TAMPEREEN TEKNILLINEN YLIOPISTO

JONI KÄKI

SYMBIAN MIDDLEWARE -OHJELMISTOPROJEKTIN TESTAUKSEN KEHITTÄMINEN

Diplomityö

Tarkastajat: professori Jarmo Harju
ja DI Aki Kemppainen
Tarkastajat ja aihe hyväksytty
Tieto- ja sähkötekniikan
tiedekuntaneuvoston
kokouksessa 3. maaliskuuta 2010

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
KÄKI, JONI: Symbian middleware -ohjelmistoprojektin testauksen kehittäminen
Diplomityö, 50 sivua, 2 liitesivua
Kesäkuu 2010
Pääaine: Tietoliikenneverkot ja -protokollat
Tarkastajat: professori Jarmo Harju ja DI Aki Kemppainen
Avainsanat: Symbian, testaus, ketterät menetelmät, middleware, DVB-H

Ohjelmistoprojekteissa tuotettavien ohjelmistojen monimutkaisuus, laajuus ja määrä kasvavat jatkuvasti. Tämä tuo toteutettavien ohjelmistojen testaukselle uusia haasteita. Yhä laajempia kokonaisuuksia on pystyttävä toteuttamaan ja testaamaan tiukkojen aikarajojen puitteissa. Järjestelmien testausta kehitettävä, jotta toteutettavien ohjelmistojen laatu pystyttäisiin takaamaan kasvavien vaatimusten mukaiseksi. Testausta pyritään kehittämään ottamalla käyttöön erilaisia työkaluja ja automatisoimaan mahdollisimman suuri osa testauksesta.

Tässä diplomityössä kuvataan ohjelmistoprojektin testauksen kehittämistä Symbian OS middleware -ohjelmistoja toteuttavassa projektissa. Projektin moduulitestausta kehitettiin toteuttamalla testiautomaatiokehys olemassa olevien moduulitestien ajamiseen. Integraatio- ja järjestelmätestauksessa on usein ongelmana, että kaikki testattavat komponentit eivät ole vielä valmiita. Tällöin joudutaan kehittämään uusia työkaluja, joiden avulla puutteista aiheutuvat ongelmat voidaan kiertää. Järjestelmätestausvaiheessa on lisäksi mahdollista, että komponentteja ei voida testata tuotantokäytössä olevia järjestelmiä vastaan. Myös tällöin on luotava ratkaisuja, miten testaus voidaan mahdollistaa.

Arvioinnin perusteella voidaan sanoa, että projektin tavoitteet saavutettiin. Projektin testisykliä pystyttiin lyhentämään ratkaisevasti sekä integraatio- ja järjestelmätestauksen ongelmia onnistuttiin ratkomaan.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
KÄKI, JONI: Improving testing in Symbian middleware project
Master of Science Thesis, 50 pages, 2 Appendix pages
June 2010
Major: Computer Networks and Protocols
Examiners: Professor Jarmo Harju and M.Sc. Aki Kemppainen
Keywords: Symbian, testing, agile, middleware, DVB-H

Complexity, scope and volume of software products continue to grow. This brings up new challenges for software testing processes. Larger and larger entities must be able to be implemented and tested in the strict time limits. Testing must be improved so that certain level of quality can be guaranteed and growing requirements could be fulfilled. Testing can be improved through the introduction of a variety of tools and test automation.

This thesis describes a Symbian OS middleware software development project. Testing in the project was improved through successful implementation of test automation framework using the existing module tests.

In the project it was apparent, that common problem in integration- and system testing is often that all the tested components are not yet ready. In this case, there is need to develop new tools to help circumventing these problems. During system testing phase, it is also possible that the components can not be tested in production environment. Again, there is need to create solutions for how to allow the testing.

In the end, we can say that the project objectives were met. The project was able to drastically shorten the test cycle. Also integration- and system testing problems were successfully solved.

ALKUSANAT

Tämä diplomityö on tehty Sasken Finland Oy:n alihankintaprojektina asiakkaalle vuosien 2008-2010 aikana. Sasken Finland Oy on Sasken Communication Technologies Ltd:n tytäryhtiö. Sasken Finland Oy:n toimialaan kuuluvat alihankintaprojektit muun muassa useille mobiiliteknologia-alan suuryrityksille.

Kiitän työn tarkastajina toimineita professori Jarmo Harjua Tampereen teknillisestä yliopistosta ja diplomi-insinööri Aki Kemppaista Sasken Finland Oy:stä arvokkaista neuvoista ja kommenteista.

Lisäksi haluan kiittää vanhempiani Eila ja Jorma Käkeä, jotka ovat koko opintoaikani kannustaneet opiskelemaan. Avovaimolleni Jenni Paavilaiselle kuuluu myös suuri kiitos saamastani kannustuksesta ja tuesta niin opiskeluaikanani kuin työtä kirjoittaessanikin. Viimeisimpänä haluan kiittää tytärtäni Linneaa siitä, että hän on muistuttanut minua työn tuskaisimpina hetkinä, että on olemassa elämää opiskelun jälkeenkin.

Lopuksi haluan kiittää ystäviäni, jotka ovat kannustuksellaan ja vinkeillään aikaansaaneet intoa puskea tutkinnon viime rippeet kasaan.

Ruutana, 1. kesäkuuta 2010

Joni Käki
Nevastie 24
36110 Ruutana
SUOMI

SISÄLLYS

Tiivistelmä II
Abstract III
Alkusanat IV
Termit ja niiden määritelmät VII
1. Johdanto 1
2. MobiiliTV-asiakasohjelma 3
2.1. Symbian OS 3
2.1.1. Aktiiviset oliot 4
2.1.2. Asiakas-palvelin-suunnittelumalli 5
2.1.3. Verkkototeutukset Symbianissa 5
2.1.4. Symbian muistinhallinta 6
2.1.5. Virheiden käsittely 6
2.2. DVB-H middleware 7
2.3. Kokonaisjärjestelmä 8
3. Ohjelmistojärjestelmien testaus 10
3.1. Yleistä ohjelmistojen testauksesta 10
3.2. Moduulitestaus 12
3.3. Integraatiotestaus 13
3.4. Järjestelmätestaus 13
3.5. Testauksen automatisoiminen 14
4. Verkkotekniikat 15
4.1. DVB-H 15
4.1.1. Virransäästö ja aikajakoisuus 15
4.1.2. Multiprotocol Encapsulation 16
4.1.3. Handover 18
4.2. IPDC 18
4.2.1. PSI/SI 18
4.2.2. Sähköinen palveluopas 19
4.2.3. ESG- ja SDP-kuvaus 21
4.2.4. Sisällönjakelu 22
4.2.5. Palveluiden ostaminen ja suojaus 24
4.3. Interaktiivinen paluukanava 26
4.3.1. SMS-viesteihin perustuva paluukanava 26
4.3.2. Pakettikytkentäinen mobiilitiedonsiirto 26
4.3.3. HTTP-protokolla 27
4.4. Tiedonsiirtokanavan laatu 27
5. Testauksen kehittämisen tarve 29
5.1. Moduuli ja integraatiotestauksen kehittämistarpeet 29
5.2. Järjestelmätestauksen kehittämistarpeet 30
5.3. Siirtyminen ketterään ohjelmistoprosessiin 31

6. Testauksen kehittäminen 32
6.1. Testattava järjestelmä 32
6.2. Modulitestauksen automatisointi 33
6.2.1. Nightlyrunner 34
6.2.2. STIF-testikehys 35
6.2.3. Kattavuusmittaukset 36
6.2.4. Raportointi 36
6.3. RampTester 36
6.3.1. RampTesterin käyttöönotto 37
6.3.2. RampTesterin eri toimintamoodit 37
6.3.3. RampTester esimerkkitestitapaus 38
6.3.4. RampTesterin testiraportit 38
6.4. PumppuTester 40
6.4.1. PumppuTesterin käyttöönotto 40
6.4.2. PumppuTester esimerkkitestitapaus 41
6.4.3. PumppuTesterin testiraportit 42
6.4.4. Testien toistettavuus 42
6.5. TestResponder 43
6.5.1. Testipalvelimen toteutus 43
6.5.2. Konfiguraatiorajapinnan protokolla 44
6.5.3. Asiakasrajapinnan protokolla 44
6.5.4. Testiserverin yhdistäminen integraatiotesteihin 45
7. Yhteenveto 46
7.1. Testauksen kehittämisen arviointi 46
7.2. Jatkokehitysideoita 47
Lähteet 48
Liitteet 51

TERMIT JA NIIDEN MÄÄRITELMÄT

3G	3rd Generation. Yleinen lyhenne kolmannen sukupolven matkapuhelinverkkotekniikoille.
18Crypt	Järjestelmä palvelun ostamiseen ja suojaukseen.
AMR-WB+	Extended Adaptive Multi-Rate -Wideband. Häviöllinen äänenpakkausstandardi.
ALC	Asyncronous Layer Coding.
CDP	Content Delivery Protocol.
DVB-H	Digital Video Broadcast-Handheld. Yksi digitaalisen television standardeista Erityisesti pienitehoisille ja liikkuville päätelaitteille suunniteltu järjestelmä.
DVB-T	Digital Video Broadcast-Terrestial. Maanpäällisen digitaalisen television standardi.
ESG	Electronic Service Guide. Sähköinen ohjelmaopas.
FDT	File Delivery Table.
FEC	Forward Error Correction. Virheenkorjauskoodi.
FLUTE	File deLivery over Unidirectional Transport. Protokolla tiedostojen jakeluun joukkojakeluverkoissa.
GPRS	General Packet Radio System. Toisen sukupolven matkapuhelinverkkojen pakettikytkentäinen siirtopalvelu.
H.264/AVC	H.264 eli MPEG-4 AVC (Advanced Video Coding) on videonpakkausstandardi. Se sopii käytettäväksi esimerkiksi matkapuhelimessa katsottavan videokuvan pakkaamiseen, kuten esimerkiksi digitaalisissa DVB-H-televisiolähetyksissä.
HE AAC v2	High-Efficiency Advanced Audio Coding. Häviöllinen äänenpakkausstandardi.

HTTP	Hypertext Transport Protocol.
HSPA	High Speed Packet Access. Kolmannen sukupolven matkapuhelinverkkojen pakettikytkentäinen siirtopalvelu.
IP	Internet Protocol. Verkkokerroksen protokolla internet liikennöintiin. IPv4 tarkoittaa protokollan versiota 4 ja ja IPv6 versiota 6.
IPDC	Internet Protocol Datacasting. Joukko standardeja IP-datan välittämiseksi joukkojakeluverkossa.
IPSec	IP Security Architecture. Protokolla internet yhteyksien suojaamiseen.
ISMACryp	ISMA Encryption and Authentication, version 1.1. Salaus- ja autentikointipalveluiden määrittely.
KMM	Key Management Message. Avaimenhallintaviesti.
KSM	Key Stream Message. Avainvirtaviesti.
Middleware	Middlewarella eli välikerrosohjelmistolla tarkoitetaan palveluita, jotka mahdollistavat useamman prosessin yhteentoimivuuden tarjoamalla hallitun rajapinnan käyttöjärjestelmän palveluihin.
MPE	Multiprotocol Encapsulation.
MPEG	Motion Picture Experts Group.
MPEG-2	Standardi yleisradiotasoisen kuvan pakkaukseen.
MPEG-2 ES	MPEG-2 Elementary Stream.
MPEG-2 TS	MPEG-2 Transport Stream.
OSF	Open Security Framework. Järjestelmä palvelun ostamiseen ja suojaukseen.
PID	Program ID. Ohjelmatunniste.

Perl	Practical Extraction and Report Language. Larry Wallin kehittämä tulkattava ohjelmointikieli.
PSI/SI	Program Specific Information/Service Information. Palveluntarjoajan lähetysvirran mukana lähettämä tieto, jonka avulla vastaanotin tunnistaa lähetyksessä tulevat palvelut.
RTP	Real-time Transport Protocol. Protokolla tosiaikaiseen tiedonsiirtoon.
RO	Rights Object. Käyttöoikeusolio.
S60	Nokia OYJ:n kehittämä matkapuhelinten sovellusalusta ja käyttöliittymä.
SDP	Session Description Protocol. Protokolla istuntojen tai esitysten kuvaamiseen.
Service System	DVB-H-operaattorin hallinnoima palvelin, joka välittää dataa DVB-H-verkkoon.
SPP	Service Purchase and Protection.
SRTP	Secure Real-time Transport Protocol. Suojattu protokolla tosiaikaiseen tiedonsiirtoon.
SSL	Secure Sockets Layer. Yksi yleisimpiä salausprotokollia.
STIF	Testausjärjestelmä, joka on erityisesti Symbian OS -käyttöjärjestelmän laitteille suunniteltu.
Symbian OS	Symbian Ltd:n kehittämä pienitehoisille ja vähillä resursseilla varustetuille laitteille tarkoitettu käyttöjärjestelmä.
TCP	Transmission Control Protocol.
TEK	Traffic Encryption Key. Liikenteen salausavain.
Testaus	Toimenpiteet, joiden tarkoituksena on löytää ohjelmistossa olevat virheet.

TLS	Transport Layer Security. Yksi yleisimpiä salausprotokollia.
TOI	Transport Object Identifier.
TPS	Transmitter Parameter Signalling.
UDP	User Datagram Protocol.
UMTS	Universal Mobile Telecommunications System. Kolmannen sukupolven matkapuhelinteknologia.
VC-1	Video Codec 1. Videonpakkausalgoritmi.
XML	eXtensible Markup Language. Rakenteinen merkintäkieli, jossa tiedon merkitys kuvataan tiedon mukana.

1. JOHDANTO

Symbian OS on johtava älypuhelimissa käytettävä käyttöjärjestelmä. Puhelimissa käytettävien ohjelmistojen monimutkaisuus, laajuus ja määrä kasvavat jatkuvasti. Tämä tuo toteutettavien ohjelmistojen testaukselle uusia haasteita. Yhä laajempia kokonaisuuksia on pystyttävä toteuttamaan ja testaamaan tiukkojen aikarajojen puitteissa. Jotta toteutettavien ohjelmistojen laatu pystyttäisin takaamaan kasvavien vaatimusten mukaiseksi, on järjestelmien testausta pystyttävä kehittämään. Testausta pyritään kehittämään ottamalla käyttöön erilaisia työkaluja ja automatisoimaan mahdollisimman suuri osa testauksesta.

Oman lisävaatimuksensa testauksen kehittämiseen tuo siirtyminen perinteisestä vesiputousmallin mukaisesta ohjelmistokehityksestä uudempaan ketteriä menetelmiä käyttävään ohjelmistokehitykseen. Ohjelmistosta on pystyttävä ketterässä ohjelmistokehityksessä julkaisemaan versioita lyhyemmissä sykleissä tuolloin toteutukselle ja testaukselle jää vähemmän aikaa. Varsinkin uusia ominaisuuksia toteuttaessa ilmentynyt regressio on pystyttävä havaitsemaan ensitilassa ja korjaamaan toteutuksessa ilmenneet virheet. Perinteisesti ohjelmistojen testaus on ollut ohjelmistoprojektin osa-alue, joka suoritetaan ohjelmistoprojektin loppuvaiheessa ohjelmiston ollessa jo valmis. Ketterä ohjelmistokehitys ei lyhyiden sprinttiensä takia välttämättä jättäisi testaukselle tarpeeksi aikaa, mikäli testaus suoritettaisiin projektin lopussa. Siksi testausta on kehitettävä siten, että ohjelmisto saadaan mahdollisimman pian testattavaksi. Mahdollisimman aikaisella testauksella pystytään luonnollisesti löytämään virheet aikaisemmassa vaiheessa, jolloin niiden korjaaminen on helpompaa ja halvempaa.

