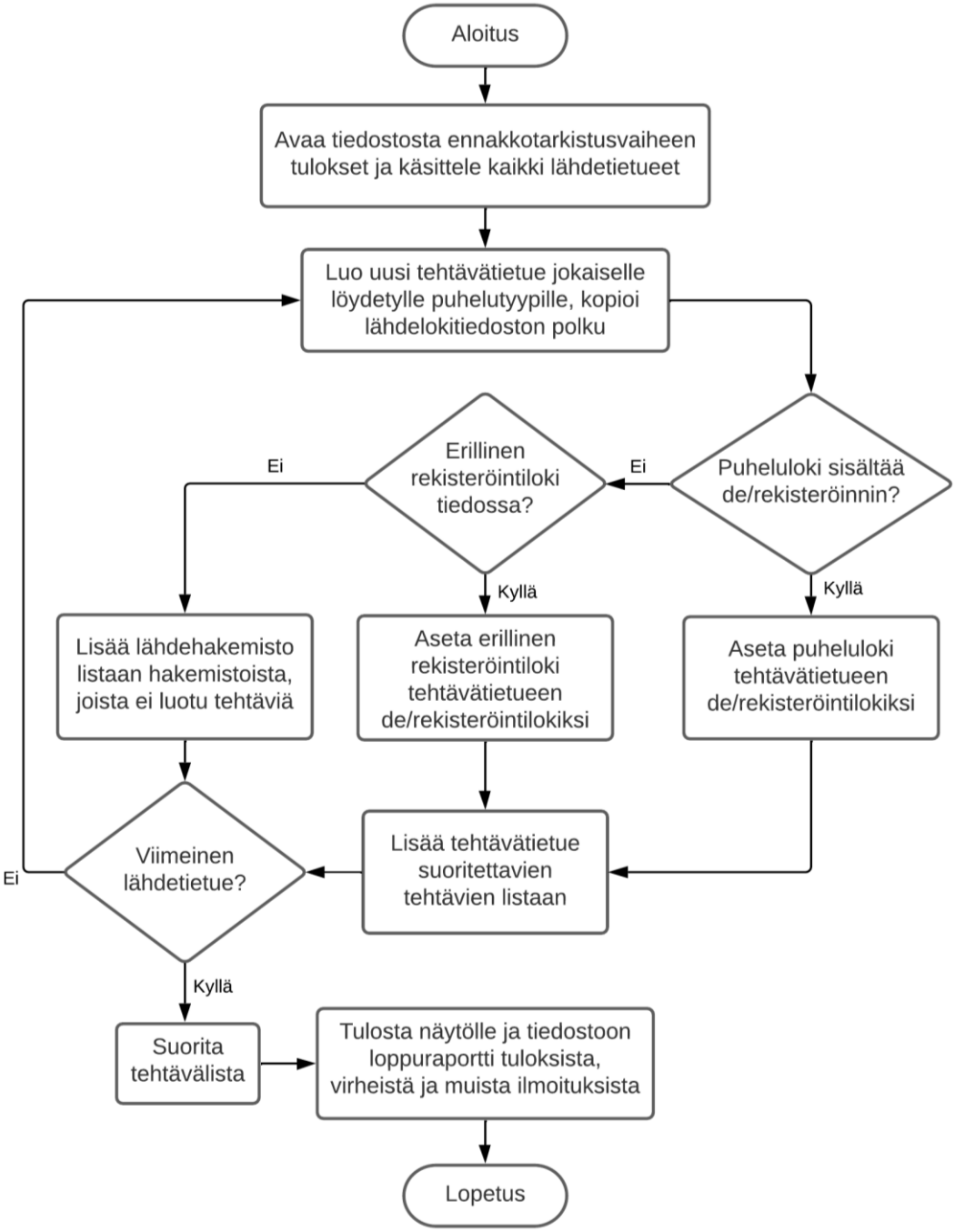
Eräajettava ennakkotarkastusskripti kävi läpi kaikki puretun lokipaketin sisältämät alihakemistot ja analysoi niistä löytyvien modeemilokitiedostojen sisältöä. Skripti kirjoitti analysoinnin tulokset CSV-tiedostoihin, joita voitiin avata ja jatkokäsitellä taulukkolaskentaohjelmassa (kuva 8). Ensimmäisissä lokipaketeissa oli lopullisten testitiedostojen generoimisen kannalta runsaasti ylimääräisiä tiedostoja, joten tutkimalla niiden hakemistorakennetta ja niiden sisältämiä tiedostotyyppejä pystyttiin niiden sisältöä optimoimaan merkittävästi tunnistamalla ja jättämällä pois tarpeettomat hakemistot ja tiedostotyypit.

