TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Tutkintotyö

Jari Airaksinen

M-TEST-JÄRJESTELMÄ MATKAPUHELIMEN MULTIMEDIAN KÄYTTÖTAPAUSTESTAUKSESSA

Työn ohjaaja Lehtori Erkki Hietalahti
Työn teettäjä Nokia Oyj, valvojana DI Hannu Vuolle
Tampere 2006

TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikka
Ohjelmistotekniikka
Airaksinen, Jari

m-Test-järjestelmä matkapuhelimen multimedian käyttötapaustestauksessa

Tutkintotyö 28 sivua + 1 liitesivu
Työn ohjaaja Lehtori Erkki Hietalahti
Työn teettäjä Nokia Oyj, valvojana DI Hannu Vuolle
Helmikuu 2006
Hakusanat testaus, multimedia, käyttötapaus, automatisointi

TIIVISTELMÄ

Matkapuhelimissa on nykyään paljon erilaisia toimintoja ja erityisesti multimediaominaisuudet ovat tärkeitä. Erilaisia video- ja ääniyhdistelmiä on paljon, ja kun lisäksi matkapuhelimen rajoittuneet resurssit otetaan huomioon, voidaan todeta että toimintojen kattava testaus käsin ei ole järkevää. Käyttötapausten testaaminen on hyvä automatisoida, sillä testit suoritetaan käyttäjän näkökulmasta ja käytetään tarjottua graafista käyttöliittymää. Tämän työn tavoitteena oli dokumentoida multimedian käyttötapaustestauksessa käytetty testiympäristö, selvittää sen toimintaperiaate ja kertoa, kuinka se toimi suunnitellussa tehtävässään.

Testauksen automatisointiin valittiin yhdistelmäksi m-Test- ja QuickTest Pro -ohjelmat. Kohdelaite, jonka multimediaominaisuuksia testattiin, oli Nokian 6630 älypuhelin. Testijärjestelmä helpotti ja nopeutti suurten testiajojen suoritusta ja vapautti testaajia muihin tehtäviin. Suuria ongelmia ei järjestelmän käyttöönoton ja varsinaisen käytön aikana havaittu. Järjestelmän tulevaisuutta projektin päätyttyä ei varsinaisesti selvitetty, mutta sillä olisi varmasti käyttöä ja kehitysmahdollisuuksia myös vastedes.

TAMPERE POLYTECHNIC
Computer Systems Engineering
Software Engineering
Airaksinen, Jari m-Test system in mobile phone multimedia use case testing
Thesis 28 pages, 1 appendix
Thesis Supervisor lecturer Erkki Hietalahti
Commissioning Company Nokia Oyj. Supervisor: Hannu Vuolle (MSc)
February 2006
Keywords testing, multimedia, usecase, automation

ABSTRACT

Mobile phones are equipped with many different functions these days and the multimedia functions are especially important. Considering the many combinations of audio and video formats and the limited resources of mobile phones, manual testing is clearly not feasible. It is more practical to employ automated use case testing, which is done from the user's point of view with a graphical user interface. The aim of this thesis is to document the test environment used in multimedia use case testing, describe its working principle, and report how it fulfilled its designed purpose.

The programs selected for automated testing were m-Test and QuickTest Pro. The target device for multimedia functionality testing was the Nokia 6630 smartphone. The test system facilitated and expedited the running of large test sets, leaving the testers more time for other tasks. There were no real problems during the installation and the actual use of the system. The future use or development possibilities of the test system were not studied, but the system has potential for development and would likely continue to be useful.

ALKUSANAT

Tämä tutkintotyö on tehty Nokia Oyj:n teknologiayksikössä Tampereella. Työn aiheen sain kesällä 2004 aloitetusta testaus-projektista. Itse työ on kirjoitettu vuoden 2005 aikana.

Haluan kiittää työn ohjaajaa Erkki Hietalahtea, valvojaa ja tiimin vetäjää Hannu Vuolletta sekä kollegoitani kärsivällisyydestä työn valmistumista odottaessa.

Tampereella 8. helmikuuta 2006

Jari Airaksinen

SISÄLLYSLUETTELO

LYHENTEIDEN SELITYKSET

AAC	Häviöllinen äänen kompressointialgoritmi, erityisesti musiikille (Advanced Audio Coding)
AMR-NB	Ensisijaisesti puheäänen kompressoinnissa käytetty algoritmi (Adaptive Multi-Rate Narrowband)
AMR-WB	Laajempikaistainen versio AMR-algoritmista, voidaan käyttää myös musiikin kompressointiin (Adaptive Multi-Rate Wideband)
BT	Standardi laitteiden langattomaan kommunikointiin radioaalloilla (Bluetooth)
DSP	Digitaalinen signaaliprosessori
H.263	ITU-T-standardi digitaalisen videon pakkaukseen videokonfferenssia varten, käytetään myös tavalliseen videokuvaan
IP	Internet protokolla, Internetin verkkokerroksessa
LUA	Yleiskäyttöinen skriptikieli ohjelmien ohjelmalaajennuksien toteuttamiseen (http://www.lua.org/)
MP3	Häviöllinen äänen kompressointialgoritmi (MPEG-1 Audio Layer 3)
MPEG4	Videon kompressointialgoritmi (Moving Picture Experts Group kompressiostandardi versio 4)
MT	Päätelaite ottaa vastaan puhelun tai muun verkkotoiminteen (Mobile terminated)
PCM	Pulssikoodimodulaatio, perustapa ilmaista ääni-informaatio digitaalisessa muodossa (Pulse Code Modulation)
RA	RealNetworksin äänen kompressointialgoritmi (Real Audio)
RGB	Tapa esittää digitaalinen kuva pakkaamattomana, kuvan jokainen pikseli saa oman väriarvon (Red Green Blue)
RV	RealNetworksin videon kompressointialgoritmi (Real Video)
UIQ	Sony-Ericssonin kehittämä matkapuhelimien kosketusnäyttö-pohjainen käyttöliittymä
USB	Yleiskäyttöinen sarjaväylä oheislaitteiden liittämiseksi tietokoneeseen (Universal Serial Bus)

1 JOHDANTO

Testaus ja virheiden korjaaminen ovat ohjelmiston kehityskaaren tärkeimpiä osia ja niihin käytetäänkin yleensä runsaasti resursseja. Projektien aikataulut ovat kuitenkin usein tiukkoja, ja kun ohjelmistokehityksellä on tapana venyä, joudutaan testauksen määrässä tekemään kompromisseja. Tällöin testauksen tehokkuuteen kannattaa kiinnittää huomiota. Mitä aikaisemmassa vaiheessa ohjelmiston kehitystä virheet löytyvät, sitä halvemmaksi ja usein nopeammaksikin niiden korjaaminen tulee. Suurin osa virheistä löytyykin juuri alkuvaiheen vaatimuksista ja suunnittelusta /2, s. 10/. Tämän vuoksi testauksen suunnittelu ja toteutus tulisikin olla projekteissa mukana jo suunnitteluvaiheesta lähtien.

Sulautetut järjestelmät asettavat testaukselle omat vaatimuksensa. On otettava huomioon rajoittuneet resurssit, kuten muistin määrä ja suoritustehon puute. Mahdollisia käyttötapauksia saatetaan joutua usein jotenkin rajoittamaan tästä syystä. Myös erilaisia poikkeustilanteita syntyy sulautetuissa järjestelmissä helpommin kuin esimerkiksi PC-järjestelmissä, ja testauksessa on tutkittava huolellisesti myös näistä kaikista toipuminen.

Nykyaikaisissa älypuhelimissa multimediatoiminnot ovat hyvin merkittävä ominaisuus. Yhtälailla, kuten puhelutoiminnallisuuden, multimedian tulee toimia luotettavasti ja tehokkaasti. Kun mukaan tuodaan uusia tuettuja video- ja musiikkiformaatteja, kasvaa mahdollisten yhdistelmien lukumäärä hyvin nopeasti. Testattavaa syntyy paljon, kun halutaan varmistaa kaiken luotettava toiminta. Projektien tiukoissa aikatauluissa testauksen automatisointi on ainoa järkevä ratkaisu.

Käyttötapaustestaus, johon kategoriaan tämänkaltainen toiminnallisuuden testaus kuuluu, soveltuu hyvin automaation kohteeksi. Niin on siksi, että siinä kohdetta käsitellään korkeimmalta mahdolliselta tasolta, käyttöliittymästä. Älypuhelimen käyttöliittymän testaamiseen on olemassa lukuisia kaupallisia ohjelmistoja. Tässä projektissa työkaluiksi valittiin m-Test- ja QuickTest Pro -ohjelmat.