Ohjelmistoa integroitaessa haasteena on, etteivät kaikki tarvittavat komponentit ole vielä valmiina tai järjestelmä ei ole vielä tarpeeksi vakaa, jotta kaikkia komponentteja voitaisiin ajaa. Tälläinen tilanne on varsin yleinen ohjelmistoalihankintana toteutettavissa projekteissa, joissa eri alihankkijat toteuttavat osia kokonaisjärjestelmästä. Tarvittaessa tälläisissa tapauksissa on toteutettava erilaisia työkaluja testauksen mahdollistamiseksi.

Tässä diplomityössä kuvataan ohjelmistoprojektin testauksen kehittämistä Symbian OS middleware -ohjelmistoja toteuttavassa projektissa. Projekti on toteutettu Sasken Finland Oy:n alihankintaprojektina asiakkaalle.

Diplomityön luvussa 2 esitellään ohjelmistoprojektin asiakasohjelmisto ja siihen liittyvät käsitteet yleisellä tasolla. Lisäksi luvussa kuvaillaan asiakasohjelman arkkitehtuuria sekä kokonaisjärjestelmää, johon asiakasohjelmistoa käyttävä mobiilipäätelaite kuuluu. Luvussa 3 käsitellään ohjelmistojen testauksen taustalla olevaa teoriaa. Luvussa 4 käsitellään tietoliikennetekniikan teoriaa työn osalta oleellisilta osuuksilta. Luvussa 5 käsitellään syitä, joiden takia projektin testausta päätettiin kehittää. Luvussa 6 kerrotaan miten projektin testausta kehitettiin esittelemällä toteutettuja ja käytettyjä työkaluja. Luvussa 7 arvioidaan testauksen kehittämisen onnistumista.

2. MOBIILITV-ASIAKASOHJELMA

Diplomityössä käsiteltävä ohjelmisto on osa projektia, jossa tarkoituksena on toteuttaa MobiiliTV-asiakasohjelma Symbian S60 -käyttöjärjestelmän ja DVB-H-standardin mukaisen vastaanottimen sisältäviä päätelaitteita varten.

Tässä luvussa kuvataan Symbian OS -käyttöjärjestelmää ja niitä ominaisuuksia, jotka vaikuttavat itse sovelluksen toteutamiseen. Lisäksi kuvaillaan MobiiliTV-asiakasohjelmaa yleisellä tasolla sekä kokonaisjärjestelmää, jonka osana on päätelaite, jossa MobiiliTV-asiakasohjelmaan ajetaan.

2.1. Symbian OS

Symbian OS on mobiililaitteisiin suunniteltu 32-bittinen käyttöjärjestelmä. Mobiililaitteiden rajallisista resursseista johtuen käyttöjärjestelmään on tehty ratkaisuja, joilla pyritään mahdollisimman tehokkaaseen resurssien käytöön. Näistä ratkaisuista seuraa, että toteutettaessa mobiiliin laitteeseen ohjelmistoa tulee tietyt rajoitukset ottaa huomioon.

Välikerrosohjelmistot koostuvat useista eri palvelinohjelmistoista, joiden yhteistyöllä toteutetaan järjestelmän haluamat palvelut. Usean eri ohjelmiston yhtäaikainen ajaminen vaatii järjestelmältä moniajo-ominaisuuksia. Symbian OS -käyttöjärjestelmän vuorontava moniajo toteutetaan mikroytimessä tapahtuvassa vuorontajassa. Mikroytimen muita tehtäviä ovat muistinsuojaus ja laiteajureista vastaaminen. Lähes kaikki muut toiminnot Symbian OS käyttöjärjestelmässä tapahtuvat prosesseissa. Prosessit koostuvat yhdestä tai useasta säikeestä, joille ydin suorittaa irrottavaa vuorontamista. Moniajon toteuttaminen säikeiden avulla on kuitenkin vaikeaa, koska usean säikeen suorituksessa joudutaan huolehtimaan käytettävien resurssien poissulkemisesta esimerkiksi mutexien ja semaforien avulla. Säikeiden välillä tapahtuva vuorontaminen tuhlaa lisäksi resursseja, koska muistia ja kellojaksoja joudutaan käyttämään ympäristövaihdoksiin.

Yleisempi tapa toteuttaa moniajettavia ohjelmistoja Symbian OS -käyttöjärjestelmään onkin käyttää hyväkseen järjestelmän tarjoamia kevyempiä menetelmiä. Näitä menetelmiä ovat aktiiviset oliot, asiakas-palvelin-suunnittelumallin mukainen toteutustapa sekä prosessien väliset viestinvälitysmekanismit. [Harrison, 2003] [Stichbury, 2005]

.

2.1.1. Aktiiviset oliot

Symbian OS on tapahtumapohjaisuuteen perustuva käyttöjärjestelmä. Ylemmällä tasolla käyttäjän ja järjestelmän välinen vuorovaikutus toteutetaan tapautumapohjaisesti ja alemmalla tasolla asynkronisten palvelupyyntöjen toteutus pohjautuu tapahtumiin. Toiminnot suoritetaan tapahtumakäsittelijöissä, jotka perinteisesti on toteutettu erillisinä säikeinä, joiden välillä vuorontaminen on turhan raskas operaatio. Aktiivisten olioiden tapauksessa yksi säie pystyy käsittelemään aktiivisen vuorontajan avulla yhtä tai useampaa rinnakkaista aktiivista oliota. Suoritukseen valitaan aina korkeimman prioriteetin omaava tapahtuma. Aktiivinen vuorontaja etsii kutakin tapahtumaa vastaavan aktiivisen olion ja kutsuu tämän RunL()-metodia, jolloin suoritus siirtyy aktiiviselta vuorontajalta aktiiviselle oliolle.

Aktiivisen vuorontajan toiminta perustuu yhteistoiminnalliseen (cooperative multitasking) moniajoon. RunL()-metodin suoritusta ei keskeytetä vuorontajan toimesta, joten tarkoitus on, että metodi suorittaa haluamansa operaatiot mahdollisimman lyhyesti. Tämän jälkeen palataan vuorontajaan käsittelemään seuraavana oleva tapahtuma. Kyseessä ei ole siis puhdas moniajo vaan tapahtumien käsittelyä sarjamuodossa.

Aktiivisten olioiden käyttö mahdollistaa nopeamman käsittelyn kuin säikeisiin perustuva moniajo. Haittapuolena on että väärin toteutettuna pitkään suoritettavana oleva aktiivinen olio varaa järjestelmää, jolloin korkeampiprioriteettisten tapahtumien käsittely hidastuu.

Aktiivinen olio toteutetaan periyttämällä se luokasta CActive. CActive määrittelee kaksi puhdasta virtuaalifunktiota RunL() ja DoCancel(), joille on aina kirjoitettava toteutus. RunL() toteuttaa olion tapahtumakäsittelijän ja DoCancel() käsiteltävän tapahtuman peruuttamistoimenpiteet. Kolmantena toteutettavana funktiona on RunError(), jolla voidaan toteuttaa virhetilanteiden käsittely RunL()-funktiolle.

Aktiivisen olion luonnin yhteydessä olio rekisteröidään aktiiviselle vuorontajalle. Tämän jälkeen se jää kuuntelemaan TRequestStatus tyyppistä palvelupyyntöä arvolla ”KRequestPending”. Palvelupyynnön tullessa muutetaan TRequestStatuksen arvoa ja aktiivinen vuorontaja saa tiedon tapahtumasta. Tämän jälkeen vuorontaja käy prioriteettijärjestyksessä läpi muuttuneen statuksen omaavia olioita ja valitsee suoritukseen korkeimmalla prioriteetilla olevan. [Harrison, 2003] [Stichbury, 2005]

2.1.2. Asiakas-palvelin-suunnittelumalli

Asiakas-palvelin-suunnittelumallilla on erityisen tärkeä merkitys Symbian OS -käyttöjärjestelmässä. Asiakas-palvelin-suunnittelumalli tarjoaa mahdollisuuden jaetun resurssin toteuttamiseen. Asiakas-palvelin-suunnittelumallissa tarkoitus on eristää resurssi sitä käyttäviltä ohjelmilta ja tarjota asynkroninen palvelu, jolla resurssia voidaan tasapuolisesti käsitellä eri ohjelmista.

Asiakas ja palvelin toimivat omissa säikeissään ja sijaitsevat eri prosesseissa, joten ne eivät pysty käsittelemään samaa muistialuetta. Käyttöjärjestelmän ydin huolehtii asiakkaan ja palvelimen välisten viestien välittämisestä. Suurin osa Symbian OS:n tarjoamista palveluista, kuten tiedosto-operaatiot ja tietoliikenneyhteydet, on toteutettu asiakas-palvelin-suunnitelumallin mukaisesti. Malli ei ole kuitenkaan rajattu vain käyttöjärjestelmän palveluiden toteuttamiseen vaan sitä voidaan erittäin hyvin käyttää myös omissa sovelluksissa.Tyypillisin tapaus on että mallin mukaiselle palvelimelle toteutetaan sekä palvelintoteutus käsittelemään resurssia että asiakastoteutus, johon palvelua käyttävä sovellus voi suoraan linkittää itsensä.

Symbian OS -käyttöjärjestelmässä palvelinten toteuttamiseen on valmiina kehys, jossa tarvittavat viestinvälitysmekanismit on toteutettu. Viestit välitetään istunnoissa, jotka käyttöjärjestelmän ydin luo. Asiakaspuolen istunnon toteutus periytetään luokasta RSessionBase ja palvelinpuolen toteutus luokasta Cserver, joka on alunperin Cactive-luokasta periytetty aktiivinen olio.

Asiakkaan ja palvelimen välinen viestintä tapahtuu luokan RMessage avulla.Viestit voidaan lähettää halutusta toimintatavasta riippuen synkrononisella tai asynkronisella versiolla Send()- tai SendReceive()-funktioista. Asiakaspään vastauksen odottamistapa riippuu täten valitusta tavasta. Palvelinpään toteutus käsittelee molemmat kutsutavat samalla tavoin. [Harrison, 2003] [Stichbury, 2005]

2.1.3. Verkkototeutukset Symbianissa

Verkkototeutukset Symbian OS -käyttöjärjestelmässä toteutetaan edellämainitun asiakas-palvelin-suunnittelumallin mukaisesti. Tietoliikenneyhteyksien toteuttamiseen käytetään RSocket-luokkaa. Koska ei voida tietää millon dataa lähetetään tai vastaanotetaan, toteutetaan pyynnöt asynkronisesti. Yhteydet muodostava luokka on tällöin aktiivinen olio, joka rekisteröityy aktiiviselle vuorontajalle odottamaan dataa.

Tiedonsiirto Rsocket-luokassa tapahtuu joko yksinkertaisilla Read()- ja Write()-funktioilla tai enemmän parametreja mahdollistavilla Send()- ja Recv()-funktioilla. Kaikki tiedonsiirtofunktiot ovat asynkronisia, mikä on otettava huomioon toteutettaessa sovellusta. [Harrison, 2003] [Stichbury, 2005]

Verkkototeutuksia implementoitaessa joudutaan usein tilanteeseen, jossa halutaan jakaa prosessin muistiavaruus toisten prosessien kanssa, esimerkiksi järjestelmäohjaimien ja ylemmän tason verkkoliitäntätoteutuksen välillä. Jaetun muistin käyttämiseen on syynä, että tietoliikennerajapinnoissa siirretään suuria määriä dataa ja tämän datan kopioiminen useampaan otteeseen olisi erittäin raskas operaatio. Symbian OS tukeekin RChunk-luokan avulla jaetun muistiavaruuden toteuttamista. Tällöin pystytään ohittamaan prosessikohtainen muistiavaruuden hallinta ja toteuttamaan erittäin tehokkaita ns. zero-copy-operaatioita jaetun muistin välityksellä. [Sales, 2005]

2.1.4. Symbian muistinhallinta

Symbian OS -käyttöjärjestelmä on suunniteltu toimimaan hyvin rajoitetuilla muistiresursseilla varustetuissa laitteissa. Tämä reunaehto tuo mukanaan sen, että muistinhallintaan on kiinnitetty suunnittelussa erityistä huomiota.

Tärkein muistinhallinnan ominaisuus Symbian OS -käyttöjärjestelmässä on siivouspino, jolla kontrolloidaan sovellusten virhetilanteissa tapahtuvaa muistivuotoa. Siivouspinon ideana on, että sovelluksen toteuttaja ilmoittaa pinoon sovelluksen varatessa muistia paikallisille tai dynaamisesti varatuille resurseille.

Pinossa olevat paikalliset resurssit vapautetaan automaattisesti, mutta dynaamisesti varatuille resursseille toteuttajan on erikseen ilmoitettava millä tapaa varattu resurssi vapautetaan virhetilanteen tapahtuessa. [Harrison, 2003] [Stichbury, 2005]

2.1.5. Virheiden käsittely

Symbian OS:n yksi erikoisuuksista on poikkeustapauksien käsittely. Kun Symbian OS kehitettiin, eivät poikkeukset olleet mukana virallisessa C++-standardissa, joten järjestelmään kehitettiin oma mekanismi poikkeusten käsittelyyn. Tämä ansahaarniska-mekanismi muistuttaa suuresti normaalia C++-standardin mukaista poikkeusten käsittelyä, mutta on operaatioiltaan hieman kevyempi.

Ansahaarniska-mekanismi tarjoaa kaksi makroa TRAP ja TRAPD, jotka ovat hyvin samanlaisia kuin C++-standardin try()-funktio. TRAP- tai TRAPD-lohkon sisällä tapahtuva virhetilanne käsitellään C++-standardin catch()-funktiota vastaavalla tavalla omassa lohkossaan.