Date	Campaign ID	Gen	Country	Operator	Test case	reg	dereg	mo_call	mt_call	confcall	Full path
08/02/2022	20033	98	Country1	Operator1	BC001	0	1	1	1	0	2022-02-08/20033/98/Country1/Operator1/BC001/20033.009#BC001#PASS#B_1
08/02/2022	20033	98	Country1	Operator2	BC001	1	1	1	1	0	2022-02-08/20033/98/Country1/Operator2/BC001/20033.001#BC001#PASS#B
08/02/2022	20033	98	Country1	Operator2	BC001	1	1	1	1	0	2022-02-08/20033/98/Country1/Operator2/BC001/20033.001#BC001#PASS#A
08/02/2022	20033	98	Country1	Operator4	BC001	1	1	1	1	0	2022-02-08/20033/98/Country1/Operator4/BC001/20033.017#BC001#PASS#A
08/02/2022	20033	98	Country1	Operator4	BC001	1	1	1	1	0	2022-02-08/20033/98/Country1/Operator4/BC001/20033.017#BC001#PASS#B
08/02/2022	20033	98	Country1	Operator3	BC001	1	1	1	1	0	2022-02-08/20033/98/Country1/Operator3/BC001/20033.025#BC001#PASS#B
08/02/2022	20033	98	Country1	Operator3	BC001	1	1	1	1	0	2022-02-08/20033/98/Country1/Operator3/BC001/20033.025#BC001#PASS#A
08/02/2022	20033	98	Country1	Operator5	BC001	1	1	1	1	0	2022-02-08/20033/98/Country1/Operator5/BC001/20033.033#BC001#PASS
08/02/2022	20033	98	Country1	Operator1	BC001	1	1	1	1	0	2022-02-08/20033/98/Country1/Operator1/BC001/20033.009#BC001#PASS#A
08/02/2022	20033	98	Country1	Operator1	CCall001	1	1	0	0	1	2022-02-08/20033/98/Country1/Operator1/CCall001/20033.009#CCall001#PASS#A_1
08/02/2022	20033	98	Country1	Operator1	CCall001	1	1	0	0	1	2022-02-08/20033/98/Country1/Operator1/CCall001/20033.009#CCall001#PASS#A
08/02/2022	20033	98	Country1	Operator1	CCall001	0	0	0	1	0	2022-02-08/20033/98/Country1/Operator1/CCall001/20033.009#CCall001#PASS#C
08/02/2022	20033	98	Country1	Operator1	CCall001	0	0	0	1	0	2022-02-08/20033/98/Country1/Operator1/CCall001/20033.009#CCall001#PASS#B
08/02/2022	20033	98	Country1	Operator1	CCall001	1	1	0	1	0	2022-02-08/20033/98/Country1/Operator1/CCall001/20033.009#CCall001#PASS#D
08/02/2022	20033	98	Country1	Operator3	CCall001	0	0	0	1	0	2022-02-08/20033/98/Country1/Operator3/CCall001/20033.025#CCall001#PASS#C
08/02/2022	20033	98	Country1	Operator3	CCall001	0	0	0	1	0	2022-02-08/20033/98/Country1/Operator3/CCall001/20033.025#CCall001#PASS#B
08/02/2022	20033	98	Country1	Operator3	CCall001	0	0	0	0	1	2022-02-08/20033/98/Country1/Operator3/CCall001/20033.025#CCall001#PASS#A
08/02/2022	20033	98	Country1	Operator3	CCall001	0	0	0	1	0	2022-02-08/20033/98/Country1/Operator3/CCall001/20033.025#CCall001#PASS#D

KUVA 8. Ennakkotarkistusskriptin eräajon tulokset. Maiden ja operaattorien nimet muutettu.

4.2 Generointiskripti: eräajo ja jatkokehitys

Kun lokipakettien sisältöä oli analysoitu riittävästi, siirryttiin testitiedostojen generoinnin eräajoskriptin kirjoittamiseen. Skriptin alussa luodaan tehtävälista lukemalla erillisestä tiedostosta ennakkotarkistusvaiheessa luodut tietueet, jotka käsitellään yksi kerrallaan (kuva 9).