2 KÄYTTÖTAPAUSTESTAUS

Testaus ja virheiden etsintä (debugging) eivät ole sama asia, vaikka ne usein sekoitetaankin keskenään. Testaus pyrkii osoittamaan ohjelmistossa olevan virheitä, kun taas virheenmetsästyksessä pyritään löytämään virheen aiheuttaja ja korjaamaan se. /2, s. 11/

2.1 Yleistä testauksesta

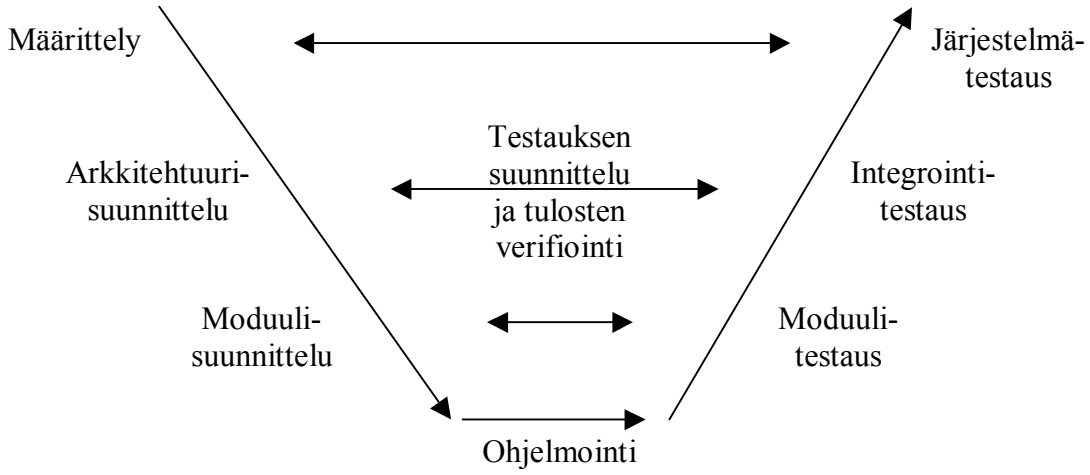
Testauksen suorittamiseen ja testitapausten valintaan on kaksi peruslähtökohtaa: lasilaatikkotestaus (glass / white box testing, structural testing) ja mustalaatikkotestaus (black box testing, functional testing). Lasilaatikkotestauksessa ideana on hyödyntää tietoa ohjelman varsinaisesta toteutuksesta valittaessa sopivia ja kattavia testejä. Mustalaatikkotestauksessa valinnat perustuvat ohjelman spesifikaatioon, eli varsinaista toteutusta ei tunneta. Tiedossa on vain se, miten ohjelman tulisi käyttäytyä tietyssä tilanteessa. Olipa kyseessä sitten kumpi testitapa tahansa, ei testitapauksia tule valita sattumanvaraisesti. Huomioon kannattaa ottaa ainakin virheiden löytymisen tehokkuus, kattavuus, käytön ja analysoinnin tehokkuus sekä ylläpidon helppous /2, s. 12/. Lasi- ja mustalaatikkotestauksen lisäksi on olemassa vielä kolmas lähestymistapa, harmaalaatikkotestaus (gray box). Tässä lähestymistavassa ohjelman spesifikaation lisäksi voidaan käyttää hyväksi tietoa ohjelman toteutusperiaatteista. Koodia ei tunneta, mutta saattaa olla tiedossa, että ohjelma käyttää jotain tiettyä algoritmia tai tietyn toiminnon toteutuksesta on olemassa karkea kuvaus.

2.2 Testauksen tasot

Ohjelmistojen testaus jaetaan ns. V-mallin (kuva 1) mukaisesti karkeasti kolmeen tasoon: moduuli-, integrointi- ja järjestelmätestaukseen. Testauksen jokaiseen tasoon kuuluu 4 työvaihetta: suunnittelu, testiympäristön luonti, testin suorittaminen ja tulosten tarkastelu. Testit on suunniteltava alkaen järjestelmätason testeistä yksittäisen komponentin testaukseen ennen kuin niitä ryhdytään suorittamaan.

Testien suoritusjärjestys on käänteinen suunnittelujärjestykseen nähden, eli ensin testataan yksittäiset komponentit ja edetään kohti järjestelmätason testejä. Aika-

taulusyistä eri tasojen testausta tehdään käytännössä ainakin hieman limittäin, eli esimerkiksi integrointitestaus alkaa jo ennen kuin moduulitestaus on saatettu loppuun. Testitapausmielessä moduulitason testit ovat lasilaatikkotestausta ja järjestelmätason testit yleensä mustalaatikkotestausta ja osittain harmaalaatikko-testausta.



Kuva 1 Testauksen V-malli /1, s. 271/

Moduulitestauksessa, jota kutsutaan myös yksikkö- tai komponenttitestaukseksi, testataan nimen mukaisesti yksittäisten pienten moduulien toiminnallisuus. Moduuli tai yksikkö käsitetään yleensä ohjelmiston pienimmäksi testattavissa olevaksi osaksi. Komponentti puolestaan voi olla myös isompi kokonaisuus, vaikka se usein liitetäänkin samaan seuraan moduulin ja yksikön kanssa./2, s.15/ Testauksen suorittaa yleensä moduulin tekijä osittain koodauksen aikana, ja lopullinen testi tehdään ennen kuin moduuli julkaistaan eteenpäin integrointiin. Moduulitestauksen tulisikin olla kiinteä osa ohjelmakoodin tekoa, sillä tämän tason virheiden korjaukset tulevat olemaan kalliita myöhemmissä vaiheissa ja virheet saattavat olla hyvin hankalasti jäljitettävissä.

Ohjelmiston integroinnissa moduuleita ja niiden kokonaisuuksia yhdistetään entistä suuremmiksi kokonaisuuksiksi. Integrointitestauksessa ei testata enää yksittäistä toiminnallisuutta vaan moduulien välisiä rajapintoja ja niiden yhteensopivuutta. Suurehkoissa projekteissa integroinnin ja integrointitestauksen suorittaa yleensä

oma tiimi, jolla on aikaa paneutua testisuunnitelmiin, -tarpeisiin ja testien tekemiseen. Ohjelmoijat ovat yleensä kiinni ohjelman ja komponenttien jatkokehityksessä ja virheiden korjauksessa, joita integrointitesteissä yleensä löytyy.

Järjestelmätestauksen kohteena on kokonainen integroitu ohjelmisto. Yleensä testaus suoritetaan lopullisessa kohdejärjestelmässä, jossa sitä tullaan käyttämään. Testauksessa on tarkoituksena verrata ohjelmiston toimintaa asetettuihin vaatimuksiin ja toiminnalliseen määrittelyyn. Näkökulman tulisi olla mahdollisimman asiakaslähtöinen, eli testaajilla ei saisi olla liian syvällistä tuntemusta järjestelmän toteutuksesta ja arkkitehtuurista. Järjestelmätestaus jaetaan yleensä kahteen osaan, funktionaaliseen ja ei-funktionaaliseen testaukseen. Funktionaalisen testauksen tarkoituksena on selvittää, että tuote toimii toiminnallisen määrittelyn mukaisesti. Käyttötapaustestaus on tärkein osa funktionaalista testausta. Ei-funktionaaliset testit koostuvat mm. kuormitustestauksesta, tietoturvatestauksesta ja käytettävyystestauksesta.

2.3 Käyttötapaukset

Käyttötapauksia pidetään useimmiten osana ohjelmiston kehitysmenetelmiä ja 90-luvulla niistä tulikin yleinen menetelmä toiminnallisten vaatimusten kartoittamiseksi. Käyttötapaukset ovat toimintoketjuja, mielekkäiden tehtäväkokonaisuuksien suorituksia, jotka syntyvät käyttäjän toimista järjestelmässä. Tarkoituksena on siis kuvata, kuinka järjestelmän tulisi toimia käyttäjän ja/tai toisen järjestelmän kanssa. Käyttötapausten avulla järjestelmän ohjekirja ikään kuin kirjoitetaan ennen varsinaisen teknisen suunnittelun aloitusta /1, s. 141-143; 3/.

Käyttötapaukset kuvataan yleensä kirjallisesti, ja yksi tapaus keskittyy yhden tavoitteen saavuttamiseen. Tekstimuotoisen kuvauksen lisäksi voidaan käyttää apuna muita kuvaustekniikoita, kuten UML-kuvauskieltä tai tapahtumasekvenssikaavioita. Käyttötapauksen kuvauksen tulisi olla mahdollisimman konkreettinen eikä sen tulisi ottaa kantaa varsinaiseen tulevaan toteutukseen. Käyttötapausten suunnitteluvaiheessa voidaan pitää yllä erillistä dokumentaatiota havaituista

toteutusta koskevista asioista. Nämä merkinnät otetaan myöhemmin järjestelmän suunnittelussa huomioon. /1, s. 142-143/

Käyttötapausten ei tarvitse kattaa koko järjestelmän toimintoja, sillä ne eivät korvaa toiminnallisen määrittelyn kuvauksia. Usein kuitenkin käyttötapauksilla pyritään kattamaan mahdollisimman suuri osuus toiminnallisuuksista. Tämä tarkoittaa, että käyttötapauskuvauksia tullaan tarvitsemaan useita kymmeniä ellei jopa satoja pienissäkin järjestelmissä.