Poikkeuksia Symbian OS -käyttöjärjestelmässä kutsutaan nimellä leave. Leave tapahtuu kutsumalla User::Leave()-funktiota, jonka vastine C++-standardissa on throw()-funktio. User::Leave()-funktiolle annetaan parametrina poikkeuskoodi, joka kuvaa virhetilanteen laadun. Poikkeuskoodeina voidaan käyttää käyttöjärjestelmän määrittelemiä

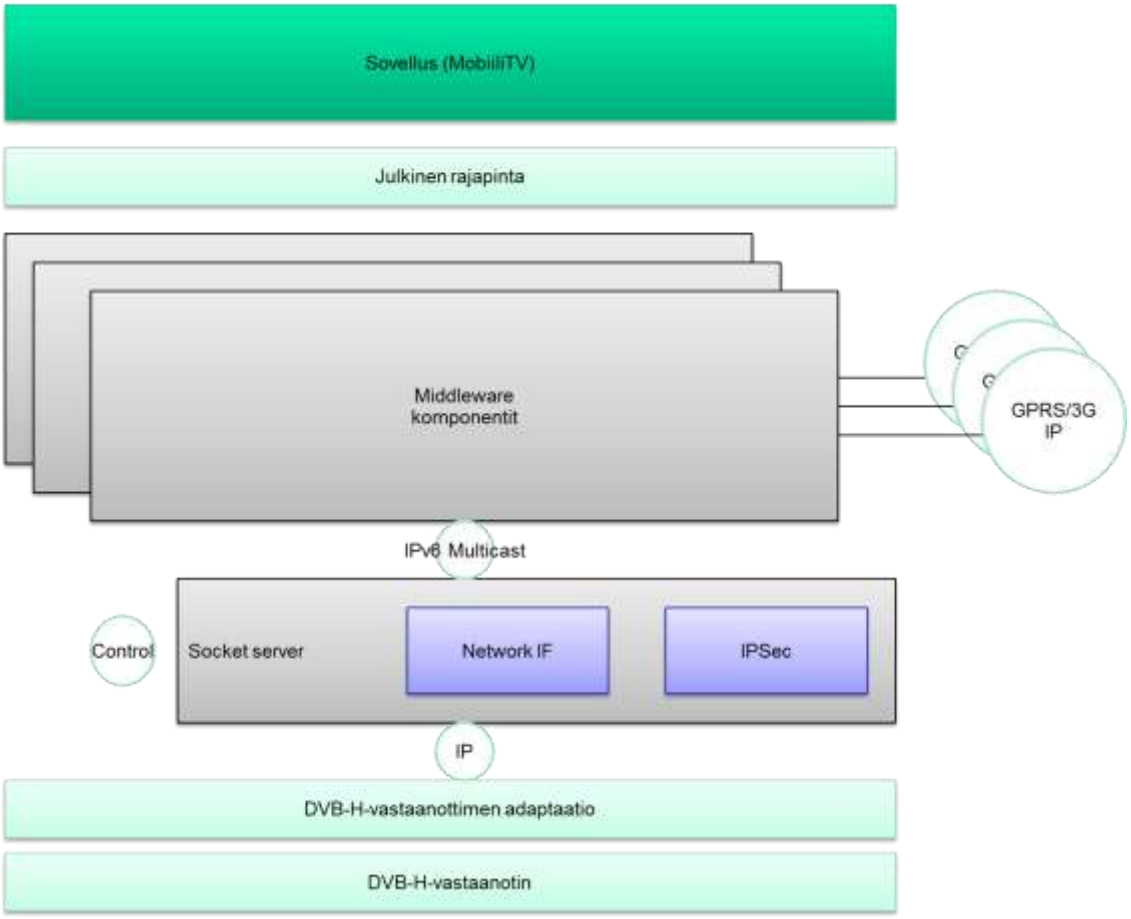
virhekoodeja tai sovelluksen itsemäärittelemää joukkoa paluukoodeja. [Harrison, 2003] [Stichbury, 2005]

2.2. DVB-H middleware

Middleware eli välikerrosohjelmistot ovat ohjelmistoja, jotka toteuttavat palveluita käyttöliittymän ja sovellusalueen tiedon tallennuksesta vastaavien ohjelmistojen välissä. Ne tarjoavat läpinäkyvät rajapinnat saatavilla oleviin palveluihin kyseessä olevan olevien ominaisuuksien ja resurssien puitteissa. Välikerrosohjelmistoille hyvin soveltuva rooli kokonaisjärjestelmässä on eri tietokantojen ja tietojärjestelmien yhdistäminen. Välikerrosohjelmisto voi keskustella useiden erilaisia tietovirtoja tarjoavien palveluiden kanssa ja yhdistää ne ylemmälle tasolle tarjottavaksi kokonaisuudeksi. Vastaavasti välikerrosohjelmisto toimii käyttöliittymäohjelmiston ja tietojärjestelmien välissä välittäen käyttöliittymän viestejä eteenpäin.

Välikerrosohjelmistojen tarkoituksena on irrottaa tietokantakäsittely, monimutkaiset laskenta-algoritmit, laitteistoadaptaation kanssa keskustelu ja rinnakkaisuuden hallinta käyttöliittymäohjelmista, jolloin kokonaisuudesta saadaan joustavampi, toimintavarmempi ja helpommin ylläpidettävä.

Projektin tarkoituksena oli toteuttaa tarvittavat välikerrosohjelmistot, jotta päätelaitteen DVB-H-vastaanottimen verkkoadaptaation tuottamat DVB-H-tietovirrat sekä muut ulkoiset palvelut pystytään tarjoamaan käytettäväksi MobiiliTV-asiakasohjelmalle eri Symbian OS -päätelaitteissa. Kuvassa 2.1 on esitelty päätelaitteen ohjelmistoarkkitehtuuri ja rajapinnat yleisellä tasolla.



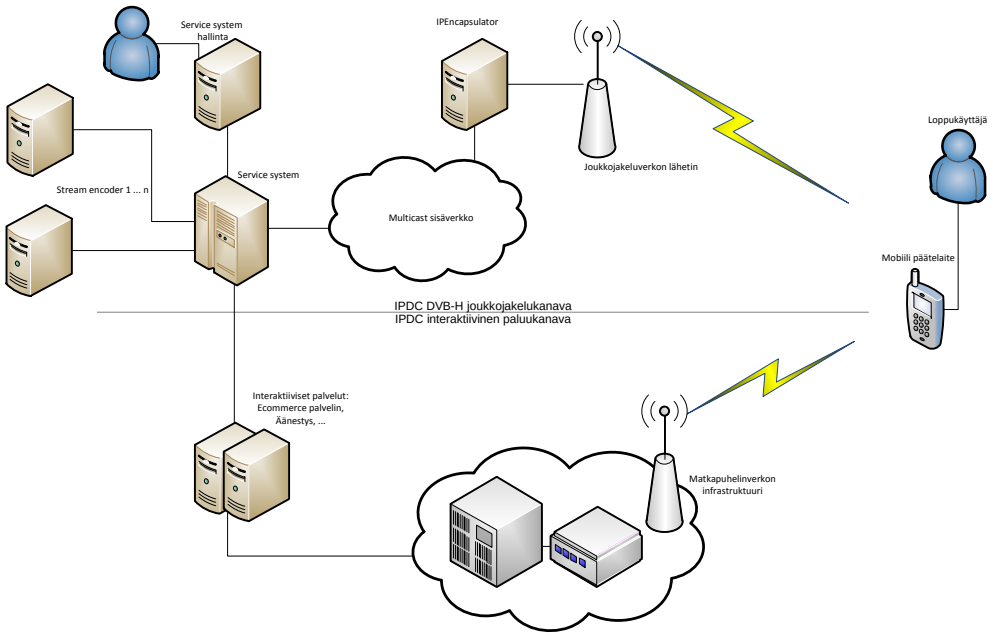
Kuva 2.1: Päätelaitteen ohjelmistoarkitehtuuri ja rajapinnat yleisellä tasolla

2.3. Kokonaisjärjestelmä

Projektissa toteutettu päätelaite on osa IPDC-päästä-päähän-arkkitehtuuria, joka koostuu useista eri palveluista, joiden tuottajina on useita eri alojen toimijoita. Ensimmäisenä päästä-päähän-arkkitehtuurin ketjussa ovat sisällöntuottajat, jotka tarjoavat digitaalisen sisältönsä kanavaoperaattorin käyttöön. Kanavaoperaattori tuottaa sisällöstä verkko-operaattorille lähetettävää tietovirtaa. Verkko-operaattori hallinnoi sisällön jakelua joukkojakelukanavaa pitkin päätelaitteeseen. Käyttäjän halutessa käyttää interaktiivisia palveluita, paluuviesti palaa matkapuhelinoperaattorin verkkoa pitkin kanava-operaattorin hallinnoimalle palvelimelle, josta on yhteys esimerkiksi laskutukseen. Interaktiivista paluukanavaa käyttäviä palveluita ovat esimerkiksi Ecommerce, äänestäminen, sekä katsojalukujen mittaus. Kuvassa 2.2 kuvataan päästä-päähän-arkitehtuurin palvelut.

Kuva 2.2: Päästä-päähän-arkkitehtuurin palvelut

Kuvassa 2.3 esitellään lähetysjärjestelmän oleelliset komponentit. Stream encoder -palvelimet vastaavat sisällöntuottajien tietovirtojen muunnoksista. Service System -palvelimet vastavat kokonaisjärjestelmän toimivuudesta kokoamalla sisällöntuottajien tietovirrat, interaktiiviset palvelut ja muun sisällön, esimerkiksi ohjelmaoppaan yhdeksi kokonaisuudeksi. IPEncapsulator-palvelin vastaa Service System -palvelinten tietovirtojen enkapsuloinnista IP-paketteihin joukkojaeltavaksi DVB-H-lähettimellä. Matkapuhelinverkon infrastuktuuri on matkapuhelinoperaattorin toteuttama kokonaisuus, johon tässä ei oteta kantaa.



Kuva 2.3: Päästä-päähän arkkitehtuurin verkkokuva [Faria et al., 2006]

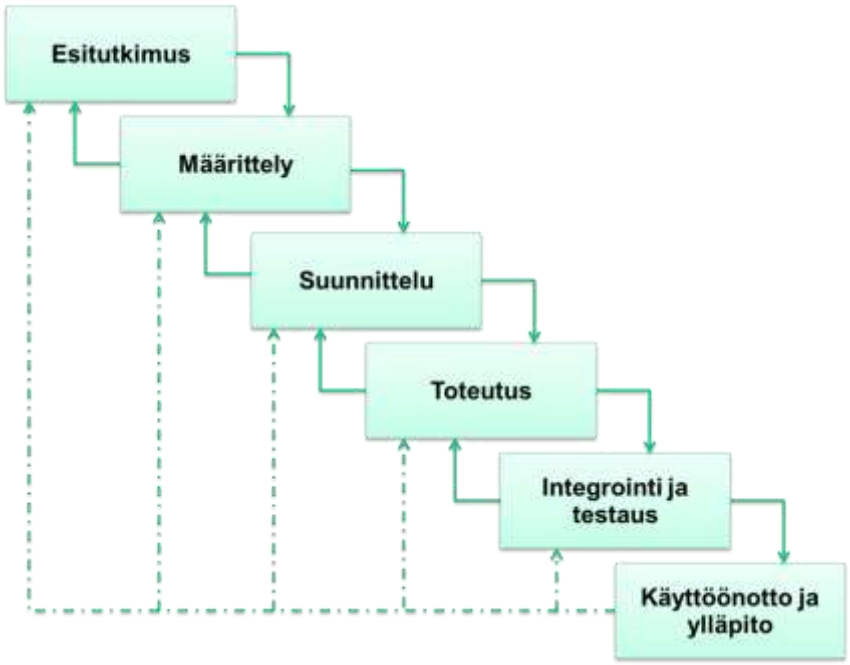
3. OHJELMISTOJÄRJESTELMIEN TESTAUS

Tässä luvussa käsitellään ohjelmistojärjestelmien testausta ja testauksen taustalla olevaa teoriaa.

3.1. Yleistä ohjelmistojen testauksesta

Ohjelmistojen testauksella tarkoitetaan ohjelmiston laadun parantamiseen tähtääviä toimenpiteitä. Testauksen päätarkoituksena on ohjelmistossa olevien virheiden löytäminen. Lisäksi testauksella pyritään saamaan riippumaton näkemys testattavan ohjelmiston käyttöönottoon liittyvistä riskeistä. Ohjelmiston testauksen aikana ohjelmistoa suoritetaan testisyötteillä ja pyritään aikaansaamaan virheitä. Virheellä (bug, mistake, error) tarkoitetaan ohjelmistojen testauksen yhteydessä poikkeamaa toiminnalliseen- tai tekniseen määrittelyyn verrattuna. Testitapaukset määritellään siten että annetulla testisyötteellä ohjelmiston sisäinen tila ja syötteet ovat suorituksen lopussa määrittelyn mukaisia.

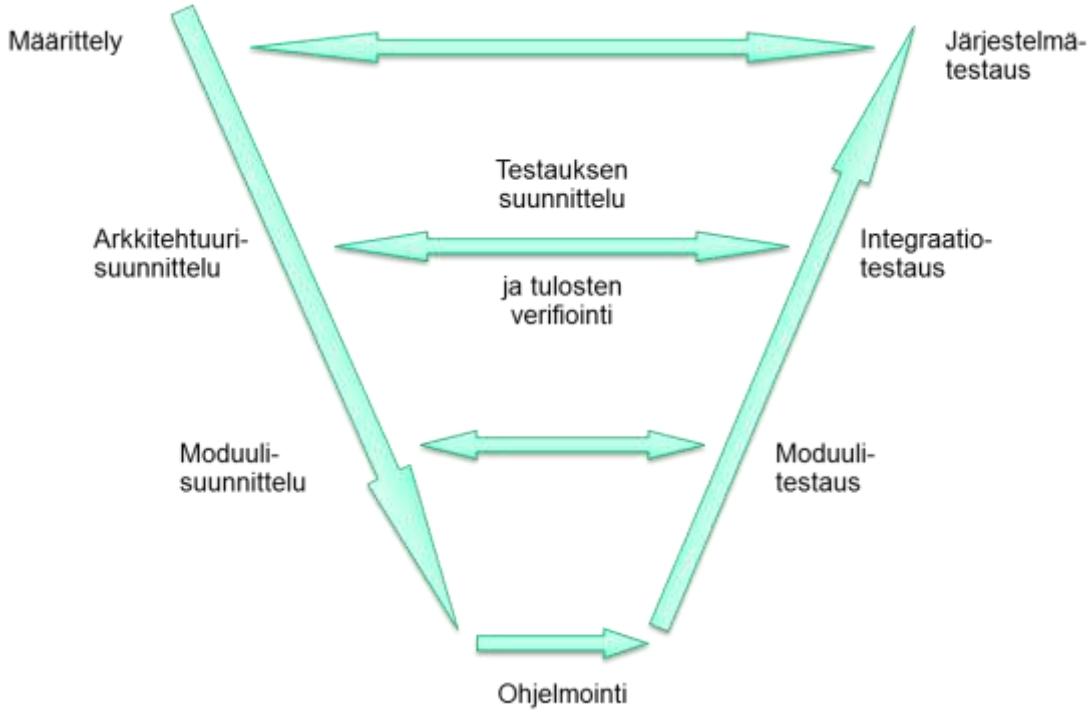
Ohjelmiston testaus voidaan myös määritellä prosessina validoida ja tarkistaa, että ohjelmisto vastaa teknisiä ja taloudellisia vaatimuksia, jotka suunnitteluvaiheessa sille on asetettu. Validoinnilla tarkoitetaan, että ohjelmisto toteuttaa määrittelyssä vaaditut ominaisuudet.