KUVA 9. Generointitehtävälistan luonnin ja suorittamisen vuokaavio

Lokeista löytyneiden tapahtumien perusteella tietue voi sisältää rekisteröinnin, de-rekisteröinnin eli rekisteröinnin poiston sekä yhden kolmesta puhelutyypistä; lähtevän puhelun (MO), saapuvan puhelun (MT) tai konferenssipuhelun. Jokaiselle löydetylle, erityyppiselle puhelutapahtumalle luodaan aina erillinen tehtävä eli oma tehtävätietueensa, johon tallennetaan sen lähdelokitiedoston polku. Jos puhelutapahtuman loki sisältää lisäksi rekisteröinti- tai de-rekisteröintitapahtuman, merkitään se myös tietueen rekisteröinti- tai de-rekisteröintilokiksi. Muussa tapauksessa käytetään erillistä rekisteröinti- tai de-rekisteröintilokia, jos sellainen on tiedossa. Onnistuneesti luotu tehtävä sisältää

aina rekisteröintitapahtuman, jolloin se voidaan lisätä tehtävälistaan. Jos rekisteröintitapahtumaa ei ollut löydettävissä, lisätään lokitiedostojen yhteinen lähdehakemistopolku listaan hakemistopoluista, joista ei kyetty luomaan tehtäviä.

Kun lähdetietue on kokonaisuudessaan käsitelty, suoritetaan tehtävälista järjestyksessä. Onnistuneen yksittäisen tehtävän lopputuloksena on generoitu testitiedosto. Virhetilanteissa saatava virheilmoitus tallennetaan muistiin näytettäväksi eräajon tulosten loppuraportoinnissa. Loppuraportissa tulostetaan lisäksi listaukset suoritettujen tehtävien lopputuloksesta, huomioista, varoituksista sekä hakemistoista, joiden sisältämistä lokeista skripti ei onnistunut luomaan yhtäkään tehtävää (kuva 10).

```
Job  86: [PASS]   20321/country1/operator6
Job  87: [FAIL]   20321/country1/operator1      CANCEL-message detected after 1 completed call setups
Job  88: [PASS]   20321/country1/operator6

Batch operator test case generation complete after 36 minutes 17 seconds.

Notes:
20083/country4/operator5    dereg+reg+confcall    Registration #12: could not find deregistration response (200 OK) message
20151/country3/operator2    dereg+reg+mocall      Registration #0: de-registration message list is empty
20033/country1/operator4    dereg+reg+confcall    Registration #0: PANI-header change detected.
-> Original: 3GPP-E-UTRAN-FDD;utran-cell-id-3gpp=id1 (headers_registration_ue_register_initial_req), new: 3GPP-GERAN;cgi-3gpp=id2 (headers_deregistrat

Warnings:
20151/country3/operator2    dereg+reg+mocall      Call type is MO call, but "MO_CALL_IND" message list is empty.
20033/country1/operator1    dereg+reg+mocall      Dial number or SIP URI in first INVITE From-header does not match with any p_associated_uri seen in
-> "sip:+dial_number_1@ims.mnc010.mcc404.3gppnetwork.org" not in [sip:+dial_number_2@ims.mnc010.mcc404.3gppnetwork.org, tel:+dial_number_2]

Errors:
20033/country1/operator1    dereg+reg+mocall      Stopping after encountering "NAS_PS_REG_STATUS_REGISTERED_ROAMING" message in registration #0.
20033/country1/operator4    dereg+reg+mocall      [FAIL]   Missing requested scenarios: MO call
20151/country3/operator2    dereg+reg+mocall      Registration #0 was skipped. Reason: call message list is empty.
20151/country3/operator3    reg+mocall            Registration #0 was skipped. Reason: could not find registration 200 OK response message.
20321/country1/operator1    dereg+reg+confcall    [FAIL]   CANCEL-message detected after 1 completed call setups

Folders without jobs created:
20154/country1/operator1
```

KUVA 10. Generointiskriptin eräajon tulokset. Tunnistetiedot kuvassa muutettu.