2.4 Multimedia käyttötapaukset

Multimedia käsittää tässä tapauksessa äänen ja videokuvan toistamisen. Testeissä käytetty älypuhelin tukee seuraavia ääniformaatteja: PCM, MP3, AAC, AMR-NB, AMR-WB, RealAudio8 ja RealAudioVoice. Videoformaateista tuettuja ovat H.263, MPEG4 ja RealVideo8. H.263- ja MPEG4-muotoisen videon ääniraidan formaattina voi olla AAC, AMR-NB tai AMR-WB. RealVideo8-videon ääniraita on joko RealAudio8 tai RealAudioVoice. MP3 ja PCM on pelkästään audio-kappaleiden soittoon ja puhelimen soittoääniksi tarkoitettu. Taulukossa 1 on esitetty vielä havainnollisemmin ääni- ja videoformaatit sekä niiden keskinäiset yhdistelmät.

Taulukko 1 Tuetut ääni- ja videoformaatit sekä videoiden tuetut ääniraidat

Ääni / Video	PCM	MP3	AAC	RA8	RAV	AMR-NB	AMR-WB
H.263			X			X	X
MPEG4			X			X	X
RV8				X	X		

Kaikki ääni- ja videotoiminnot ovat DSP-kiihdytettyjä eli puhelimessa on digitaalinen signaaliprosessori, joka hoitaa ääni- ja videodatan dekoodaamisen. Koodekit ovat C- ja assembler-kielillä toteutettuja ohjelmia, jotka tarpeen mukaan dynaamisesti ladataan DSP:lle käyttöön. Koodekin tehtävä on purkaa ääni- tai videodata sellaiseen muotoon, jonka kohdelaite osaa esittää. Äänen osalta tämä on

PCM-formaatti ja videokuvan osalta RGB-data. Tietoa kompressoidaan, jotta sitä pystyttäisiin tallentamaan enemmän pieneen tilaan ja jotta sen siirtäminen tietoverkkojen yli sujuisi entistä nopeammin.

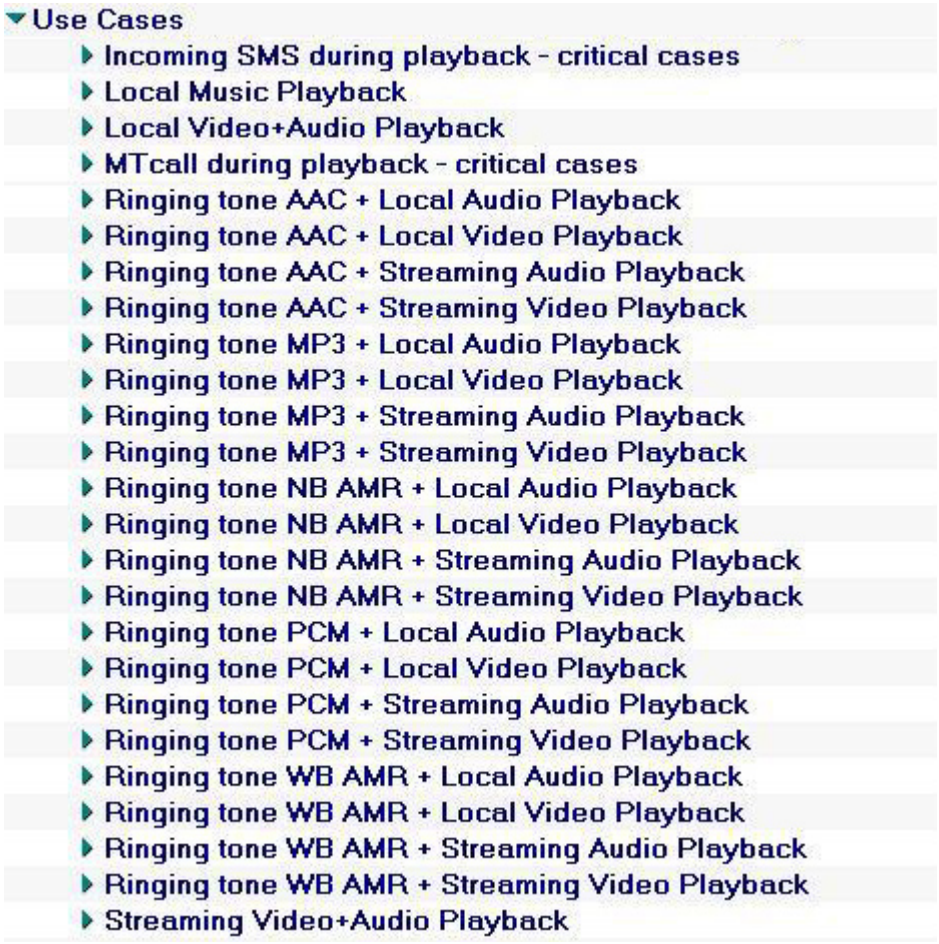
Käytännön suorituskyky- ja muistirajoitteiden vuoksi eri koodekkiyhdistelmiä joudutaan rajoittamaan. Esimerkki käytännön rajoitteesta on, että vain kaksi audiokoodekkia voi olla yhtä aikaa käytössä ja niiden lisäksi yksi video-koodekki. Tämä vastaa tilannetta, jossa kesken videotiedoston soiton tulee puhelu, jota varten tarvitaan soittoääni. Eri video- ja audiokoodekit kuluttavat eritavalla DSP:n rajallisia muisti- ja suoritinresursseja. Resurssien kulutus saattaa myös vaihdella käytetyn mediatiedoston mukaan, esimerkiksi paljon liikettä sisältävä videotiedosto voi vaatia algoritmilta enemmän suorituskykyä kuin vähän liikettä sisältävä. Tällöin täytyy joko testata usealla eri tiedostolla tai määritellä ”worst case”-tiedosto, jota käytettäessä koodekin resurssien kulutus on huipussaan.

Multimediakäyttötapaukset jaetaan kahteen ryhmään: staattiset käyttötapaukset ja staattiset käyttötapaukset satunnaisilla tapahtumilla. Staattisissa käyttötapauksissa musiikkia tai videota toistetaan häiriöttä alusta loppuun. Tiedoston toiston aikana puhelu, tekstiviesti, kalenterihälytys tai muu järjestelmän osa voi aiheuttaa tarpeen toiselle audiolähteelle soittoäänenä tai muuna huomioäänenä. Näitä käyttötapauksia kutsutaan staattisiksi käyttötapauksiksi satunnaisilla tapahtumilla.

Kuvassa 2 on ruutukaappaus määritellyistä käyttötapaustesteistä projektin testitietokannassa. Testit on ryhmitelty toisiinsa liittyviksi kokonaisuuksiksi käyttötapausluonteensa mukaisesti. Esimerkiksi ”Local Music Playback” pitää sisällään testin, jossa paikallisesta levyjärjestelmästä toistetaan yksi tiedosto kutakin formaattia. Vastaavasti ”Local Video+Audio Playback” -testissä käydään läpi kaikki mahdolliset ääni- ja videoformaattiyhdistelmät. Paikallisesta levyjärjestelmästä toiston lisäksi testataan äänen ja videon suoratoisto verkon yli. Nämä local- ja streaming-testit yhdessä muodostavat staattisen testauksen.

Kun tiedoston soiton päälle lisätään aiemmin mainittu satunnainen toiminto, kuten esimerkiksi puhelun saapuminen, syntyy suuri joukko uusia käyttötapauksia. Viisi

eri audioformaattia voi olla asetettuna soittoääneksi tai muuksi huomioääneksi: AAC, MP3, PCM, AMR-NB ja AMR-WB. RealNetworksin audiokoodekit on rajattu huomio- ja soittoäänien valikomasta pois. Kullekin mahdolliselle huomioääniformaatille määriteltiin neljä testiä, joissa ääni tulee paikallisesti toistetun sekä verkon yli suoratoistetun videon ja audion päälle. Kussakin testissä käydään läpi kaikki mahdolliset yhdistelmät.



Kuva 2 Multimedia käyttötapaukset

Soitettavat tiedostot on valittu ”worst case” -periaatteen mukaisesti. Videoiden ja audion sisältö on siis sellainen, joka muistin ja suoritintehon kulutuksen kannalta rasittaa järjestelmää eniten. Käyttötapaustesteissä on riittävää olettaa, että mikäli kyseinen tiedosto soi ja toimii moitteettomasti, myös vähemmän vaativat tiedostot toimivat. Niin on siksi, että dekooderin toimittaja on testannut toimivuuden ja standardien mukaisuuden omissa testeissään jo aiemmin. Näissä kutakin dekooderia testataan laajasti erilaisilla sisällöillä, joista sen tulee suoriutua.

Rajoituksista huolimatta testattavia yhdistelmiä syntyy kuitenkin niin paljon, että puhtaasti käsin kaikkien läpikäymiseen kuluu liikaa aikaa. Automatisoinnille on siis selkeä tarve. Kuitenkin myös automatisoituna koko testikokoelman läpikäyminen vie päiviä. Sen vuoksi oli tarve määritellä alkuperäistä suppeampi testijoukko, josta saataisiin tuloksia huomattavasti lyhyemmässä ajassa. Kyseisen ”Critical cases” -testin sisältöä vaihdettiin sen mukaan, mitä erityisesti oli tarve testata, esimerkiksi saatujen virheen korjausten tai muuttuneen toiminnallisuuden perusteella. Se toimi myös samalla regressiotestauksena.