Kuva 3.1: Ohjelmistotuotannon vesiputousmalli [Haikala ja Märijärvi, 2006]

Ohjelmiston testaus voidaan valitusta testausmenetelmästä riippuen toteuttaa missä tahansa ohjelmistoprojektin vaiheessa. Ohjelmistojen testausprosessi voidaan jakaa testitapausten suunnitteluun, testiympäristön pystyttämiseen, testien ajamiseen sekä testitulosten tarkasteluun. Käytettäessä kuvassa 3.1 esiteltyä vesiputousmalliin perustuvaa prosessia suurin testauksen työmäärä on silloin kun projektin toteusvaihe on valmis. [Glenford, 2004]

Kuvassa 3.2 esitellyn testauksen V-mallin vasen reuna mukailee ohjelmistotuotannon vesiputousmallia. V-mallissa testitapausten suunnittelu etenee ohjelmiston suunnittelun mukana siten, että ohjelmiston osaa suunniteltaessa suunnitellaan myös miten sitä tullaan testaamaan. Hyväksymistestaus ja järjestelmätestaus suunnitellaan määrittelyvaiheessa, intergraatiotestaus suunniteluvaiheessa ja moduulitestaus moduulisuunnitteluvaiheessa.



Kuva 3.2: Testauksen V-malli [Burnstein, 2003]

Ohjelmistoprojektin resurssien puutteen vuoksi ohjelman toteuttaja kirjoittaa usein myös ohjelmiston moduulitestit. Uusissa ketterää ohjemistoprosessia noudattavissa projekteissa käytetään puhtaimmillaan Test Driven Developmentia eli testausohjattua kehitystä. Tämä tarkoittaa, että suuri osa testauksesta on ohjelmiston toteuttajan vastuulla jo ohjelmistoa kirjoitettaessa ja implementaatio aloitetaan kirjottamalla ensiksi testitapaus toteutettavalle ominaisuudelle. Usein kuitenkin käytetään testausohjatun kehityksen kevennettyä versiota, eli testausta toteutuksen lomassa.
Testituloksina ohjelmistojen testauksessa voidaan pitää testiraporttia, joka kertoo onnistuneiden ja epäonnistuneiden testien määrän. Lisäksi testiajoista voidaan kerätä erilaista kattavuustietoa.

Kattavuustiedolla voidaan tutkia miten hyvin tehdyt testitapaukset käyvät läpi testattua koodia. Erilaisia kattavuusmittoja ovat esimerkiksi lausekattavuus, päätöskattavuus, ehtokattavuus, päätös/ehtokattavuus ja moniehtokattavuun. Näistä yksinkertaisin eli lausekattavuus mittaa miten ohjelmakoodin yksittäiset rivit on suoritettu. Päätös ja päätös/ehtokattavuus mittaavat, miten ohjelmakoodin eri haarautumisehdot tulevat läpikäydyiksi. Moniehtokattavuus vaatii lisäksi, että kaikkien haarautumisehtojen kaikki yhdistelmät käydään lävitse. On huomattava, että harvoissa tilanteissa yksinkertaisimmankaan ehtokattavuuden tulokseksi saadaan täyttä sataa prosenttia.

Kattavuusmittojen kerääminen aloitetaan yleisesti moduulitestausvaiheessa, jolloin testauksen kattavuus on helppo saada korkealle tasolle. Siirryttäessä testaamaan suurempia kokonaisuuksia testauksen V-mallin mukaisesti, kattavuus yleensä laskee.
Lisäksi testituloksina voi olla erilaisia tehokkuusmittauksia, esimerkiksi järjestelmän resurssien kulutuksen seurantatietoa.

Ohjelmistojen testausta voidaan jaotella eri tasoille riippuen siitä miten suurta osaa ohjelmistosta kyseisellä hetkellä testataan. Testaamalla ohjelmistoa erilaisissa osakokonaisuuksissa voidaan ohjelmistossa piilevät virheet paikallistaa tiettyyn osaan ohjelmistoa. Ohjelmistojen testaus jaetaan moduulitestaukseen, integraatiotestaukseen ja järjestelmätestaukseen.

3.2. Moduulitestaus

Moduulitestauksessa eli yksikkötestauksessa testataan yksittäistä ohjelmiston moduulia. Yksittäinen moduli ei ole yleensä sellaisenaan suoritettavissa oleva osa vaan pieni 100-1000 rivin osuus toteutetusta ohjelmasta. Tämän takia joudutaan sen testausta varten rakentamaan testipeti, jossa moduulia voidaan testata erillään muusta ohjelmistosta. Moduulia kutsutaan tekemällä testiajuri, joka hoitaa moduulin tarjoamien palveluiden kutsumisen. Usein moduuli käyttää muiden moduulien tarjoamia palveluita. Tällöin nämä muut moduulit korvataan tynkämoduuleilla, jotka palauttavat yksinkertaisia paluuarvoja. Ideaalitapauksessa kukin testi on toisistaan riippumaton. [Burnstein, 2003]

Moduulitestauksessa testataan moduulin sisäistä toimintaa, paikallisia tietorakenteita, liittymäpintoja, eri suorituspolkuja ja virheiden käsittelyä. Moduulitestauksessa pystytään toteamaan toteutettujen algoritmien toteutusvirheitä vertaamalla moduulien toimintaa määriteltyyn. Moduulitestaus on niinsanottua lasilaatikkotestausta eli siinä testaus suoritetaan hyvinkin tarkasti lähdekoodia ja sen toimintaan tutkimalla.

Moduulitestaus on yleisesti melko mekaanista työtä, joten testaus pyritään automatisoimaan erilaisten testikehysten avulla.

3.3. Integraatiotestaus

Integraatiotestauksessa jo toteutettuja moduuleita integroidaan toimimaan keskenään ja testataan näiden välistä yhteistoimintaa. Testauksen tarkoituksena on testata miten moduulien väliset rajapinnat toimivat. Integraatiovaiheessa testaus voidaan suorittaa joko iteratiivisesti osa kerrassaan integroiden tai niinsanotulla ”big-bang” taktiikalla. Yleisesti ottaen iteratiivinen tapa on selkeämpi, koska virheellisten toteutusten paikantaminen on helpompaa pienemmästä määrästä integroitavia moduuleita. Iteratiivinen integrointi voidaan suorittaa kahdella tavalla: alhaalta kokoavasti tai ylhäältä jäsentävästi aloittaen. [Burnstein, 2003]

Yleisin tapa aloittaa integraatiotestaus on aloittaa alhaaltapäin. Tällöin integroidaan yleensä yksittäisiä moduulitestattuja moduuleita keskenään. Etuna tässä on, että voidaan ensiksi testata yksittäisten moduulien sisäiset rakenteet ja rajapinnat ja vasta tämän jälkeen modulien välisten rajapintojen yhteistoiminta.

Myös integraatiotestauksessa tarvitaan testiajureita korvaamaan hierarkiassa ylemmällä tasolla olevia komponentteja ja kutsumaan integroituja moduuleita. Mikäli aloitetaan integroimaan ylhäältäpäin voidaan tarvita tynkämoduuleita.

Myös integraatiotestaus on lasilaatikkotestausta, koska siinä tutkitaan ohjelmiston sisäistä toimintaa.

3.4. Järjestelmätestaus

Järjestelmätestauksessa testataan kokonaista toimivaa ohjelmistoa. Järjestelmätestauksessa pyritään tutkimaan täyttääkö valmis ohjelmisto asiakkaan sille asettamat vaatimukset. Järjestelmätestaus on niinsanottua mustalaatikkotestausta: siinä testataan ohjelmaa sen määrittelydokumentaation vaatimuksia vastaan. Järjestelmätestauksessa pyritään löytämään ristiriitoja alkuperäisten vaatimusten ja toteutetun ohjelmiston välillä.

Järjestelmätestauksessa testataan koko ohjelmaa ja myös siihen liittyviä ulkoisia osia, esimerkiksi matkapuhelinohjelmiston ollessa kyseessä testataan myös verkkoa ja verkossa olevia palvelimia ja näiden ohjelmistoja.

Järjestelmätestauksen kuuluu myös ei-toiminnallisten vaatimusten testausta. Ei-toiminnallisilla vaatimuksilla tarkoitetaan esimerkiksi tehokkuusmittauksia, kuormitusmittauksia ja luotettavuusmittauksia. [Burnstein, 2003]

3.5. Testauksen automatisoiminen

Ohjelmistojen koon ja vaatimusten kasvaessa alkaa testattavan ohjelmakoodin määrä kasvaa sellaisiin määriin, että merkittävä osa projektissa olevista resursseista kuluu testitapausten ajamiseen ja tulosten keräämiseen sekä analysointiin.

Testauksen automatisointiin on tämän takia tarjolla erilaisia työkaluja ja testikehyksiä. Näiden avulla on mahdollista ajaa suuria määriä samoja testitapauksia toistuvasti. Tämä testiajojen toistaminen mahdollistaa ohjelmaan tulleiden virheiden havaitsemisen ja helpottaa virheen sisältävän ohjelmakoodin löytämistä, koska on tutkittava ainoastaan epäonnistuneen testiajon välillä ohjelmakoodiin tulleet muutokset.

Toinen testauksen automatisoinnin tuoma hyöty on, että manuaalisessa testauksessa on aina mahdollisuus inhimillisten virheiden tapahtumiseen. Esimerkiksi testisekvenssiin voi tulla virhe testaajan suorittaessa suurta joukkoa testejä.

Myöskin testiajojen tuottamien tulosten analysointi helpottuu kun käytetyt ohjelmistot tarjoavat tilannetietoa ohjelmakoodin elinkaaresta. Automaattisen raportoinnin avulla pysytään tunnistamaan ohjelmakoodin krittisimmät kohdat. Näiden kohtien löytäminen on tärkeää, sillä tunnetusti virheet kasaantuvat monimutkaisimpiin osiin ohjelmakoodia.

Testauksen automatisoinnista saavutetaan usein suurin hyöty testauksen alemmilla tasoilla. Moduulitesteissä suoritetaan usein suuri joukko yksinkertaisia testitapauksia, joiden ajaminen manuaalisesti olisi usein erittäin työlästä. Ylemmillä testitasoilla testitapaukset yleensä ovat huomattavan paljon monimutkaisempia, joten testauksen automatisointiin käytettyä aikaa ei välttämättä pystytä saamaan takaisin.

4. VERKKOTEKNIIKAT

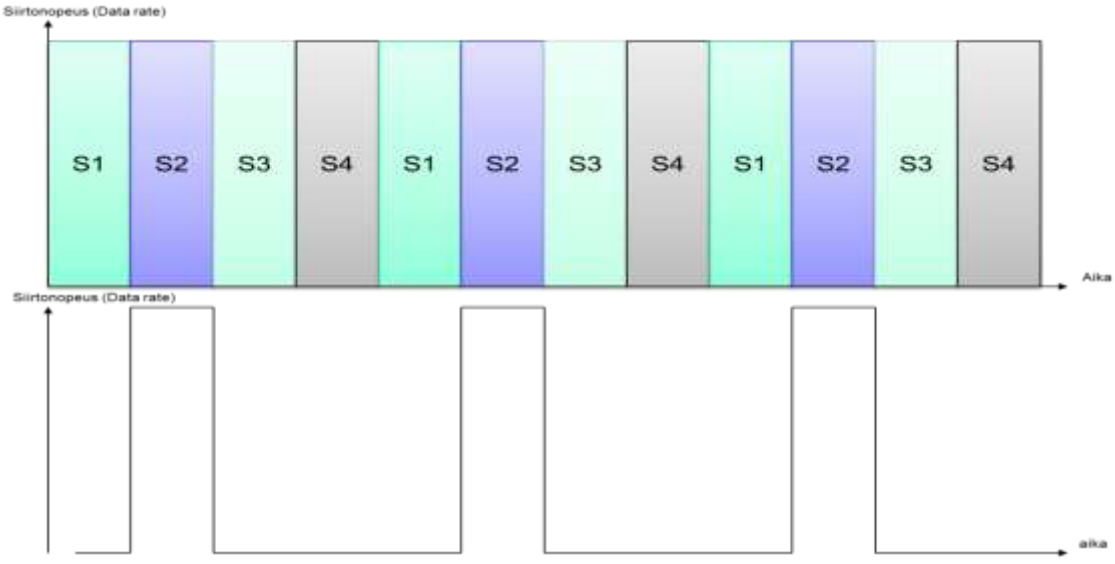
Tässä luvussa esitellään järjestelmässä käytössä olevia verkkotekniikoita: Ensiksi joukkojakelutekniikka DVB-H ja sen päällä toimiva IPDC. Toisena verkkorajapintana on interaktiivinen paluukanava, jonka toteutustekniikkana on pakettikytkentäinen tiedonsiirto käyttäen GPRS tai 3G/HSPA palveluita.

4.1. DVB-H

DVB-H on yksi digitaalisen television standardeista. DVB-H on erityisesti mobiileihin päätelaitteisiin tarkoitettu tekninen määrittely. DVB-H on perinteisen maanpäällisen digitaalisen television DVB-T:n ylijoukko, laajentaen DVB-T:n määrittelyä siltä osin, että määrittelyä voidaan soveltaa mobiileihin, akkukäyttöisiin päätelaitteisiin.

4.1.1. Virransäästö ja aikajakoisuus

DVB-H:n tärkeimmät erityispiirteet ovat virransäästöominaisuudet sekä parempi päätelaitteen liikkuvuuden sietokyky. Virransäästöominaisuudet perustuvat siirtoyhteyskerroksen aikajakoiseen protokollaan, joka tasaisen bittivirran sijaan lähettää korkealla bittinopeudella tietoa purskeissa. Purskeisella bittivirralla voidaan saavuttaa jopa 90-95% virransäästö verrattuna tasaiseen bittivirtaan. Aikajakoinen lähetystapa on pakollinen DVB-H-standardissa. Esimerkki lähettimen käyttöprofiilista aikajakoisessa tiedonsiirrossa löytyy kuvasta 4.1.