Koska eräajoskriptin kehitystyössä käytettiin aina suuria, kymmeniä eri testitapauksia sisältäviä lokipaketteja, vei koko tehtävälistan suorittaminen aikaa vähintään puoli tuntia. Tämän vuoksi oli hyödyllistä laskea ja tulostaa jokaisen tehtävän alkaessa arvio eräajon jäljellä oleviin tehtäviin kuluvasta ajasta, joka laskettiin jo suoritettujen tehtävien keskimääräisen keston ja jäljellä olevien tehtävien lukumäärän perusteella. Arvio tarkentui aina jokaisen suoritetun tehtävän jälkeen. Mittausten perusteella keskimäärin noin kaksi kolmasosaa testitiedostojen generointiskriptin suorittamisen vaatimasta kokonaisajasta kuluu modeemilokitiedostojen avaamiseen skriptissä ja yksi kolmasosa skriptin muun koodin suorittamiseen. Automaation valmistuttua skriptien suoritusajan merkitys on kuitenkin hyvin vähäinen, koska automaatio suorittaa tehtävät öisin itsenäisesti ja yksittäinen, öisin suoritettava tehtävälista tulee sisältämään yleensä vain muutamia käsiteltäviä testiskenaarioita.

Työn edetessä havaittiin generoinnin lopputuloksissa yleiseksi ilmoitus de-rekisteröinnin puuttumisesta, mikä tarkoitti, ettei puhelulokista tai erillisestä rekisteröintilokista ollut löydettävissä de-rekisteröintitapahtumaa. Tämän vuoksi siirryttiin välillä myös de-rekisteröinnin yhdistämisen toteuttamiseen. Kuten erillisen rekisteröinnin ja puhelutapahtuman tapauksissa, saattoi puhelimen konfiguraatio olla muuttunut puhelutapahtuman ja de-rekisteröinnin välillä lokien ajallisista eroista johtuen. Tämä aiheutti jälleen virheitä testiajossa, joita korjattiin käyttämällä riittävästi aikaa eri ongelmatapausten tunnistamiseen ja analysointiin ja lopulta skriptin älykkyyden parantamiseen. Osa korjauksista pystyttiin ehtologiikalla rajaamaan koodissa tehtäväksi siten, että niitä suoritettiin vain tarpeen mukaan lokien yhdistelytapauksissa. Näin voitiin ehkäistä sitä, että tehdyt korjaukset mahdollisesti aiheuttaisivat uusia ongelmia toisaalla, mikä hidastaisi työssä etenemistä.

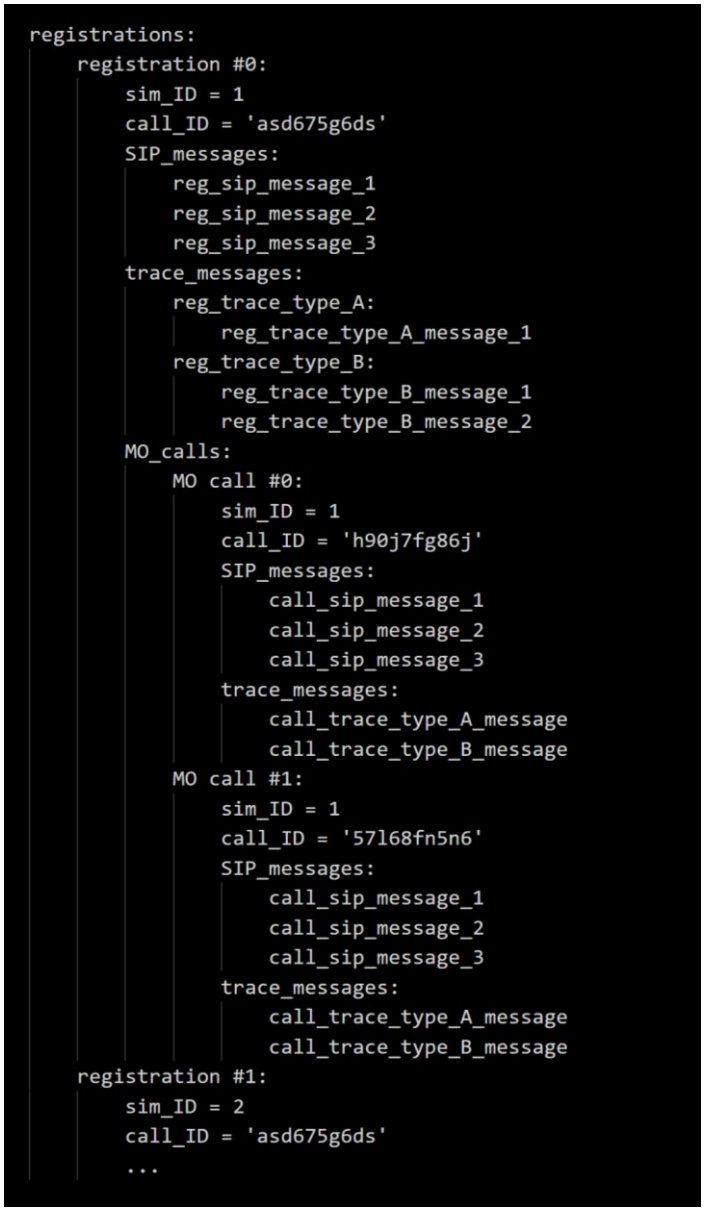
Viimeisin kehitystyössä käytettävä lokipaketti oli tiedostokooltaan noin 200 gigatavua, ja se sisälsi satoja modeemilokitiedostoja. Niiden joukossa oli lukuisia uusia, aiemmin käsittelemättömiä maa- ja operaattoriyhdistelmiä ja myös hieman erityyppisiä testiskenaarioita. Tämän vuoksi lokeja skriptin läpi ajettaessa ilmeni uusia ongelmia, joiden ratkaisemiseen oli käytettävä aiemmin arvioitua enemmän aikaa.