3 TESTAUKSEN AUTOMATISOINTI

Testauksen automatisoinnilla voi olla erilaisia tavoitteita. Voidaan pyrkiä yksittäisen testin entistä nopeampaan suoritukseen tai suorittamaan entistä enemmän testejä manuaaliseen testaukseen ennen käytetyssä ajassa. Tärkein tavoite kuitenkin on ohjelmiston laadun parantaminen. Kaikkea ei kuitenkaan ole järkevää automatisoida, sillä automaattisten testien suunnittelu ja toteutus vaatii aikaa ja osaavien henkilöiden resursseja. Mikäli on todennäköistä, että jokin testi suoritetaan korkeintaan muutaman kerran, niin testin automatisointiin kuluisi enemmän aikaa kuin sen muutamaan manuaaliseen suoritukseen. Mitä useammin testi tullaan suorittamaan, sitä kannattavammaksi sen automatisointi tulee. /2, s.18/

3.1 Automatisoinnin etuja

Automatisoidut testit ovat parhaimmillaan korkeilla testaustasoilla, kuten regressiotesteissä, systeemitesteissä ja kuormitustesteissä, ylipäätään silloin, kun testi tarvitsee suorittaa usein tai siinä tarvitaan pitkiä toistosarjoja. Manuaalisessa testauksessa testaaja tekee virheitä väsymisen ja turhautumisen vuoksi. Automatisoitu testi on aina samanlainen. Kun testi on kerran automatisoitu, voidaan se ajaa aina tarvittaessa uudelleen lukemattomia kertoja peräjälkeen. Testi voidaan jättää pyörimään työpäivän tai -viikon päätteeksi ja tulokset tarkistaa seuraavan kerran töihin saavuttaessa. Kaikkea ei voi testata järkevällä tavalla manuaalisesti, esimerkiksi järjestelmiä, joilla voi olla useita satoja yhtäaikaisia käyttäjiä.

Automaation keinoin voidaan helposti simuloida suuria käyttäjämääriä ja kuormittaa järjestelmää muilla tavoin. /2, s.19-20; 5, s. 28-29/

3.2 Automatisoinnin ongelmia ja rajoitteita

Testauksen automatisointi ei ole kaikkivoipa keino. Siihen liittyy jopa epärealistisia odotuksia, ja sen odotetaan ratkaisevan testauksen ongelmat. Onnistuneen automaation avainsana on suunnittelu. Automaattinen testi löytää vain ne virheet, joita se on suunniteltu etsimään. Mikäli testi on huonosti suunniteltu, saattaa se löytää virheitä vähemmän kuin manuaalisissa testeissä olisi löytynyt. Automatisointi vaatii sekä ohjelmointi- että testaustaitoja ja järjestelmätuntemusta. Lisäksi testattavan ohjelmiston muuttumisesta johtuva testien ylläpito kuluttaa resursseja. Näiden resurssien löytäminen voi olla ongelma. /2, s.18-19; 5, s.29-30/

Automaattisten testien ominaispiirre on se, että ne menevät jossain vaiheessa rikki. Ihminen voi testiä suorittaessaan mukautua testattavan ohjelman muuttuneisiin piirteisiin, kuten käyttöliittymään. Automaattisilla testeillä on tämänkaltaisia ominaisuuksia hyvin harvoin ja silloinkin vain hyvin rajoittuneesti. Automaattiset testit eivät koskaan voi korvata manuaalista testausta kokonaan.

3.3 Milloin automatisoida?

Ei ole välttämättä helppoa määritellä etukäteen, koska ja mitkä testit kannattaa automatisoida. Seuraavat kolme kysymystä voivat auttaa päätöksenteossa:

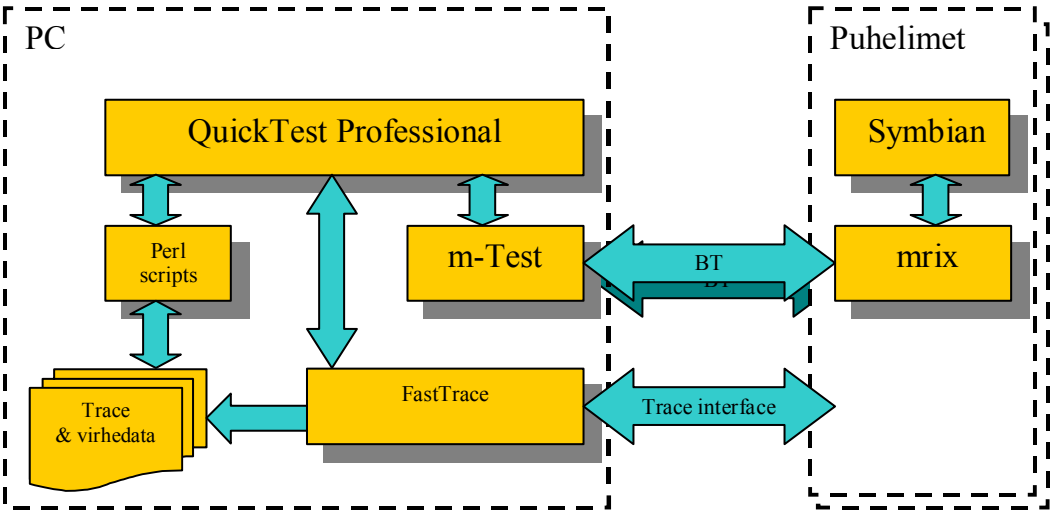
1. Kuinka paljon enemmän maksaa testin automatisointi ja sen suoritus kerran verrattuna yhden manuaalisen suorituksen hintaan?
2. Automatisoidulla testillä on äärellinen elinaika, jonka aikana sen tulee kattaa nämä ylimääräiset kustannukset. Kuinka todennäköisesti tämä testi tulee menemään rikki ennemmin tai myöhemmin?
3. Kuinka todennäköisesti testi löytää elinaikanansa muita virheitä kuin ensimmäisellä ajokerralla löytyneet?

Näistä kysymyksistä viimeisin on oleellisin. Testaajilla ei kuitenkaan yleensä ole riittävän laaja-alaista näkemystä järjestelmästä, jotta he osaisivat vastata näihin kysymyksiin. /4, s.2/

Jotta automaatiota kannattaa lähteä edes harkitsemaan, täytyy perustestauksella olla ensin riittävän vakaa pohja. Testien ja testauksen suunnittelu on prosessi, jonka tulisi kulkea käsi kädessä tuotteen suunnittelun kanssa. Aikaisessa suunnittelu-vaiheessa ei automaatiolle ole juuri käyttöä, sillä koodin muutosvauhti on liian suuri ja todennäköisyys testien rikkoutumiselle olisi suuri. Vasta kun ohjelma-koodi on kokonaisuudessaan kirjoitettu, eli projekti saavuttaa ”Code complete” -vaiheen, päästään automaation etuja hyödyntämään kunnolla. Tällöin testien painopiste on jo ehtinyt kallistua entistä laajempiin systeemitesteihin. Toki regressio-testauksessa automaatiota voidaan hyödyntää jo aiemmin, varsinkin niissä komponenteissa, joiden kehitysaste on riittävän pitkällä ja joissa ei ole suuria muutoksia tiedossa. /6/

4 TESTIYMPÄRISTÖ

Multimedia käyttötapausten testaukseen rakennetun testiympäristön yleinen arkkitehtuuri on esitetty kuvassa 3. PC:n puolella se koostuu kolmesta ohjelmasta: Mercury Interactiven QuickTest Professional, Intuwaven m-Test ja Nokian FastTrace. Lisäksi käytössä on muutamia perl-skriptejä trace- ja virhedata-tiedostojen analysointia varten. Kohdejärjestelmää eli älypuhelinta ohjataan bluetooth-yhteyden yli QuickTest Professional ja m-Test-ohjelmien avulla. Matkapuhelimessa oleva m-Testiin kuuluva mrix-komponentti ottaa vastaan komennot ja ohjaa niiden mukaisesti älypuhelimen käyttöliittymää ja ohjelmia. FastTrace on Nokian sisäinen ohjelma ja se tallentaa älypuhelimen trace-liitynnän kautta ajonaikaista debuggaus- ja virheenjäljitysdataa tiedostoon. Perl-skriptien tarkoituksena on muun muassa poimia tästä datasta mahdolliset virhetilanteet esille. Puhelimia testijärjestelmässä on kaksi kappaletta. Toinen on varsinainen testipuhelin, joka sisältää testattavan ohjelmiston. Toinen puhelin toimii apupuhelimena, jolla ensisijaiseen puhelimeen voidaan soittaa kesken testin suorituksen.