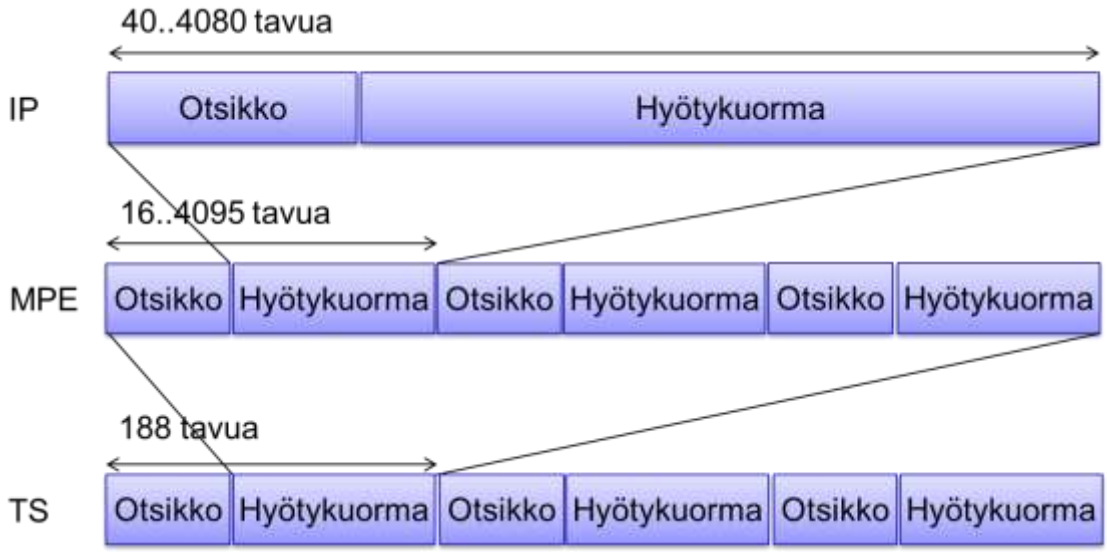


Kuva 4.1: Vastaanottimen käyttöprofiili aikajakoisen signaalin vastaanotossa [Faria et al., 2006]

4.1.2. Multiprotocol Encapsulation

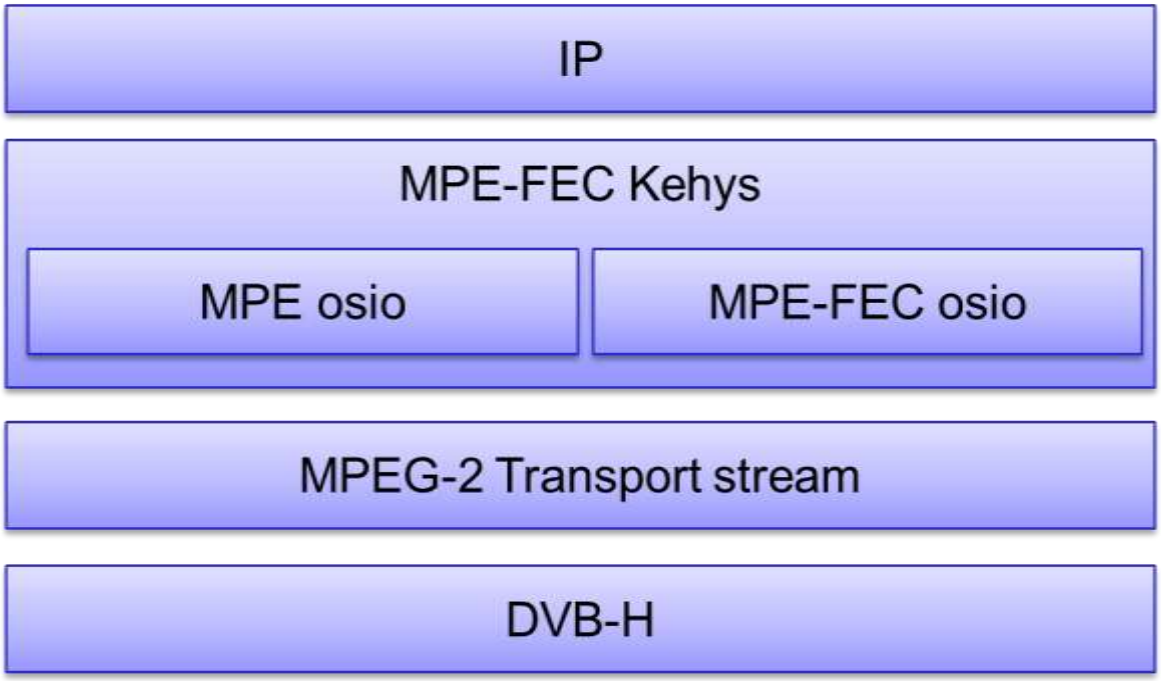
DVB-H on IP-pohjainen ratkaisu, joka toteutetaan MPE:n (Multi-Protocol Encapsulation) avulla. MPE mahdollistaa IP-liikenteen kuljettamisen DVB-verkkojen MPEG-2 transport streamin mukana kuvan 4.2 esittelemän kapseloinnin mukaisesti.

IP-paketti paketoidaan yhteen MPE-sektioon. MPE-virta laitetaan MPEG-2 ES -virtaan, joka koosttuu MPEG-2 TS paketeista ja PID-ohjelmatunnisteesta. [Faria et al., 2006]



Kuva 4.2: IP-paketin kapselointi MPEG-2 TS paketteihin [ETSI TS 102 470, 2006]

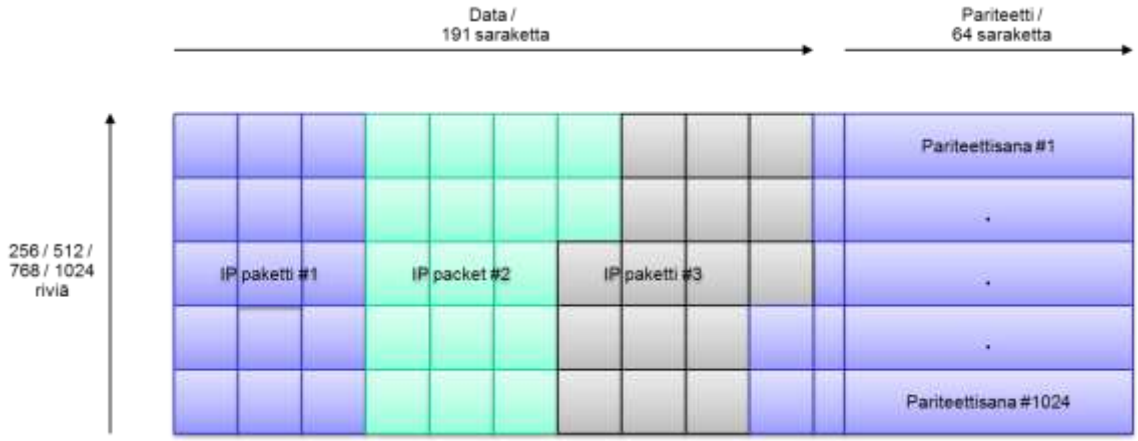
Päätelaitteen liikkuvuuden sietokykyä parannetaan DVB-H:n määrittelyssä lisäämällä linkkitason protokollaan MPE-FEC-kehykset, (Forward Error Correction) jolla parannettaan kantoaallon ja kohinan suhdetta ja doppler-siirtymän aiheuttaman häiriön sietokykyä mobiilissa välityskanavassa. Myös impulssihäiriöiden sietokyky paranee. MPE-FEC-kehyksessä IP-paketit suojataan Reed-Solomon-pariteettitiedolla. Kuvassa 4.3 on esitelty DVB-H-protokollapino MPE-FEC-kehyksineen.



Kuva 4.3: DVB-H-protokollapino [Faria et al., 2006] [ETSI TS 102 470, 2006]

MPE-FEC-kehys on matriisi, jossa on 255 saraketta ja valinnainen määrä rivejä. RS(255,191) koodaus määrittää että hyötydatalle on käytössä 191 riviä ja Reed-Solomon pariteetille 64 riviä. MPE-FEC-osan käyttö on valinnaista DVB-H standardin mukaan. (Kumar 2007)

Kuvan 4.4 kuvaamalla tavalla jäsennettään siirrettävä data MPE-FEC-kehykseen sarakkeittain ja tämän jälkeen pariteettikoodi lasketaan riveittäin. Matriisin solut kuvaavat siirrettäviä tavuja. Sarakkeittain ja riveittäin jäsentely mahdollistaa tiedonsiirrossa tapahtuvien virhepurskeiden havaitsemisen ja korjaamisen. Virhepurskeet jakautuvat useammalle riville joten useampi pariteettisana kattaa niiden alueella olevaa dataa ja virheet on helpompi korjata.



Kuva 4.4: FEC-Kehys [Faria et al., 2006]

4.1.3. Handover

Handover-siirtymällä tarkoitetaan liikkuvan päätelaitteen saumatonta siirtymistä verkkosolusta toiseen. DVB-H:n on tuettava päätelaitteen handover-siirtymiä. Näin pystytään takaamaan liikkuvan päätelaitteen vastaanottama katkeamaton tiedonsiirto. DVB-H-standardi määrittää TPS-bittejä välitysparametrien siirtämiseen. Välitysparametreja ovat esimerkiksi tieto kanavakoodauksesta, modulaatiosta sekä aikajakoisuuden ja MPE-FEC:n käytön signalointi. Handover-siirtymien kannalta tärkein TPS-bitti on solutunniste (Cell ID).

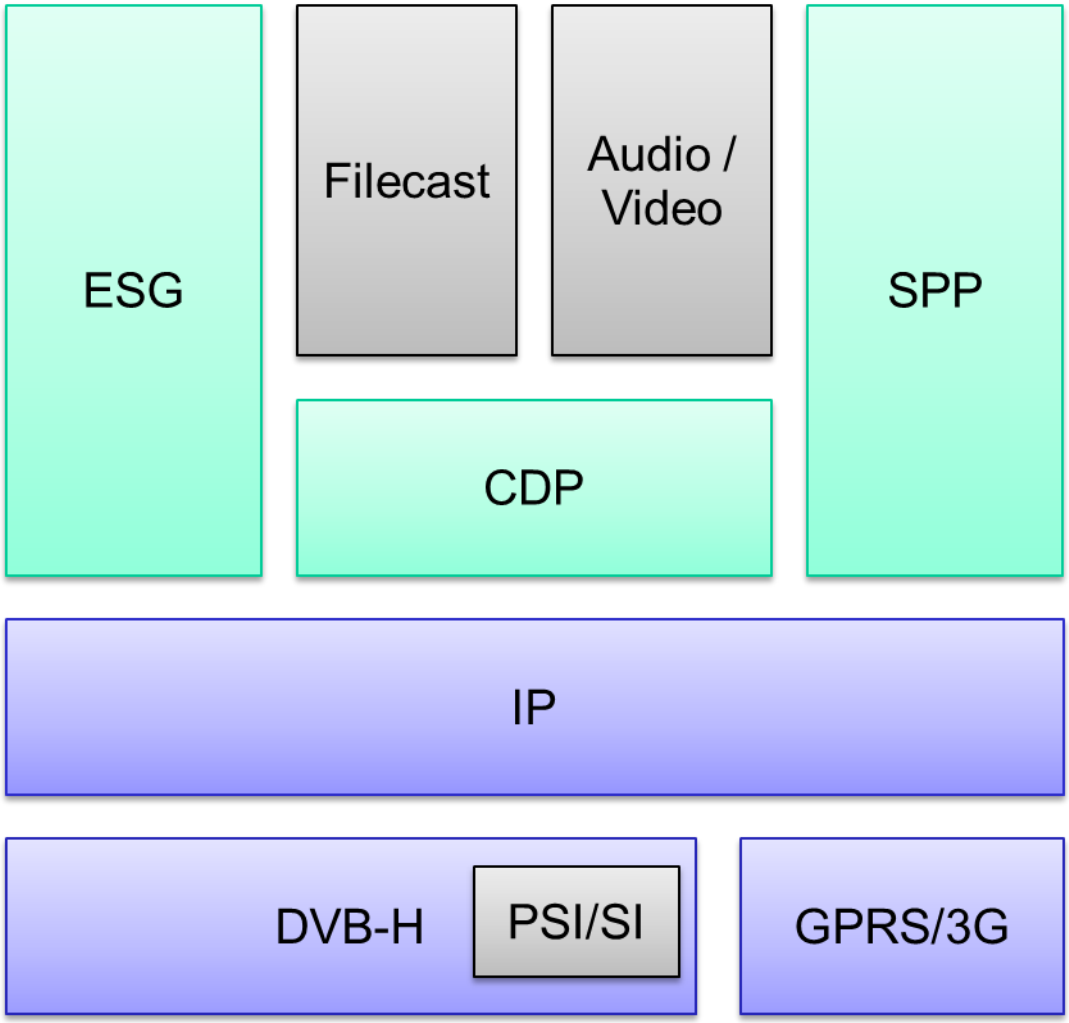
4.2. IPDC

IPDC eli IP Datacasting DVB-H-verkon yli on päästä-päähän-lähetysalusta kaikenlaiselle digitaaliselle sisällölle ja palveluille. Järjestelmä on optimoitu silmälläpitäen rajatuilla resursseilla varustettuja päätelaitteita. IP Datacastille luonteenomainen piirre on, että se yhdistää yksisuuntaisen DVB-joukkojakelun ja kaksisuuntaisen interaktiivisen paluukanavan. IPDC-alustalla mahdollistetaan konvergenssi joukkolähetyskanavan ja tietoliikennedomainien palveluiden välillä. IPDC sisältää määrittelyt palveluille, jotka tarjoavat tarvittavat komponentit kaupallisen mobiiliTV-palvelun tuottamiseen. DVB-IPDC on joukko järjestelmäpalveluiden määrittelyjä, jotka on suunniteltu DVB-H-protokollan kanssa käytettäviksi. Nämä palvelut määrittelevät mitä tarjotaan, kuinka se tarjotaan, kuinka se määritellään ja kuvataan sekä kuinka se suojataan. Kuvassa 4.5 on IPDC-määrittelyjen mukainen protokollapino. Määrittelyt kattavat systeemiarkkitehtuurin, käyttötapaukset, DVB PSI/SI-signaloinnin, elekronisen ohjelmaoppaan ESG (Electronic Service Guide), sisällönjakeluprotokollat CDP (Content Delivery Protocol) ja palveluiden ostamiseen ja suojaamiseen tarvittavat protokollat SPP (Service Purchase and Protection).

DVB-IPDC-verkon osoiteavaruutena voidaan käyttää sekä IPv4- että IPv6-osoiteavaruuksia. IPDC-tasolla jakelu perustuu multicast-tyyppiseen lähetykseen, jolloin vastaanottajan tarvitsee vain rekisteröityä kuuntelemaan jotakin tiettyä multicast-osoitetta. [ETSI TR 102 469, 2006]

4.2.1. PSI/SI

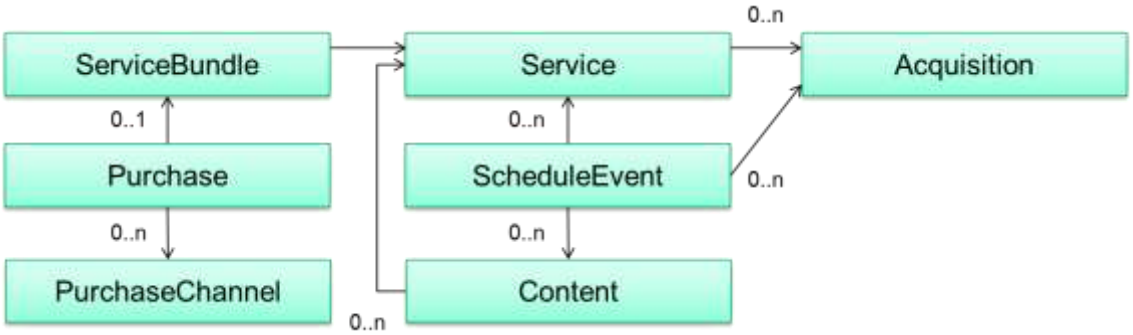
PSI/SI-taulut (Program Specific Information / Service Information) sisältävät ohjelmavirran sisällä lähetettävää tietoa ja ovat oleellinen osa digitaalista kuvansiirtoa. Taulut määrittelevät kanavapaketin sisällön ja niiden avulla vastaanotin osaa vastaanottaa haluttua kanavaa.