4.3 SIP- ja trace-viestien ryhmittelytoteutuksen uudistus

Koska SIP- ja sisäisten trace-viestien ryhmittelytoteutus osoittautui riittämättömäksi lukuisten lokeissa esiintyvien erilaisten poikkeustapausten huomioimisen vuoksi, käytettiin noin viikko sen uudistamiseen. Uudessa toteutuksessa suuri osa yksittäisiin puheluihin liittyvästä informaatiosta pyritään ryhmittelemään mahdollisimman varhain ja kattavasti suoraan puhelulle luotuun tietueeseen. Esimerkiksi puheluun liittyvät SIP- ja trace-viestit ryhmitellään puhelutietueeseen muiden sen sisältämien tietojen lisäksi. Eri viestityyppien ryhmittelyn jälkeen skripti etsii vielä jokaiselle puhelutietueelle rekisteröinnin, johon se kuuluu, ja liittää tietueen siihen.

Aikaisemmassa toteutuksessa esimerkiksi yksittäiseen puheluun kuuluvat erityyppiset trace-viestit liitettiin yleisemmin rekisteröintiin, joka oli aktiivisena kyseisen puhelun aikana. Tämän vuoksi puhelutapahtumia suodatettaessa täytyi selvittää tapauskohtaisesti, mitkä rekisteröinnin sisältämistä trace-viesteistä liittyivät juuri poistettuun puheluun, jotta osattaisiin poistaa niistä olennaiset viestit. Uusi toteutus selkeytti rekisteröintien ja puheluiden suodattamista ja muuta käsittelyä huomattavasti. Esimerkiksi jos puhelutietueita suodatettiin pois jollakin kriteerillä, poistuivat samalla myös

kaikki niihin liittyvät erityyppiset viestit. Vastaavasti rekisteröintejä suodatettaessa poistuivat myös kaikki niiden sisältämät puhelutapahtumat tietoineen. Uudistetussa toteutuksessa generoitua tietuetta rekisteröinneistä ja niiden sisältämistä puheluista on havainnollistettu kuvassa 11.



KUVA 11. Havainnekuva uudistetussa ryhmittelytoteutuksessa generoidusta tietueesta

4.4 Muuntotaulukoiden dynaaminen generointi

Testitiedostojen generointiskriptissä käytetään muuntotaulukoita, joiden avulla voidaan muuntaa tiettyjä lukuarvoja esimerkiksi merkkijonoiksi tai C-kielisen testausohjelman koodissa esiintyviksi enum-vakioiksi. Käsin kirjoitetut muuntotaulukot ovat ongelmallisia, koska ne vaativat skriptin kehittäjältä manuaalista ylläpitoa silloin, kun taulukossa esiintyvien arvojen vastaavuuksissa tapahtuu muutoksia testausohjelman lähdekoodissa tai muualla. Ongelma ratkaistiin kirjoittamalla toteutus, jolla suurin osa skriptin käyttämistä muuntotaulukoista voidaan generoida dynaamisesti eräajon alkuvaiheessa.