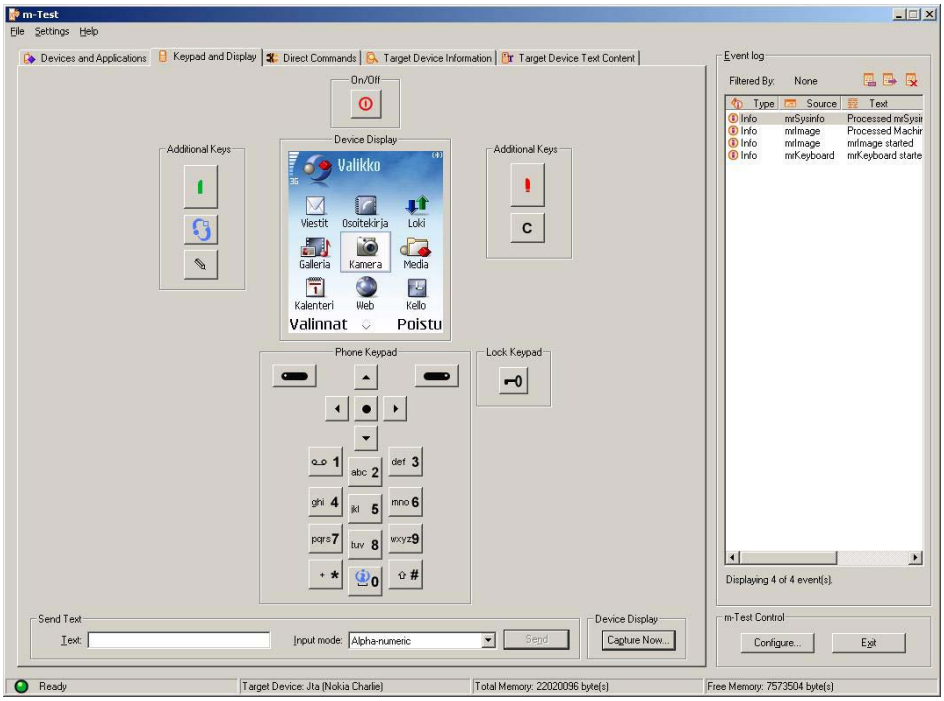
Kuva 3 Testijärjestelmän yleinen arkkitehtuuri.

4.1 m-Test ja mrix

m-Test on brittiläisen Intuwave nimisen yrityksen kehittämä windowsohjelma, joka mahdollistaa Symbian pohjaisten älypuhelinten ohjaamisen ja testaamisen PC:ltä käsin. Se ei siis ole simulaattori tai emulaattori, vaan se ottaa suoran yhteyden älypuhelimeen joko bluetooth-, infrapuna-, USB- tai sarjakaapelilinkin kautta. Käyttämällä bluetoothia, voi m-Test muodostaa yhteyden useaan puhelimeen. Kuitenkin vain yhtä puhelinta voi ohjata kerrallaan, muut yhteydet odottavat aktiivisina taustalla. Ohjauksen kohteena olevan puhelimen voi vaihtaa m-Testin käyttöliittymästä.

Käyttöliittymässään (kuva 4) m-Test tarjoaa sillä hetkellä aktiivisessa ohjauksessa olevan puhelimen näytön kuvan sekä näppäimistön painikkeet. Kun käyttöliittymän painiketta painetaan tai annetaan jokin muu komento, m-Test joko suorittaa kyseisen toiminnon heti itse tai lähettää sen eteenpäin kohdelaitteelle. Komennot, jotka eivät muuta kohdelaitteen tilaa voi m-Test suorittaa itse heti. Esimerkiksi bittikarttavertailut ja tekstintunnistus voidaan tehdä heti, sillä vertailun pohjana oleva kohdelaitteen näyttö on jo tiedossa. Kohdelaitteelle lähtevät komennot ovat tyypillisesti näppäinten painalluksia. Joitakin toimintoja on laajennettu, kuten tekstin syötössä ”Send text” -komento, jonka avulla kokonaisen tekstijonon tai puhelinnumeron voi lähettää helposti kohdelaitteeseen yhdellä käskyllä. Toinen

hyödyllinen ominaisuus on puhelimessa olevien ohjelmien käynnistäminen ilman valikoissa selaamista.

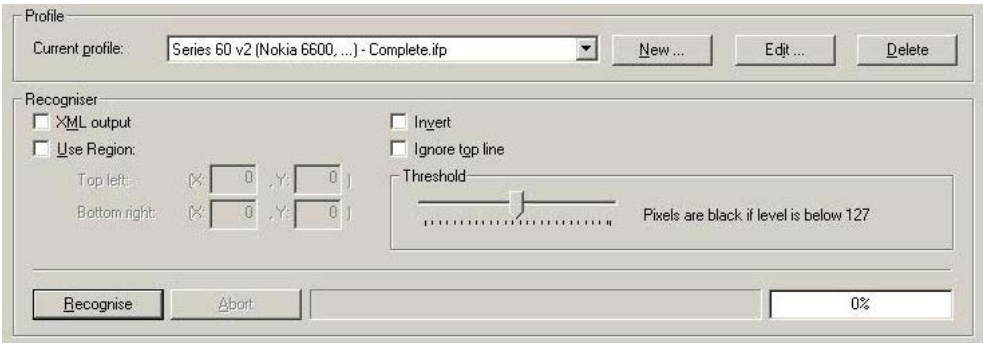


Kuva 4 M-Test-ohjelman käyttöliittymää

Kuvassa 4 näkyvä käyttöliittymän välilehti on testauksen aikana eniten käytetty, sillä sen avulla saa suoritettua suurimman osan puhelimen ohjauksesta. Samalta välilehdeltä löytyy myös mahdollisuus tekstin lähettämiseen kohdelaitteeseen yhtenä merkkijonona yksittäisten käyttöliittymän näppäinten painamisen sijaan. Ensimmäiseltä välilehdeltä löytyvät toiminnot bluetooth-yhteyden kautta ohjattavan puhelimen vaihtoon. Sieltä on myös listaus puhelimesta löytyvistä ohjelmista, jotka m-Test-ohjelma kykenee käynnistämään. Kolmannella välilehdellä on mahdollisuus käynnistää suoraan komentokehoteversioita mrix käskyistä. Tätä voidaan käyttää esimerkiksi omien LUA-skriptien käynnistämiseen kohdelaitteessa. Neljänneltä välilehdeltä löytyy informaatiota kohdelaitteesta kuten vapaan muistin määrä ja akun varaustila.

Viidennen välilehden kautta suoritetaan tekstintunnistus kohdelaitteen näytöstä. Välilehdellä on kuvan 5 mukainen lomake, jolla säädetään tekstintunnistuksen parametrit. Tärkein säädettävistä arvoista on profiili, joka käytännössä tarkoittaa

kohdelaitteessa käytettyä fonttityyppiä. Tämän lisäksi voidaan asettaa alue jolta tekstintunnistus suoritetaan. Pienemmältä alueelta tunnistus käy nopeammin, ja toisinaan halutaankin vain tietää jonkun tarkkaan määritetyn alueen, kuten ikkunan otsikon, sisältämä teksti. Kolmas oleellinen asetettava arvo on tekstintunnistuksen toleranssi. Sen avulla m-Test-ohjelma pyrkii erottamaan merkkeihin kuuluvat pikselit taustakuvasta. Kun tekstintunnistus on suoritettu, löytyneet tekstit tulevat samalle välilehdelle omaan tekstilaatikkoonsa. Siitä ne voi kopioida talteen tai ohjelmallisesti tutkia mitä tekstiä tunnistukseen käytetyllä alueella sillä hetkellä oli.

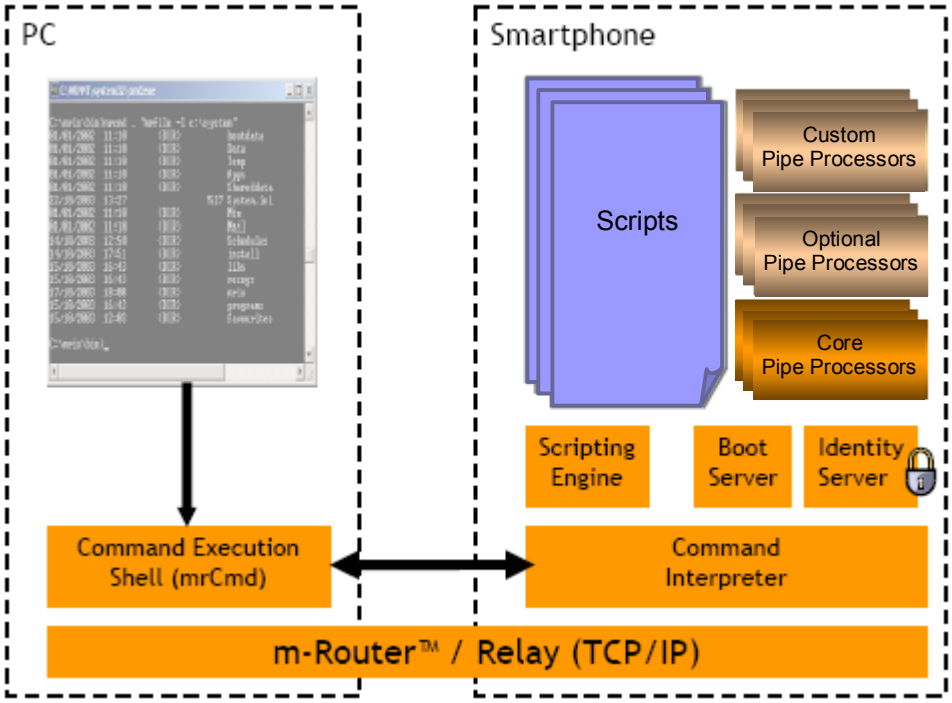


Kuva 5 Tekstintunnistuksen asetukset

m-Test-ohjelma on suunniteltu Series60-sarjan älypuhelimille, mutta tuki löytyy myös Series80-sarjan laitteille (Nokia 9500) ja UIQ-laitteille (Sony Ericsson P800 ja P900). Ennen kuin m-Test voi ohjata älypuhelinta, täytyy siihen asentaa mrix-komponentti. Mrix on etäkomentojen suoritusympäristö älypuhelimessa. Komento-tulkki kytkeytyy älypuhelimeen ”putkiprosessoreiden” kautta. Ne ovat pieniä itsenäisiä C++ -moduleja tai LUA-skriptitiedostoja. Kukin niistä koteloi ryhmän älypuhelimen toiminnallisuudesta kuten näppäimistökäsittelijä tai puhelu-toiminnallisuus. /7; 8; 9/