Kuva 4.5: IPDC-protokollapino [ETSI TS 102 472, 2006]

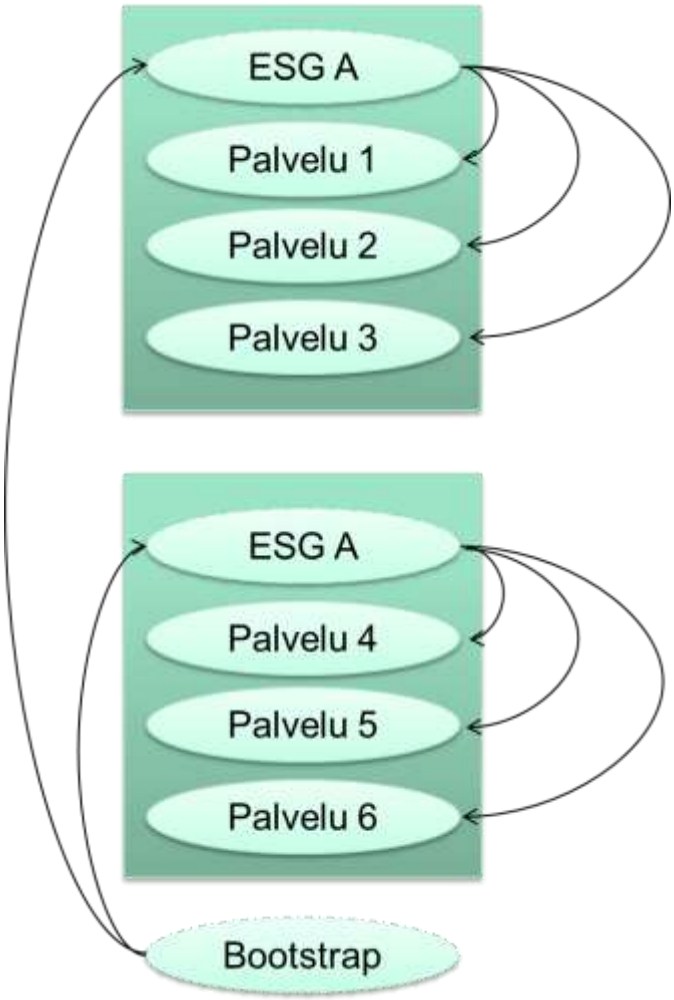
4.2.2. Sähköinen palveluopas

Sähköinen palveluopas (ESG) tarjoaa IPDC käyttäjille tiedon kyseisen alueen tarjolla olevista IPDC palveluista. Päästä-päähän-operaatio tarjoaa ESG-datan jakelun joko joukkojakeluverkon ylitse tai interaktiivista paluukanavaa käyttäen push/pull-moodissa. Normaalissa käyttötapauksessa ESG-data jaetaan Filecast-tietovirtoina joukkojakelukanavaa pitkin. ESG-data koostuu kahdesta osasta informaatiota. ESG-datan osat ovat sisältöä kuvaava osa (User attraction) ja sisällön saatavuutta kuvaava osa (Acquisition). ESG-data jaellaan pääosin XML-fragmentteina, mutta osa saatavuusinformaatiosta jaetaan SDP-tiedostoina, joita päätelaite tarvitsee löytääkseen palveluvirtoja ja konfiguroidakseen informaatiota käyttävät sovellukset oikein. Myös muun muotoisen informaation, kuten kuvatiedostojen, välittäminen on ESG-datan mukana mahdollista. Kuvassa 4.6 näkyy ESG kenttien väliset riippuvuussuhteet. Kuvassa oleva Acquisition-kenttä on saatavuuteen viittaava osa ja muut ovat sisältöä kuvaavia kenttiä.



Kuva 4.6: ESG-kenttien riippuvuussuhteet [ETSI TR 102 469, 2006]

Kuvassa 4.7 esitellään miten ESG datan jakelu tapahtuu bootstrappingin avulla. Ennalta sovitun IP-osoite ja portti -yhdistelmän kautta suoritetaan bootstrapping-yhteys, josta saadaan osoitteet muihin ESG-tauluihin. Päätelaite osaa automaatisesti vastaanottaa Filecast-tietovirtoja sovitusta osoitteesta ja pystyy näin lataamaan palveluoppaan tiedostot ja myöhemmässä vaiheessa päivityksiä niihin. [ETSI TR 102 469, 2006], [DVB Document A099, 2008]



Kuva 4.7: ESG bootstrapping [ETSI TR 102 469, 2006]

4.2.3. ESG- ja SDP-kuvaus

Edellä mainittiin, että päätelaitteelle jaetaan SDP-tiedostoina kuvaukset palveluiden saatavuudesta. SDP-tiedostot tarjoavat standardin mukaisen esitystavan siirrettävän tiedon muodosta. SDP-tiedostojen kuvaukset ovat protokollariippumattomia ja yleiskäytettäviä eri järjestelmissä. [SDP RFC, 2006]

SDP-tiedostojen avulla päätelaite pystyy löytämään osoitteet joista palveluvirrat löytyvät sekä asettamaan palvelun kuluttamiseen vaaditut konfiguraatioparametrit vastaanottosovellukseen. Seuraavana esitellään IDPC-suoratoistopalvelun (streaming) kuvaava SDP-tiedosto. [ETSI TS 102 472, 2006]

```
v=0
o=ghost 2890844526 2890842807 IN IP4 192.168.10.10
s=IPDC SDP Example
i=Example of IPDC streaming SDP file
u=http://www.example.com/ae600
e=ghost@mailserver.example.com
c=IN IP6 FF1E:03AD::7F2E:172A:1E24
t=3034423619 3042462419
b=AS:77
a=source-filter: incl IN IP6 * 2001:210:1:2:240:96FF:FE25:8EC9
a=min-buffer-time:500
m=video 4002 RTP/AVP 96
b=TIAS:62000
b=RR:0
b=RS:600
a=maxprate:17
a=avg-br:48000
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42A01E; packetization-mode=1;
sprop-parameter-sets=Z0IACpZTBYmI,aMljiA==
m=audio 4004 RTP/AVP 98
b=TIAS:15120
b=RR:0
b=RS:600
a=maxprate:10
a=avg-br:14000
a=rtpmap:98 AMR/8000
a=fmtp:98 octet-align=1
```

Lisäksi tiedostossa voitaisiin kuvata vaadittu autentikaatiometodi, kuten seuraavassa esimerkissä käytetty SRTP-protokolla. [ETSI TR 102 474, 2006]

```
a=SRTPAuthentication:n
a=SRTPROCTxRate:1
```

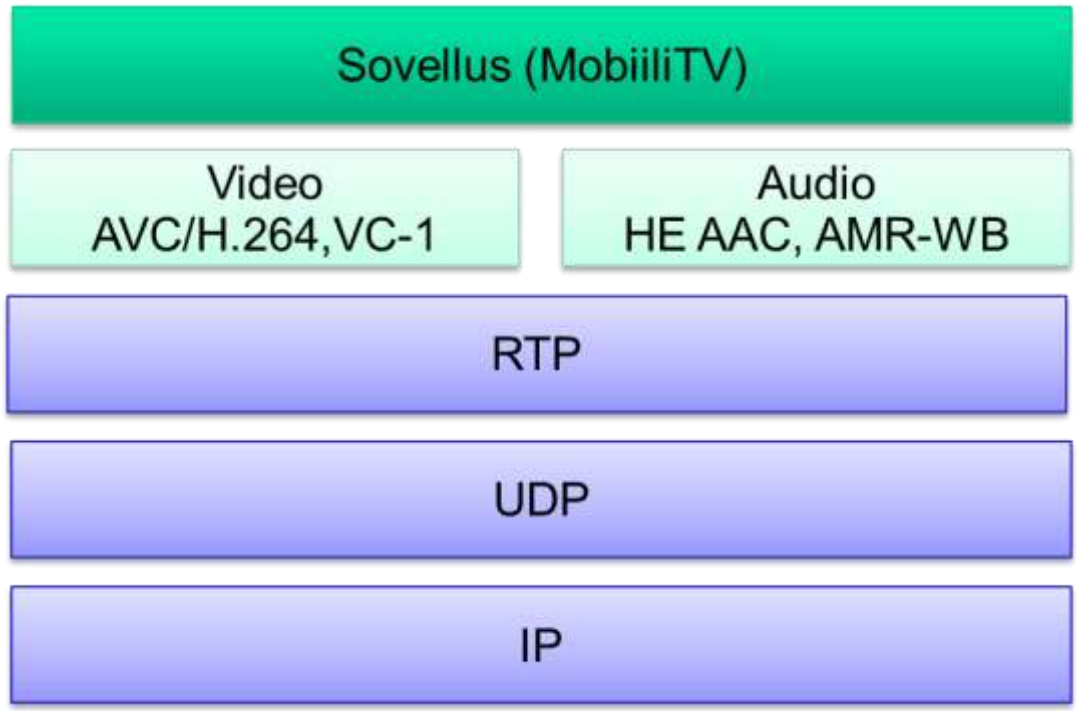
4.2.4. Sisällönjakelu

Sisällönjakelulla IPDC-järjestelmässä tarkoitetaan palveluiden sisältöä, jonka päätelaite vastaanottaa joukkojakeluverkosta. Palveluiden sisältö voi kuulua kahteen eri kategoriaan. Nämä kategoriat ovat suoratoisto (streaming) ja Filecasting. Kuvassa 4.8 esitellään miten sisällönjakeluprotokollat sijoittuvat IPDC-protokollapinoon.

Streamingilla eli suoratoistolla tarkoitetaan reaaliaikaista lähetyksen vastaanottamista siten, että tietovirrasta ei jää kopiota vastaanottajalle vaan sen sisältö käytetään välittömästi. Esimerkki suoratoistosta on MobiiliTV-lähetyksen vastaanotto, jossa vastaanotettua tietovirtaa ei ole tarkoitettu erityisemmin puskuroitavaksi tai tallennettavaksi pidemmäksi ajaksi. Suoratoistovirran vastaanottaja voi alkaa vastaanottaa virtaa millä tahansa ajanhetkellä. Vastaanottaja voi myös lopettaa vastaanottamisen milloin haluaa. Reaaliaikaisten suoratoistopalveluiden kuljettamiseen käytetään RTP-protokollaa. RTP tarjoaa suoratoistovirtaa vastaanottavalle sovellukselle tiedon kehysten sisältämästä tietotyypistä, kehysten ajastuksesta, niiden mahdollisesta häviämisestä ja kehysten sisältämän datan lähteistä.

RTP-protokollan sekvenssinumerot ja aikaleimat tarjoavat mahdollisuuden pakettien järjestyksen ja sisäisen järjestyksen ylläpitämiseen. Näin pystytään mediaa purettaessa käsittelemään vastaanotetut paketit oikeassa järjestyksessä ja synkronoimaan eri virroissa saapuneet audio ja video. Identifioimalla kehysten sisäinen tietotyyppi voidaan signaloida ylemmälle tasolle, mitä kodekkia median purkamiseen tulisi käyttää. Standardin määrittelemiä kodekkeja audiolle ja videolle ovat H.264/AVC, VC-1, HE AAC v2 ja AMR-WB+. [ETSI TS 102 005, 2007]

Kuvassa 4.7 on esitelty miten audio- ja videokodekit sijoittuvat IPDC-protokollapinoon suoratoistosovelluksen tapauksessa.



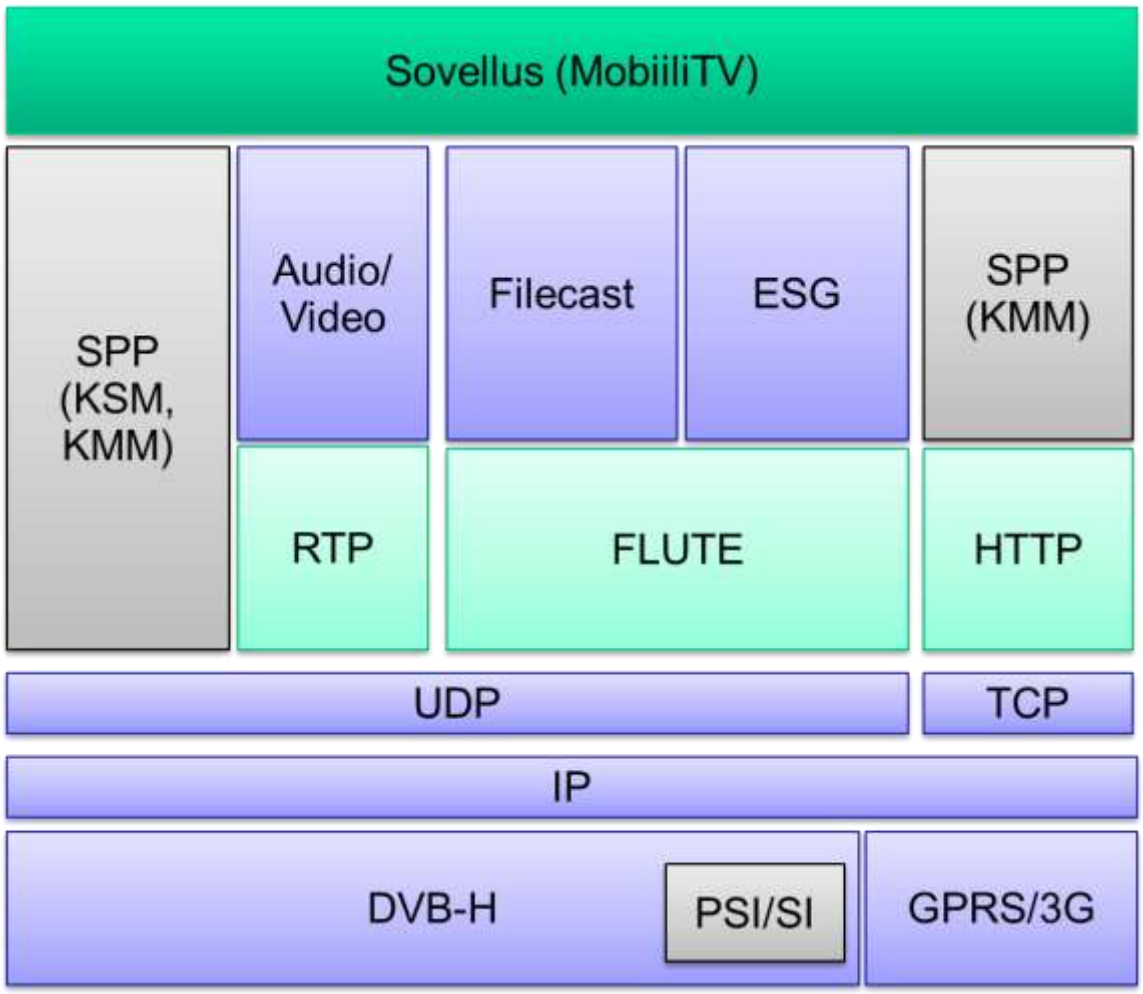
Kuva 4.7: Audio- ja videokodekkien sijoittuminen protokollapinoon [ETSI TS 102 005, 2007]

H.264/AVC on ITU:n ja ISO:n yhteinen videostandardi ja se on valittu IPDC-järjestelmän videokoodekiksi. H.264/AVC:n erityisominaisuutena on ylivoimainen pakkausteho verrattuna aikasempiin koodekkistandardeihin H.264/AVC on myös kohtuullisen mukautuva erilaisiin verkko-olosuhteisiin ja yhdistettynä MPE-FEC enkapsulointiin se pystyy selviämään melko korkeastakin virhetiheydestä.