4.4.1 Makrojen ja enum-vakioiden muuntotaulukot

Makrojen ja enum-vakioiden muuntotaulukot generoidaan lähdekooditiedostoja parseroimalla. Yksittäisen muuntotaulukon generoivalle aliohjelmalle annetaan parametreina parseroitavan lähdekooditiedoston polku ja lista makroissa tai enum-vakioissa esiintyvistä, yhteisistä etuliitteistä tai avainsanoista. Aliohjelma lukee polun osoittaman tiedoston sisällön ja etsii siitä säännöllistä lauseketta (RegEx) käyttäen sen makrojoukon tai enum-määrittelylohkon, jossa kaikki annetut avainsanat esiintyvät.

Haettu määrittely jaetaan erillisiksi riveiksi rivinvaihtomerkin (’\n’) perusteella ja sitä jatkokäsitellään poistamalla mahdolliset kommenttirivit ja tyhjät rivit. Sen jälkeen jokaiselta riviltä tallennetaan muistiin makron tai enum-vakion nimi ja sille määritelty lukuarvo. Lukuarvoa voidaan joutua tulkitsemaan, koska se saattaa olla heksadesimaali- tai bittisiirtomuodossa. Lisäksi viimeksi luettu arvo tallennetaan muistiin ja sitä päivitetään siltä varalta, että jollakin rivillä on hyödynnetty enum-määrittelyn ominaisuutta, jonka mukaan uusi vakio, jolle ei ole erikseen määritelty arvoa, saa arvokseen edellisen vakion arvon kasvatettuna yhdellä.

Perustapauksissa enum-vakion arvo on selvitettävissä joko lukemalla se suoraan määrittelyriviltä tai päättelemällä se viimeksi tallennetusta arvosta. Vaativammissa tapauksissa enum-määrittelyjoukossa esiintyy yhdistelymakroja, joilla esimerkiksi yksittäinen määrittelyrivi on muodostettu. Tällöin on hyödyllistä ajaa parseroitava lähdekooditiedosto esikääntäjän läpi, joka kirjoittaa makrot

auki. Jos tämän jälkeen enum-vakion määrittelyarvona on toinen enum-vakio, on se yleensä kuitenkin löydettävissä samasta, esikäännetystä lähdekooditiedostosta RegEx-hakulauseketta käyttämällä.

Parseroinnin valmistuttua aliohjelman muistissa on vakionimi- ja lukuarvopareja. Niistä koostetaan kaksiulotteinen taulukkomuuttuja, jossa lukuarvot ovat ensimmäisessä sarakkeessa ja niitä vastaavat vakionimet toisessa sarakkeessa ja joka palautetaan aliohjelmaa kutsuneelle koodille.

4.4.2 ISO3166-maakoodin muuntotaulukko

ISO 3166 -standardin mukaiset, kaksikirjaimiset maakoodit voidaan selvittää MCC-numerokoodista (Mobile Country Code). MCC-koodia käytetään langattomissa puhelinverkoissa sen maan tunnistamiseen, johon verkon käyttäjä kuuluu. Käyttäjien tarkkaa yksilöimistä varten MCC-koodin kanssa käytetään lisäksi MNC-koodia (Mobile Network Code). (9.)

Maakoodien muuntotaulukon generoiva aliohjelma perustuu Python-skriptiin, joka on ladattavissa GitHubissa sijaitsevasta ohjelmavarastosta (10) ja käytettävissä MIT-lisenssillä. Skripti avaa websivun (9) ja parseroi sivulla julkaistun taulukon, joka sisältää erillisissä sarakkeissa MCC- ja MNC-koodit, maakoodit kaksikirjaimisessa ja numeromuodossa sekä maiden ja puhelinoperaattorien nimet selkokielisinä ja täysimittaisina.

Skriptin koodiin on tehty joitakin muokkauksia tarpeen mukaan, kuten ainoastaan MCC-koodien ja kaksikirjaimisten maakoodien rajaaminen ja tallentaminen parseroidun informaation joukosta. Niistä koostetaan kaksiulotteinen taulukkomuuttuja, joka palautetaan aliohjelmaa kutsuneelle koodille.