Kuvassa 6 on esitetty mrix-arkkitehtuuri. Se rakentuu älypuhelimessa olevan komentotulkin ja PC:llä olevan komentokehotteen (mrCmd) ympärille. Tiedon välittämisestä PC:n ja älypuhelimen välillä vastaa Intuwaven kehittämä m-Router. Se koostuu PC:lle sekä mobiiliohjelmistoon tulevista komponenteista ja tarjoaa IP pohjaisen yhteyden näiden välille. Mahdollisia siirtoteitä ovat USB, Infrapuna, Bluetooth ja sarjakaapeli. Komentokehote lähettää m-Routerin ja komentotulkin välityksellä käskyjä putkiprosessoreille. Mikäli kyseessä on suoritettava LUA-

skripti, käskyt ohjautuvat skriptimoottorille. Tehtävän suorituksen jälkeen mahdolliset vastaukset palaavat samaa reittiä takaisin komentokehotteelle. /8; 10/



Kuva 6 mrix arkkitehtuuri /8, s.2/

Putkiprosessorit jaetaan kolmeen ryhmään. Ensimmäisen ryhmän muodostavat pakolliset, aina käytössä olevat putkiprosessorit. Ne muodostavat järjestelmän ytimen. Esimerkkeinä ovat tapahtumien hallinta sekä prosessien käynnistys ja lopetus. Toisen ryhmän muodostavat valinnaiset putkiprosessorit, jotka voidaan poistaa ajonaikaisesta järjestelmästä. Näin saadaan mrix:n käyttämä muistimäärä minimoitua. Esimerkkeinä ovat puhelutoiminnallisuuden hallinta, tiedosto-järjestelmän käyttö ja näppäinsyötteet. Kolmannen ryhmän muodostavat käyttäjän tekemät omat putkiprosessorit. Näiden avulla voidaan luoda toiminnallisuutta, jota ei mrix:n peruspaketista löydy. /8/

m-Test sisältää mrix-järjestelmän PC komentokehotetta vastaavan toteutuksen, jolla se kytkeytyy älypuhelimessa olevaan mrix järjestelmään m-Routerin kautta.

4.2 QuickTest Pro

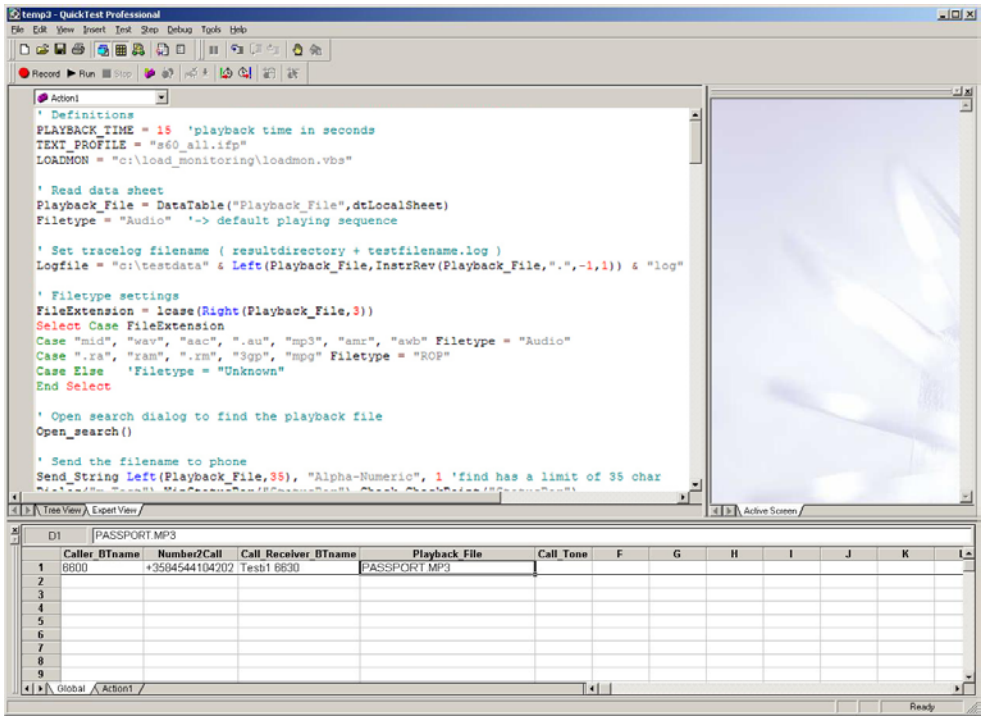
m-Test on vain käyttöliittymä älypuhelimelle eikä siinä itsessään ole ominaisuuksia testauksen automatisointiin. QuickTest Pro on yhdysvaltalaisen Mercury Interactiven kehittämä automointi- ja testaustyökalu, jonka avulla pystyy ohjailemaan mitä tahansa windows ohjelmaa. QuickTest Pro simuloi ihmistä käyttämällä tietokoneen hiirtä ja näppäimistöä virtuaalisesti syötteiden antamiseen ohjelmien graafisille käyttöliittymille. QuickTest Pro pohjautuu nauhoita & toista -toiminnallisuuteen. Tällöin käyttäjän tekemät hiirenliikkeet ja -napsautukset sekä näppäimistösyötteet tallennetaan myöhempää toistoa varten. Syötteistä muodostuu kuvakepohjainen testipuu, jossa käyttäjän toimet esitetään puumaisena rakenteena kuvaten testin askelia. Samat tiedot testin askelista saa nähtäväksi myös VBScript-tiedostona, joka luodaan samaan aikaan. Kumpaakin näkymää voi itse muokata jälkikäteen. Esimerkiksi tauot näppäinten painallusten välillä ja muut viiveet lisätään jälkikäteen, sillä niitä ei nauhoituksen aikana tallenneta. Samoin halutut silmukka- ja ehtorakenteet sekä omat vertailukutsut on lisättävä jälkikäteen tai nauhoituksen aikana pysäyttämällä nauhoitus muokkauksen ajaksi. Nauhoituksen aikana voidaan lisätä tarkastuspisteitä, joiden avulla lopulta päätetään oliko testi hyväksytty vai hylätty. Tarkastuspisteessä voidaan ottaa esimerkiksi ruutukaappaus näytöstä ja verrata sitä testin nauhoituksen aikana tehtyyn ruutukaappaukseen tai etsiä tekstintunnistuksella tiettyä tekstiä ruudulta. /11/

Nauhoitetun skriptin toiston aikana QuickTest Pro ottaa tietokoneen hiiren ja näppäimistön hallintaansa, ja toistaa samat toiminnot kuin nauhoituksen aikana. Koska QuickTest Pro käyttää hiirtä ja näppäimistöä virtuaalisesti, ei testin aikana koneeseen tule koskea. Esimerkiksi uuden ohjelman avaaminen näytölle sotkee testauksen, sillä skripti ei osaa odottaa sellaista näytöltä löytyvän, eivätkä hiiren klikkaukset ja näppäinsyötteet osu siihen kohteeseen johon ne oli tarkoitettu.

Käytettäessä nauhoitusta testiskriptin tekoon QuickTest Pro liittää kuhunkin askeleeseen kuvakaappauksen siitä mitä oltiin tekemässä. Esimerkiksi kuva ohjelmaikkunasta, jossa painettu nappi on korostettuna. Tämä aiheuttaa tiedostokoon kasvamisen, mutta testin vaiheet ovat myöhemminkin selkeästi näkyvissä. Niitä voidaan käyttää esimerkiksi ohjeena testin suorittamisesta uudelle

testaajalle. Nämä tiedot ovat näkyvissä vain puunäkymässä, eikä niitä voi jälkikäteen muuttaa muuten kuin nauhoittamalla kyseinen kohta uudelleen.