Filecast on suoratoistopalvelun vastakohta reaaliaikaisuuden suhteen. Filecastingissa tietovirran sisältämä tiedosto vastaanotetaan ja tallennetaan päätelaitteelle kokonaisuudessaan myöhempää käyttöä varten. Käytössä on yksisuuntainen tiedostonjakeluprotokolla, jolla on pystyttävä takaamaan tiedoston integriteetti. Operaattori voi lisäksi määritellä interaktiivista paluukanavaa käyttävän korjausmekanismin. Filecast tiedostojenjakeluun on määritelty käytettäväksi FLUTE-protokollaa. FLUTE perustuu asynkroniseen kerrostuneen koodauksen protokollaan (ALC), johon yhdistetty virheenkorjausosa (FEC) tarjoaa luotettavuutta siirtoon. Protokolla tarjoaa asynkronisen yhtäaikaisen siirron yhdeltä lähettäjältä rajattomalle määrälle vastaanottajia. FLUTE toimii UDP protokollan päällä ja on IP-versiosta riippumaton. [ETSI TS 102 472, 2006]

Yksi Filecastingin erityissovelluksista on ESG-palveluoppaan lähettäminen päätelaitteille. ESG-lohkoja kuljetetaan tiedostoina FLUTE istunnon siirto olioissa (Transport Object). ESG-tiedostot signaloidaan istunnon tiedostonkuvaustaulussa (FDT) asettamalla Content-Type-attribuuttityypiksi ”application/vnd.dvb.esgcontainer”. Myöhemmässä vaiheessa ESG-palveluoppaan päivityksestä ilmoitetaan päätelaitteille

päivittämällä Filecast-istunnon siirto-olion tunnistetta (TOI) ja tiedostonkuvaustaulun kenttiä. [DVB Document A099, 2008]



Kuva 4.8: IPDC Baseline sisällönjakelun protokollapino [ETSI TS 102 472, 2006]

4.2.5. Palveluiden ostaminen ja suojaus

IPDC-määrittely sisältää SPP-osuuden (Service Purchase and Protection), joka kuvaa palveluiden ostamista ja suojausta. Määrittely sisältää kolme metodia sisällön/palveluiden salaamiseen (service based, service/pay-per-view ja pay-per-view) ja kaksi järjestelmämääritystä (OSF ja 18Crypt) muihin tarvittaviin osiin. Molemmat järjestelmät perustuvat yhteisen avaimen hierarkkiseen malliin.

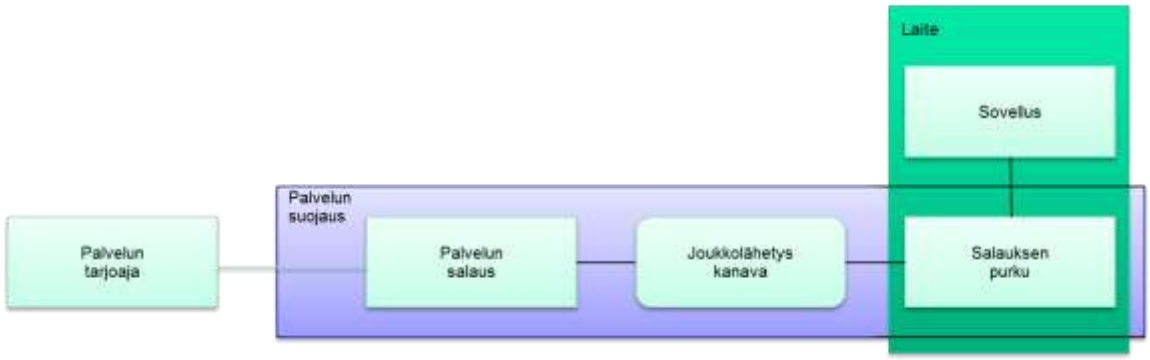
Palveluiden ostamiseen ja suojaamiseen tarkoitettu arkkitehtuuri jakautuu neljään tasoon. Nämä tasot ovat rekisteröityminen, oikeuksien hallinta, avainvirta sekä, sisällön ja palvelun suojaus.

Sisällön ja palvelun suojaus voidaan tarpeen mukaan suorittaa eri tasoilla: linkkitasolla käytetään IPSec:iä, istuntotasolla SRTP:tä ja sisältötasolla IsmaCryp:iä. Määrittelyn

mukaiset järjestelmät ja päätelaitteet toteuttavat kaikki tasot, joten näitä voidaan käyttää vapaavalintaisina yhdistelminä.

SPP-järjestelmä tarjoaa menetelmät, joilla pystytään signaloimaan päätelaitteelle vastaanotettavan palvelun käyttämä avaimenhallintajärjestelmä (KMS) sekä käytössä olevat avainvirrat. Tiedot signaloidaan ESG:n kentissä päätelaitteelle, joka parsii tiedot ja pystyy niiden perusteella suorittamaan palvelun oston ja hankkimaan avaimet sisällön avaamiseen. Joukkojakelupalveluita varten ESG-data sisältää ostettavan palvelun tunnistekentän (ServiceID). Osa tiedoista signaloidaan SDP-istunnonkuvaustiedostossa. SDP:n kentissä kuljetetaan tieto avainvirtaviesteistä (KSM). Näissä kentissä kerrotaan osoite UDP-virralle, josta kyseisen palvelun avainvirta löytyy. Avaimenhallintaviestienvirta (KMM) on puolestaan UDP-virta, joka kantaa käyttöoikeusolioita (RO). Kolmas SDP:ssä kuljetettava tieto auttaa päätelaitetta yhdistämään mikä vastaanotettavista KSM-virroista on olennainen kunkin vastaanotettavan palveluvirran kannalta. Useammat palvelut saattavat viitata samaan KSM-virtaan ja voivat näin jakaa keskenään saman liikenteen salausavaimen (TEK).

Open Mobile Alliancen digitaalisen sisällönhallinnan järjestelmän versiota 2.0 (OMA DRM 2.0) käytetään järjestelmänä oikeuksien hallintaan. OMA DRM 2.0 kuvaa palveluiden oikeuksien hallinnan, palveluihin liittyvät avaimet ja cryptograafisen suojauksen kyseisille avaimille. Yleisin tapaus on, että OMA DRM 2.0 hallinnoi oikeuksia laitteeseen tallennettujen tiedostojen käyttöön. OMA DRM 2.0 tarjoaa lisäksi keinot oikeuksien hallintaan joukkojakelukanavan ylitse. OMA DRM 2.0 tukeutuu interaktiiviseen paluukanavaan laitteiden rekisteröinnissä ja luotettavaan oikeuksien jakeluun.



Kuva 4.9: Päästä-päähän-suojaus [ETSI TS 102 474, 2006]

Kuvassa 4.9 esitellään palvelunsuojauksen sijoittuminen kokonaisarkkitehtuuriin. IPSec-protokollaa käyttämällä voidaan ratkaisu pitää täysin sisältöformaatista riippumattomana, kun taas SRTP:tä ja ISMACryp:iä käyttävät ratkaisut tarjoavat vaihtoehdot suojauksen toteuttamiseen siirtokerroksella.

IPSec on standardi tapa pitää palvelun salaus vastaanottavissa laitteissa IP-tasolla. Näin pystytään takaamaan suojauksen näkymättömyys vastaanottaville sovelluksille.

SRTP on standardi tapa suorittaa palvelun salaus vastaanottaville laitteille siirtokerroksella. SRTP:tä voidaan käyttää salaamaan kaikkea suoratoistosisältöä, joka voidaan tarvittaessa tallentaa vastaanottajalaitteeseen myöhempää purkua varten.

ISMACryp on myös standardi tapa salata palvelu siirtokerroksella. ISMACryp tarjoaa päästä-päähän-salauksen suojatun sisällön siirtämiseen ja tallentamiseen. [ETSI TR 102 474, 2006]

4.3. Interaktiivinen paluukanava

Koska DVB-H on joukkojakeluun pohjautuva verkkotekniikka, tarvitsee mobiilipäätelaite DVB-H:n lisäksi jonkun toisen tekniikan, jolla voidaan toteuttaa interaktiivisten palveluiden vaatima paluukanava. Luonnollinen valinta tähän mobiilipäätelaitteessa on pakettikytkentäinen tiedonsiirto käyttäen laitteen tarjoamia GPRS-, 3G ja HSPA-tekniikoita. Yksinkertaisiin käyttäjän kuittausta vaativiin sovelluksiin voidaan myös käyttää SMS-viesteihin pohjautuvaa paluukanavaa.

4.3.1. SMS-viesteihin perustuva paluukanava

SMS-viestit, eli tekstiviestit ovat 160 merkkiä pitkiä matkapuhelinverkossa välitettäviä viestejä. Viestit lähetetään viestikeskuksen kautta. Viestikeskus pitää viestin tallessa kunnes kohdepäätelaite on sen vastaanottanut.

4.3.2. Pakettikytkentäinen mobiilitiedonsiirto

Pakettikytkennällä tarkoitetaan tiedonsiirtoa, jossa data jaetaan pieniksi paketeiksi siirtoa varten. Paketit lähetetään käyttäen kanavaa, joka varataan siirrolle vain tarvittavaksi ajaksi. Mobiiliverkoissa toimittaessa pakettikytkentäisen tiedonsiirtopalvelun tarjoavat GPRS (General Packet Radio System) ja HSPA (High Speed Packet Access). Edellämainitut ovat matkapuhelintekniikoita, jotka tarjoavat mahdollisuuden pakettikytkentäiseen tiedonsiirtoon. GPRS tekniikka on GSM (Global System for Mobile Communications) järjestelmän, eli niinsanotun toisen sukupolven digitaalisen matkapuhelinverkon, pakettidatapalvelu. 3G ja HSPA taas ovat kolmannen sukupolven UMTS-verkkojärjestelmiä (Universal Mobile Telecommunications System). Kaikki edellämainitut järjestelmät tarjoavat riittävät ominaisuudet interaktiivisen paluukanavan toteuttamiseen. Käytännön erot eri tekniikoiden välillä näkyvät käyttäjälle toteutuvina tiedonsiirtonopeuksina, jotka ovat UMTS-järjestelmään pohjautuvissa toteutuksissa huomattavasti korkeampia. [HSPA Whitepaper, 2007]

4.3.3. HTTP-protokolla

Pakettikytkentäisen tiedonsiirron avulla toteutetun interaktiivisen paluukanavan päällä on luonnollista käyttää IP-protokollaa ja tämän päällä käytettäväksi protokollaksi sopii HTTP tai HTTPS. HTTPS-protokolla eroaa tavallisesta HTTP-protokollasta siten, että siinä yhteys salataan käyttäen SSL tai TLS -protokollaa. Sisällön kuljettaminen tapahtuu molemmissa protokollissa samalla tavalla.

HTTP ja HTTPS sisältävät erilaisia metodeja viestien välittämiseenn Näistä oleellisimmat ovat GET- ja POST-metodit. GET-metodilla luetaan yksittäinen sivu tai resurssi ja valtaosa internetissä kulkevasta HTTP-liikenteestä käyttää GET-metodia. GET-metodi on määritelty olevan idempotentti, eli perättäisten GET-pyyntöjen tulisi palauttaa sama tulos. POST-metodilla pystytään välittämään asiakkaalta palvelimelle suurempia määriä tietoa kuin GET-metodilla. POST-pyyntöjä käytetään yleisesti esimerkiksi erilaisten lomakkeiden tietojen siirtoon palvelimelle.

HTTP-kyselyihin vastauksena saadaan viesti, joka koostuu statuskoodista ja mahdollisista liitteistä viestin rungossa. Statuskoodi on numerosarja, jonka ensimmäinen numero kategorisoi viestin eri luokkiin. Standardin mukaisia luokkia on viisi: 100 - “informational”, 200 - “success”, 300 - “redirection”, 400 - “client error” ja 500 - “server error”. Miten asiakas reagoi saatuun vastausviestiin, riippuu saadusta koodista sekä mahdollisista liitteistä viestin rungossa.

Yleisin käyttötapaus IPDC-kontekstissa on siirtää erilaista XML-koodattua tietoa HTTP-protokollan ylitse. XML-standardi on sääntöjä informaation esittämiseksi rakenteisina dokumentteina, joissa tiedon merkitys on kuvattavissa tiedon sekaan. XML-kieltä käytetään sekä formaattina tiedonvälitykseen järjestelmien välillä että formaattina dokumenttien tallentamiseen. [XML RFC, 2008]

Pyynnöt lähetetään HTTP POST -metodilla käyttäen sisältötyyppiä ”application/xml”, mikäli sisältönä on vain yksi viesti tai käyttäen tyyppiä ”multipart/mixed”, jos kuormana on useampi eri viesti. Tarvittaessa kuorman sisältöä voidaan tarkentaa esimerkiksi käyttöoikeuksien hankinta viestin (ROAP) tapauksessa käyttämällä ”application/vnd.oma.drm.roap-trigger+xml” tyyppiä. Paluuviestit kyselyille lähetetään aina paluukoodin liitteenä. Normaalissa toiminnassa palvelin vastaa asiakkaan pyyntöihin HTTP/200 paluukoodilla. [HTTP RFC,1999]

4.4. Tiedonsiirtokanavan laatu

Tiedonsiirtokanavan laatua voidaan kuvata useammalla eri suureella, joista yksi on kanavan bittivirhesuhde BER. BER ilmaisee todennäköisyyden sille, että yksittäinen kanavassa kulkeva bitti kääntyy, eli nollabitti muuttuu ykköseksi tai ykkösbitti

nollabitiksi. Tiedonsiirtokanavan laatu riippuu käytetyistä tekniikoista ja siirtokanavan tyypistä. DVB-H-verkon tapauksessa käytössä on langaton siirtotie, joten lähtökohtaisesti BER on siirtotielle korkea. [Harju, 2000]

Tiedonsiirtokanavan laatua arvoioitaessa muita kiinnostavia suureita ovat signaalin taso, joka kuvaa vastaanotetun signaalin tasoa ja kantoaallon signaalisuhde CNR, joka kuvaa hyötysignaalin suhdetta taustakohinaan. Modulaation virhesuhde MER kuvaa myös signaalin laatua vertaamalla konstellaatiopisteiden sijaintia teoreettisiin arvoihin.

5. TESTAUKSEN KEHITTÄMISEN TARVE

Projektin testausprosessin kehitys todettiin tarpeelliseksi, koska ohjelmiston testaus muodostaa huomattavan pullokaulan koko tuotteen julkaisuprosessissa.