5 YHTEENVETO

Opinnäytetyön aiheena oli toteuttaa testausautomaatio, jossa päivittäin ladatuista modeemilokeista generoidaan operaattoritestitiedostoja testattavaksi ja raportoidaan saadut testitulokset eteenpäin tapauskohtaisille vastuuhenkilöille. Generointiskriptin jatkokehittämisen lisäksi työhön sisältyi lokien ennakkotarkistusskriptin kirjoittaminen, joka analysoi lokit ja varmistaa vaadittujen tapahtumien löytymisen niistä. Puuttuvia tapahtumia pyritään löytämään ja yhdistelemään toisista, yhteensopivista lokitiedostoista ennen testitiedostojen generointivaiheeseen siirtymistä.

Erillisiä, automaation eri vaiheisiin liittyviä toiminnallisuuksia saatiin toteutettua ja testattua kattavasti, ja suurin osa niistä on käytettävissä käyttäjän manuaalisilla toimilla. Automaation merkittävimmät osat ovat modeemilokitiedostojen ennakkotarkistuskripti ja operaattoritestitiedostojen generointiskripti. Kehitystyön myötä ne kykenevät yhdessä generoimaan suurestakin lokimassasta testitiedostoja, joista yli puolet läpäisee niillä ajetut testit hyväksytysti, ilman että käyttäjän tarvitsee tehdä niihin korjauksia käsin. Tämä on hyvä saavutus, sillä suuri lokimassa sisältää tyypillisesti lukuisia eri maa- ja operaattoriyhdistelmiä, joiden onnistunut käsittely vaatii generointiskriptiltä monipuolista älykkyyttä.

Vaikka täysautomaatiota ei lopulta kyetty saavuttamaan työn aloituspalaverissa asetetussa tavoiteajassa, on työn tilaaja kuitenkin tyytyväinen työn aikana saavutettuun edistymiseen, ja kehitystyötä jatketaan edelleen. Jatkokehityksessä automaation jäljellä olevat erilliset osat tullaan yhdistämään automaattisesti toimivaksi kokonaisuudeksi, jota kehitetään tarpeen mukaan. Automaation täysin valmistuttua ja toimiessa riittävän virheettömästi tullaan se ottamaan päivittäiseen käyttöön, jolloin se vähentää merkittävästi operaattoritestitiedostojen kirjoittamisen vaatimaa manuaalista työtä ja vapauttaa tältä osin henkilöstöresursseja muihin tehtäviin. Käytettävissä olevien ajantasaisten testitiedostojen lisääntynyt määrä luo myös kustannussäästöjä, sillä se mahdollistaa alustavan koodimuutosten verifioinnin nopeasti useammilla maa- ja operaattoriyhdistelmillä ennen siirtymistä todellisiin, yleensä matkustamista edellyttäviin kenttätesteihin.

LÄHTEET

1. MediaTek. About Mediatek. Hakupäivä 11.2.2022. https://i.mediatek.com/about-mediatek.

2. The Python Software Foundation 2022. The Python Tutorial. Hakupäivä 27.4.2022. https://docs.python.org/3/tutorial/index.html.

3. Wikipedia 2022. File Transfer Protocol. Hakupäivä 27.4.2022. https://en.wikipedia.org/w/index.php?title=File_Transfer_Protocol&oldid=1083263401.

4. Ubuntu.com 2022. Beginner/BashScripting. Hakupäivä 24.2.2022. https://help.ubuntu.com/community/Beginners/BashScripting.

5. Wikipedia 2022. Session Initiation Protocol. Hakupäivä 27.4.2022. https://en.wikipedia.org/w/index.php?title=Session_Initiation_Protocol&oldid=1075448157.

6. Poikselkä, Miikka & Holma, Harri & Hongisto, Jukka & Kallio, Juha & Toskala, Antti 2012. Voice over LTE (VoLTE). Chichester: John Wiley & Sons Ltd.

7. Wikipedia 2022. Session Description Protocol. Hakupäivä 27.4.2022. https://en.wikipedia.org/w/index.php?title=Session_Description_Protocol&oldid=1065556074.

8. Handley, Mark & Jacobson, Van & Perkins, Colin 2006. SDP: Session Description Protocol. IETF. Hakupäivä 27.4.2022. https://datatracker.ietf.org/doc/html/rfc4566.

9. Mcc-mnc.com country networks. Mobile Country Codes (MCC) and Mobile Network Codes (MNC). Hakupäivä 6.4.2022. https://www.mcc-mnc.com.

10. GitHub 2022. MCC-MNC-table. Hakupäivä 6.4.2022. https://github.com/musalbas/mcc-mnc-table.