Kuvassa 7 on näkymä QuickTest Pro-ohjelman käyttöliittymästä skriptien manuaalisessa muokkaustilassa. Ohjelmassa on excel tyyppinen tietotaulukko, jonka sisältämää tietoa voidaan hakea testiskripteissä muuttujiin funktiolla DataTable. Taulukko näkyy kuvan 7 alalaidassa. Taulukon yksi rivi vastaa aina yhtä testin iteraatiokierrosta. Ensimmäisellä kierroksella käytetään ensimmäisen rivin tietoja, toisella kierroksella toisen ja niin edelleen. Multimedia käyttötapaus-testeissä kullekin taulukon riville asetettiin sen tiedoston nimi, joka kyseisellä iteraatiokierroksella haluttiin soitettavan. Laajemmissa testeissä käytettyjä tietoja olivat lisäksi halutun soittoäänen nimi, kohdelaitteen puhelinnumero ja järjestelmässä olleiden puhelimien bluetooth nimet.



Kuva 7 QuickTest Professional -ohjelman käyttöliittymää

Liitteessä 1 on esimerkki QuickTest Pro-testiskriptistä, joka suorittaa yhden iteraation ”Local Music Playback”-käyttötapaustestistä. Koska kaikkien multi-media käyttötapausten testaaminen oli perustoiminnaltaan hyvin samanlaista, samaa perusrunkoa voitiin hyödyntää myös monimutkaisemmissa käyttö-

tapaustesteissä. Lisäyksenä niihin tehtiin esimerkiksi ohjaus, jolla toinen puhelin saatiin soittamaan testauksen alla olevaan puhelimeen. Myös testin alustusta laajennettiin asettamaan puhelimen soittoääneksi halutun formaatin mukainen tiedosto.

Perustesti koostuu kahdesta vaiheesta. Aluksi puhelin määritellään haluttuun lähtötilaan. Seuraavaksi suoritetaan varsinainen testi-iteraatio halutuille audio- ja/tai videoformaateille. Virhetilanteessa puhelin palautetaan perustilaan valmiiksi seuraavaa iteraatiota varten. Testi-iteraatio jakaantuu seuraaviin vaiheisiin:

1. Alkuarvojen asetus, mm. soitettavan tiedoston nimi
2. Soitettavan tiedoston haku ja tarkistus, että se löytyi
3. Toiston aloitus
4. Soitto ja lokin tallennus määritellyn ajan verran
5. Toiston lopetus ja lopputilan tarkistus
6. Lokitiedoston analysointi.

4.3 FastTrace ja skriptit

FastTrace on Nokian sisäinen ohjelma. Se toimii pc:llä ja ottaa vastaan matkapuhelimen debug- ja trace-viestejä tähän tarkoitukseen tehdyn liitynnän kautta. Ohjelman avulla nämä viestit voidaan tallentaa tiedostoon myöhempää käyttöä varten. Debug-tulosteet ovat muun muassa ilmoituksia ohjelmiston tilasta, virheviestejä ja muuta informaatiota, joka katsotaan hyödylliseksi virheitä etsittäessä. Trace-viestit kertovat tarkemmin järjestelmän tilasta, muistin määristä, vuorossa olevista prosesseista ja niiden vaihtelusta. Trace-viestien avulla saadaan selville muun muassa kuinka eri audio- ja videokoodekit kuormittavat prosessoria ja kuinka ne käyttävät muistia. FastTracen tekemästä tiedostosta parsitaan perl skripteillä erilleen nämä tiedot ja jatkokäsitellään ne. Lopputuloksena on esimerkiksi excel-kuvaaja prosessorin kuormituksesta ajan funktiona jaoteltuna käynnissä olleiden prosessien kesken. Toinen käyttötarkoitus on testin aikana syntyneiden virheviestien poiminta muun tiedon joukosta ja niiden raportointi omaan tiedostoonsa.

5 PÄÄTELMÄT

Multimedia käyttötapaustestauksen tarkoituksena oli selvittää video- ja audio -sisältöjen toimivuus käyttäjän näkökulmasta. Testauksessa ei otettu kantaa videoiden ja musiikin toiston laatuun, sillä sen arvioiminen automatisoidusti ei onnistu järkevästi eikä se ole tarkoituksenmukaistakaan. Testauksessa keskitytäänkin vain siihen, että toisto saadaan käyntiin, ja että yhteistoiminta soitto- ja muiden järjestelmä-äänien kanssa toimii. m-Test-ohjelma on luonteeltaan suunniteltu käyttöliittymän testaamiseen, ja siksi m-Test + QuickTest Pro -yhdistelmä toimiikin hyvin tämän kaltaisessa funktionaalisessa testauksessa, jossa tarkoituksena on simuloida ihmisen toimintaa. Tätä multimedian käyttötapaustestaus nimenomaan on. Myös m-Test-ohjelman kyky ohjata useaa kohdelaitetta helpottaa monimutkaisempien käyttötapausten testausta.

5.1 Järjestelmä

Automatisoiduista testeistä saadaan eniten hyötyä, kun testattavaa on paljon ja samoja testejä suoritetaan useasti. Kuten aiemmin on mainittu, multimedia käyttötapauksissa testattavaa riittää. Kaikkien eri yhdistelmien läpikäyminen käsityönä vaatisi aivan liian paljon aikaa. Tätä urakkaa m-Test-järjestelmän odotettiin helpottavan, ja se todettiin kyseiseen tehtävään hyvin soveltuvaksi. Automatisoidut testit eivät kuitenkaan ole nopeudeltaan välttämättä optimaalisia. Ihminen osaa sopeutua ja pystyy helposti havainnoimaan koska järjestelmä on sellaisessa tilassa, että testausta voidaan jatkaa. Testiskripteissä joudutaan usein tyytymään kiinteisiin viipeisiin, joihin joudutaan ottamaan turhaakin varmuusmarginaalia. Tästä johtuen suppeamman ”Critical cases” -testijoukon testaaminen kävi jopa nopeammin käsin suoritettuna testauksena. Critical cases -testauksen tarkoituksena olikin saada nopeasti kuva sen hetken toiminnallisuudesta, ja siinä painotettiin kohteita, joihin oli muutoksia tullut. Käsin testauksen ansiosta myös toiston laatua voitiin tarkkailla visuaalisesti. Laaja testikokonaisuus oli silti pääasiallinen testaus. Se antoi tarkan kuvan siitä mikä toimi ja mikä ei. Critical cases täydensi kokonaisuutta.

Testausjärjestelmä saatiin suorittamaan suuri määrä testejä automatisoidusti. Tuloksena tästä oli paljon debug- ja tracelokia sekä excel kuvaajia. Skriptien avulla

data jalostettiin mahdollisimman pitkälle, mutta ihmistä tarvittiin silti kokoamaan testin tulokset lopullisesti yhteen. Tulosten raportoiminen olikin suuren testikierroksen eniten työtä vaativa osa, sillä sitä ei vielä saatu automatisoiduksi. Se onkin kohde, joka tulevaisuudessa pitäisi pyrkiä automatisoimaan.

Yleisellä tasolla QuickTest Pro ja m-Test-ohjelmien käytettävyys oli hyvä. Niiden käyttämiseen pääsi sisälle ilman mainittavaa aiempaa kokemusta automatisointiohjelmien käytöstä hyvien ohjetiedostojen avulla. Eniten ongelmia järjestelmän pystytyksessä aiheutti mrix-modulin asennus puhelimeen, sillä kyseinen moduli on riippuvainen käytössä olevasta Symbian ja Series60 versiosta. Jo markkinoilla olevilla puhelimilla järjestelmän käyttöönotto olisi varmasti melko mutkaton, mutta kun kyseessä on kehitysvaiheessa oleva prototyyppipuhelin, niin ongelmia ilmenee. Ne saatiin kuitenkin ratkottua yhteistyössä moduulin valmistajan kanssa. Myös itse kohdelaitteen ohjelmiston taso aiheutti testauksen alkuvaiheessa ongelmia, sillä sen vakaus ja toiminnallisuus ei aina ollut täysin kunnossa. Esimerkki alkuvaiheessa ongelmia aiheuttaneesta toiminnallisuudesta tai sen puutteesta oli bluetooth-yhteyden epäluotettava toiminta, minkä vuoksi testin kulumista piti säännöllisesti seurata. Luonnollisesti tämä hidasti testausprosessia ja sitoi testaajan työpanosta muista tehtävistä.

5.2 Testaus

Musiikki- ja videotiedostojen toistaminen kohdelaitteessa automatisoituna toimintona oli melko yksinkertaista toteuttaa. Perusperiaatteena tiedoston toisto alkaa puhelimen tiedostonhallinnasta, jossa hakukenttään kirjoitetaan halutun tiedoston nimi. Tekstintunnistuksen avulla varmistetaan, että haluttu tiedosto löytyy. Järjestelmä avaa tiedoston oikeassa soitto-ohjelmassa, jossa toiston annettaan pyöriä halutun ajan. Samalla tallennetaan puhelimen debug- ja tracedataa tiedostoon ja mahdollisesti tehdään puhelu apupuhelimesta testipuhelimeen. Toiston kestettyä halutun ajan, se keskeytetään ja puhelin palautetaan perustilaan odottamaan seuraavaa testikierrosta. Tarvittava toiminnallisuus oli siis tekstin syöttöä ja muutamien nappien painelua, sekä kohdelaitteen näytöstä tekstin tunnistusta.

Testiskriptit päätettiin toteuttaa QuickTest Pron nauhoitustoiminnallisuuden sijaan käsinkirjoitetuilla VBScript-tiedostoilla, sillä ne todettiin ylläpidettävyydeltään helpommiksi ja selkeämmiksi. Tällöin menetetään kuvakaappaukset testin vaiheista, mutta niitä ei nähty oleellisiksi tiedoiksi. Skriptien kirjoittaminen käsin vaatii VisualBasic tuntemusta, mutta tätäkin varten QuickTest Pron mukana tuli hyvä ohje sekä lisäksi Internet on täynnä ohjeita. Nauhoitustoiminnallisuus oli ajatukseltaan hyvä, mutta ei käytännössä toimiva. Ongelmia olisi aiheuttanut esimerkiksi QuickTest Pron tapa tallentaa hiiren painallukset suhteellisina koordinaatteina avoinna olevaan ohjelmaikkunaan. Nauhoitetuillekin testeille löydettiin hyötykäyttöä tilapäisissä testeissä tapauksiin, joissa jotain asiaa piti testata pikaisesti useampaan kertaan. Esimerkiksi tapaus, jossa noin joka 20. saapunut tekstiviesti aiheutti videotoiston kaatumisen. Käsin olisi vaivalloista lähetellä jatkuvasti tekstiviestejä, mutta nauhoituksella sai nopeasti luotua helposti toistettavan testin. Nauhoittamalla sai myös helposti tuotettua esimerkkikoodia jostain tietystä toiminnasta.

Testejä varten kehitettiin mahdollisimman paljon itsenäisiä funktioita, joita käyttäen skriptit saatiin selkeämmiksi ja luettavammiksi. Funktiot peittävät m-Testin käyttöliittymän muodostaen tarvittavat rutiinit, joita tarvitaan haluttaessa esimerkiksi lähettää tekstiä kohdelaitteelle tai suorittaa tekstintunnistusta. Tämä myös helpottaa skriptien uudelleenkäytettävyyttä ja siirrettävyyttä, mikäli sellaiselle olisi tulevaisuudessa tarvetta. Perustoiminnallisuusfunktioiden lisäksi luotiin joukko funktioita erikoistoimintoja varten. Näitä ovat esimerkiksi funktiot debug- ja tracelokin tallennusta ja analysointia sekä puhelun suoritusta varten. Kaikkea video- ja audiosisältöä ei toisteta puhelimessa samalla ohjelmalla. Tässä multimedia käyttötapaustestauksessa mahdollisia toisto-ohjelmia oli kaksi erilaista. Se, kummalla toistettava tiedosto aukeaisi, pääteltiin tiedostopäätteen perusteella. Toisto-ohjelmien ohjaukseen tarvittavan koodin ero oli niin pieni, että sitä varten ei tehty omia funktioita, mutta tulevaisuuden kannalta niiden modularisointi olisi voinut olla hyödyllistä.

Puhelimen valikkojen selauksessa ja oikean kohdan löytämisessä käytettiin hyödyksi tekstintunnistusta. Se on luotettavampi tapa kuin indekseihin pohjautuva

valikon selaaminen. Tällöin valikkojärjestyksellä tai valintojen määrällä ohjelmiston eri tiloissa ei ole merkitystä, ja navigointi kuvastaa enemmän yleistä Series60 järjestelmää kuin jotain tiettyä puhelinmallia. Tekstintunnistus m-Testissä pohjautuu hahmontunnistukseen. Jotta tekstit osattaisiin tunnistaa, pitää m-Testille kertoa puhelimessa käytössä oleva fonttityyppi. Tekstin tausta voi olla jotain muutakin kuin tasainen värikenttä. Tämän vuoksi hahmontunnistukseen asetetaan myös toleranssi, jolla säädetään tunnistuksen herkkyyttä eli päätellään mitkä pikselit kuuluvat kirjaimen muotoon ja mitkä eivät. Toleranssin säätäminen sopivaan arvoon vaati useita kokeiluja, mutta kun se saatiin kohdilleen, toimi tekstintunnistus pääsääntöisesti luotettavasti.

Luotuja testiskriptejä ei juuri tarvinnut rikkoutumisen vuoksi korjailla. Tässä auttoi varsinaisen testilogiikan yksinkertaisuus ja se, että käyttöliittymä oli testauksen aloitusvaiheessa jo hyvin vakaalla pohjalla. Ainoa toisinaan korjailua vaatinut kohde oli nimenomaan edellä mainittu tekstintunnistuksen toleranssisäätö.

LÄHDELUETTELO

1 Haikala, Ilkka; Märijärvi, Jukka. Ohjelmistotuotanto 7. painos. Satku. Pieksämäki 2001. 414s.

2 Ala-Kurikka, Timo. Ohjelmistotestauksen kehittäminen ja automatisointi. Opinnäytetyö. Vaasan Ammattikorkeakoulu. Elektroniikka ja tietotekniikka. 2001. 54 s. + 2 liites.

3 Wikipedia, the free encyclopedia. [www-sivu].[viitattu 3.4.2005] saatavissa: http://en.wikipedia.org/wiki/Use_case

4 Brian Marick. When Should a Test Be Automated? [viitattu 15.4.2005] saatavissa: www.testing.com/writings/automate.pdf

5 Tianyo, Cao. Test automation in customized series 60 mobile phone software. Opinnäytetyö. EVTEK-Ammattikorkeakoulu.. Tietotekniikka. 2004. 57 s.

6 Software test automation and the product life cycle. [www-sivu].[viitattu 29.4.2005] saatavissa: http://www.mactech.com/articles/mactech/Vol.13/13.10/SoftwareTestAutomation/

7 m-Test Getting Started Guide. Intuwave Limited 2004

8 mrix DataSheet revision 1.4. Intuwave Limited 2004

9 m-Test whitepaper. [viitattu 14.3.2005] saatavissa: http://www.intuwave.com

10 m-Router Functional Description. version 0.5 Intuwave Limited 2005

11 Automated Testing – Mercury QuickTest Professional. [www-sivu].[viitattu 7.12.2005] saatavissa: http://www.mercury.com/us/products/quality-center/functional-testing/quicktest-professional/

```
' Definitions
PLAYBACK_TIME = 15  'playback time in seconds
TEXT_PROFILE = "s60_all.ifp"
LOADMON = "c:\load_monitoring\loadmon.vbs"

' Read data sheet
Playback_File = DataTable("Playback_File",dtLocalSheet)
Filetype = "Audio"  '-> default playing sequence

' Set tracelog filename ( resultdirectory + testfilename.log )
Logfile = "c:\testdata" & Left(Playback_File,InstrRev(Playback_File,".",-1,1)) & "log"

' Filetype settings
FileExtension = lcase(Right(Playback_File,3))
Select Case FileExtension
Case "mid", "wav", "aac", ".au", "mp3", "amr", "awb" Filetype = "Audio"
Case ".ra", "ram", ".rm", "3gp", "mpg" Filetype = "ROP"
Case Else   'Filetype = "Unknown"
End Select

' Open search dialog to find the playback file
Open_search()

' Send the filename to phone
Send_String Left(Playback_File,35), "Alpha-Numeric", 1 'find has a limit of 35 char
Dialog("m-Test").WinStatusBar("StatusBar").Check CheckPoint("StatusBar")

' Do the search
Key "Navikey", 1, 5, 0
Dialog("m-Test").WinStatusBar("StatusBar").Check CheckPoint("StatusBar")

' Check that file is found
If Check_Text_Wait_For(TEXT_PROFILE,"Back","All","",50,0,10) Then
    check_temp = Check_Text_Fetch(TEXT_PROFILE,"All","",50)
    If InStr(1,check_temp,Left(Playback_File,5),1) Then
        reporter.ReportEvent 0, text, "Playback file was found."
    Else
        reporter.ReportEvent 1, text, "Error: Playback file not found."
        Key "SK_Right", 3, 3, 0
        Key "End_Call", 3, 5, 0
        Application "Launch", "File_manager", 3
        ExitActionIteration("File not found")
    End If
Else
    reporter.ReportEvent 1, text, "Error: Timeout! Searched for '" & Playback_File & "'"
    Key "SK_Right", 3, 3, 0
    Key "End_Call", 3, 5, 0
    Application "Launch", "File_manager", 3
    ExitActionIteration("File not found")
End If

' Play the file
Key "Navikey", 1, 5, 0
If filetype = "ROP" Then wait(5) 'ROP starts a little slower
Dialog("m-Test").WinStatusBar("StatusBar").Check CheckPoint("StatusBar")

' Start Trace log
Trace_Log "open", Logfile
wait(PLAYBACK_TIME) 'Let it play for the specified time...
' Stop trace logging
Trace_Log "close", ""

' Leave the player
Key "SK_Right", 3, 1, 0

' Check final state
If Check_Text_Wait_For(TEXT_PROFILE,"File Manager","All","On",70,1,5) Then
    reporter.ReportEvent 0, text, "Iteration completed succesfully!"
Else
    reporter.ReportEvent 1, text, "Error: Iteration didn't end properly - aborting..."
    exitaction()
End If

' Load monitoring
Run_Load_Monitor LOADMON, Logfile
```