Tässä luvussa käydään läpi eri ongelmia, joita vanha testausprosessi tuotti, sekä puutteita, joita haluttiin korjata testausprosessin näkyvyydessä ja ennalta arvioitavuudessa. Lisäksi luvussa selvitetään ketterään ohjelmistotuotantoprosessiin siirtymisen aiheuttamia haasteita.

5.1. Moduuli ja integraatiotestauksen kehittämistarpeet

Projektissa kehitetty järjestelmä koostuu useammasta Symbian välikerrosohjelmistokomponentista. Jokaiselle komponentille on kyseisen komponentin kehittäjän toimesta kirjoitettu joukko yksikkötestejä. Testit on toteutettu käyttäen Symbian OS -testaukseen suunniteltua STIF-testikehystä, joka tarjoaa palvelut komponentin testaukseen. Testit ajettiin manuaalisesti joko Symbian-emulaattorissa tai päätelaitteessa. Manuaalinen ajo vaati kuitenkin testaajalta suurta työpanosta ja tuhlasi arvokasta kehitystyöaikaa toistuviin toimenpiteisiin, kuten järjestelmän kääntäminen, komponentin sekä sen testien kääntäminen ja testien manuaalinen ajaminen. Testauksen jälkeen oli testiraportti vielä kopioitava erikseen talteen. Ilmeni siis selkeää tarvetta saada testausta automatisoitua, jotta turhasta työstä päästäisiin eroon. Suuren työmääränsä takia testaus jäi toteutettavaksi vasta kun kyseinen ohjelmistokomponentti oli mahdollisimman valmis. Siksi testauksen valmistuminen muodostui pullonkaulaksi prosessissa ja sitä tehtiin komponentin valmistuttua kiireessä ennen julkaisua.

Testitapausten ajamiseen suunniteltu ohjelmisto oli lisäksi hyvin kankea käyttää. STIF S60 UI -käyttöliittymä ei tarjonnut testaajalle kovin erityisiä mahdollisuuksia muuttaa testijärjestelyitä, vaan kaikki testijärjestelyjen muutokset täytyi tehdä käytännössä asetustiedostoihin ja siirtää testijärjestelmään. Kuvasta 5.1 nähdään, kuinka STIF S60 UI koostuu useasta eri valikosta joiden välillä testaaja joutuu siirtymään käynnistääkseen testitapauksia ja nähdäkseen niiden tulokset.

Kuva 5.1: STIF-UI valikoita [STIF UI, 2009]

Lisäksi itse testiajon seuraaminen oli hyvin rajoittunutta ja virhetilanteen tapahtuessa kaikki tilannetieto täytyi tallettaa manuaalisesti myöhäisempää tutkimista varten.

Koska testausta suoritettiin harvoin ja testiajojen välillä ohjelmakoodiin tuli suuriakin lisäyksiä ja muutoksia, oli testiajoissa mahdollisesti ilmenevät poikkeamat usein vaikeaa kohdistaa tiettyyn muutokseen ohjelmakoodissa. Pahimmassa tapauksessa yksittäiset poikkeamat peittivät alleen muita virheitä toteutuksessaan. Prosessiin tarvittiin mahdollisuus havaita regressio ohjelmakoodissa mahdollisimman aikaisessa vaiheessa.

Aikaisemman testijärjestelyn tuotoksena oli vain testiraportti, josta ilmeni läpimenneet ja epäonnistuneet testitapaukset. Ohjelmakoodin määrän kasvaessa projektissa oli tarpeen arvioida testitapausten laatua erilaisilla kattavuusmittauksilla. Näiden kattavuusmittausten ja muiden tilastotietojen kerääminen manuaalisissa testiajoissa olisi kuormittanut testaajaa entisestään.

5.2. Järjestelmätestauksen kehittämistarpeet

Järjestelmätason testauksen kehittämisessä oli tarpeen validoida järjestelmän toimintaa erilaisissa tilanteissa. Varsinkin ohjelmiston varhaisessa kehitysvaiheessa oli erilaisten päästä-päähän-tietoliikenneyhteyksien testaus mahdotonta, koska joko palvelinpään toteutus tai välikerrosohjelmistoa käyttävän asiakasohjelman toteutus puuttui.

Tarvetta oli myös saada erilaisia testitapauksia toistettavaan muotoon, mikä taas käytännössä poissulki mahdollisuuden käyttää tuotantotason palvelinyhteyksiä.

Lisäksi järjestelmästä oli tarvetta saada kerättyä erilaista tehokkuusmittaustietoa.

5.3. Siirtyminen ketterään ohjelmistoprosessiin

Yksittäisistä syistä ehkä tärkein syy testausprosessin kehittämiselle oli projektin siirtyminen perinteisen vesiputousmallin ja iteratiivisen mallin välimuodosta ketterään ohjelmistoprosessiin.

Aikasemmin projektin yhden iteraatiosyklin pituus oli yli kuukausi. Siirryttäessä ketterään prosessiin se lyheni parhaimmillaan reilun viikon pituiseen sprinttiin. Näin lyhyessä iteraatiosyklissä ei ole mahdollista käyttää ohjelmiston testaukseen yhtä tai jopa kahta työpäivää, koska testauksen osuus itse syklistä olisi suhteettoman suuri.

Testauksen kehittäminen mahdollisti projektin sujuvan siirtymisen ketterään sykliin ilman että suurinta osaa syklien työtunneista käytettäisiin testaukseen. Automaattisten testiajojen huolehtiessa, että versionhallinnassa oleva ohjelmakoodi on kullakin ajanhetkellä kattavasti testattua, voidaan lyhyessäkin syklissä olla suhteellisen varmoja, että julkaisuhetkellä ohjelmakoodi täyttää sille asetetut vaatimukset.

6. TESTAUKSEN KEHITTÄMINEN

Tässä luvussa kerrotaan käytännön toimista miten testausta on kehitetty projektin kuluessa ja käytetyistä työkaluista.

6.1. Testattava järjestelmä

Testattava järjestelmä koostuu joukosta Symbian välikerrosohjelmistoja, jotka toteuttavat IPDC-määritelmän mukaiset palvelut. Kuvassa 6.1 esitellään järjestelmän arkkitehtuuria tarkemmin.

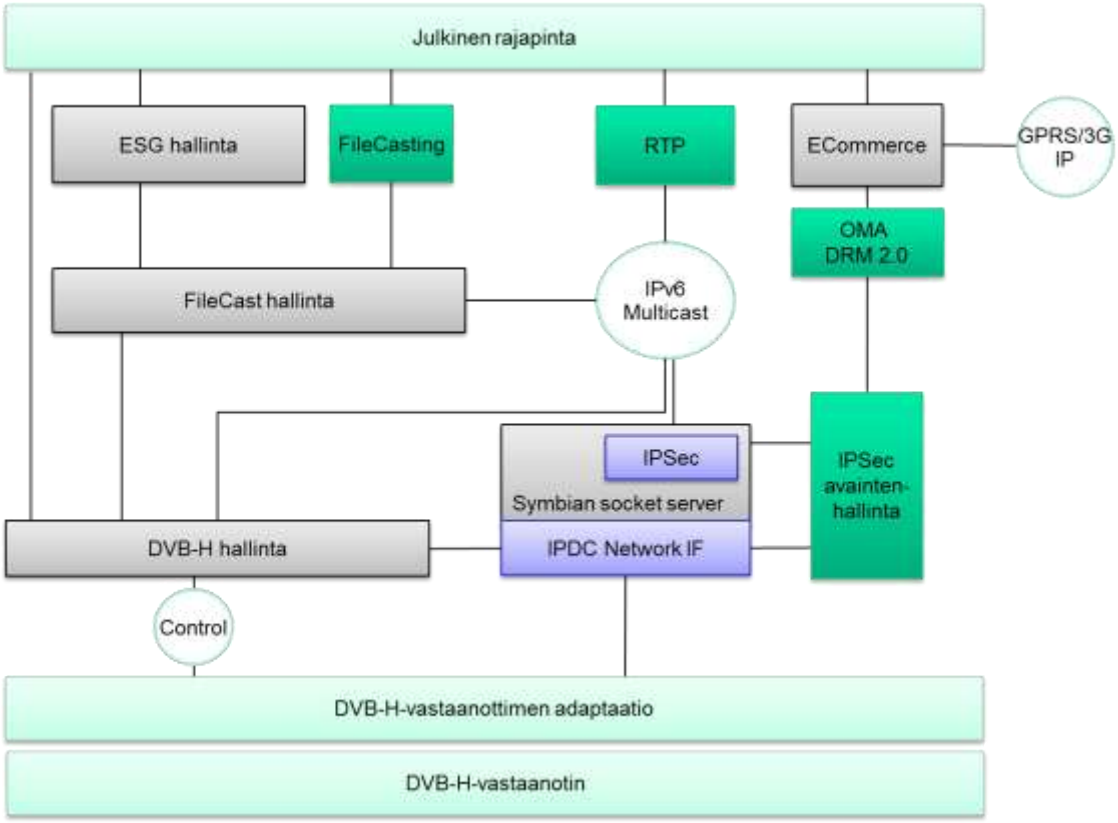
Järjestelmän keskeinen osa on DVB-H-hallintakomponentti, joka vastaa sovelluskerroksen ja laitteistoadaptaation välisestä kommunikaatiosta. DVB-H-hallintakomponentti hoitaa esimerkiksi vastaanottimen aktivoinnin, verkon hakemisen ja kanavan asettamisen. DVB-H-hallintakomponentti vastaa myös osittain muiden komponenttien kontrolloinnista.

IPDC NetworkIF -komponentti vastaa verkkoliikenteen abstrahoinnista. NetworkIF toteuttaa rajapinnan, jonka avulla pystytään luomaan IP-filttereitä multicast-osoitteisiin ja vastaanottamaan näiden kautta verkkoliikennettä. NetworkIF sisältää priorisaatiologiikan IP-filttereiden hallinnointiin. NetworkIF tarjoaa multicast-socket-yhteyksienhallinnan muille välikerroskomponenteille ja julkisen rajapinnan kautta MobiiliTV-asiakasohjelmalle. NetworkIF toteuttaa myös rajapinnan IPSec-yhteyksien luomisen.

FileCast-hallinta vastaa FileCast-lähetteiden lataamisesta ja käsittelystä. FileCast-hallinta tarjoaa rajapinnan ESG-hallinnalle sähköisen ohjelmaoppaan ylläpitoon.

ESG-hallinta vastaa sähköisen ohjelmaoppaan ylläpidosta, johon sisältyy ESG-datan parsiminen sekä ESG-päivitysten hakeminen.

ECommerce on esimerkki interaktiivisen palvelun toteuttavasta komponentista. Ecommercen rajapintoina ovat digitaalisten oikeuksien hallinta (DRM) ja julkisen rajapinnan lisäksi interaktiivisen paluukanavan toteuttavat matkapuhelinverkon rajapinnat.



Kuva 6.1: Testattavan järjestelmän arkkitehtuuri

6.2. Modulitestauksen automatisointi

Komponenttien kehittäjät olivat jo valmiiksi kirjoittaneet STIF-testikehystä käyttäviä testitapauksia omille komponenteilleen, joten lähtökohtaisesti tarkoitus oli, että automaatioon tultaisiin käyttämään myös STIF-testikehystä. Tällä vältettiin turhaa työtä, ettei testejä jouduttu uudelleenkirjoittamaan ja taattiin yhteensopivuus asiakkaan järjestelmiin.

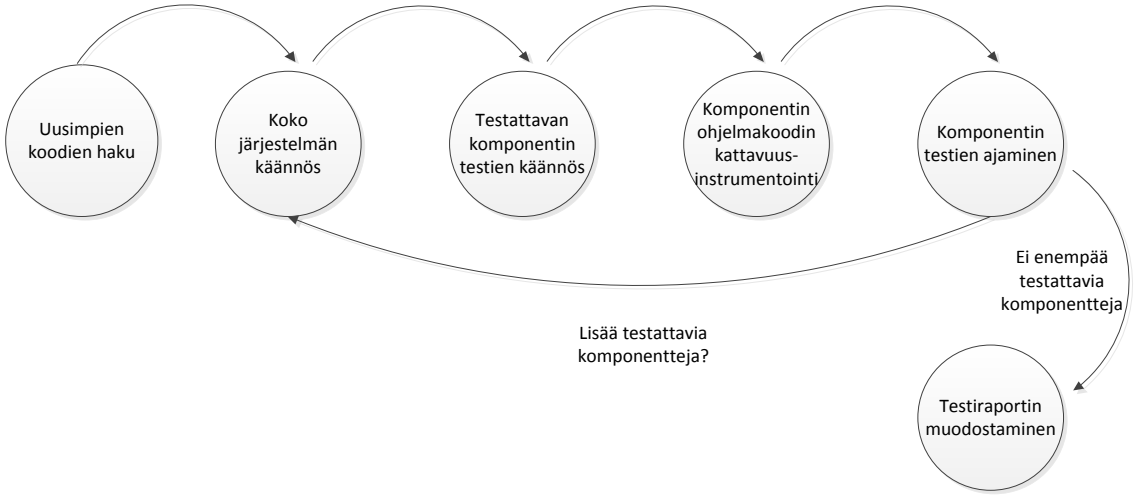
STIF-testikehys tarjoaa manuaalisten GUI- ja konsoli-käyttöliittymien lisäksi komentoriviltä ajettavan ATS-rajapinnan. ATS-rajapinta tarjoaa mahdollisuuden tehdä vapaavalintaisella ohjelmointikielellä komentosarjoja ja rakentaa näiden ympärille testausautomaatiota. Automaation toteutus päätettiin toteuttaa Perl-kielisillä komentosarjoilla ATS-rajapinnan ympärille. Ennen toteutustavan valintaa tehtiin pikainen katselmus myös valmiisiin testiautomaatiota ja jatkuvaa integrointia tarjoaviin ohjelmistoihin kuten Cruise Control, mutta näiden yhdistäminen STIF-testikehykseen ei olisi todennäköisesti ollut kovin helppoa.

Näin ollen päädyttiin kirjoittamaan testausautomaatiojärjestelmä itse. Yhtenä tavoitteena järjestelmän testauksen kehittämisessä oli kattavuustyökalun käyttöönotto ja testien kattavuuden automaattinen arviointi.

6.2.1. Nightlyrunner

Testausautomaatiojärjestelmän nimeksi tuli Nightlyrunner. Nimi kuvastaa järjestelmän lähtökohtaa suorittaa kaikki halutut testitapaukset kerran vuorokaudessa yöllä tapahtuvassa testiajossa. Jatkuvan intergroinnin periaatteiden mukaisesti toimittaessa tavoite olisi, että jokainen versionhallintaan tehtävä muutos aiheuttaisi muutoksen välittömän testauksen, mutta järjestelmää suunniteltaessa päätettiin, että kerran yössä suoritettava koko järjestelmän käännös ja kaikkien komponenttien testaus riittäisi.

Nightlyrunnerin toiminta perustuu kuuteen eri tilaan joiden riippuvuussuhteet esitellään kuvassa 6.2.



Kuva 6.2: Nightlyrunner testiajon tilat

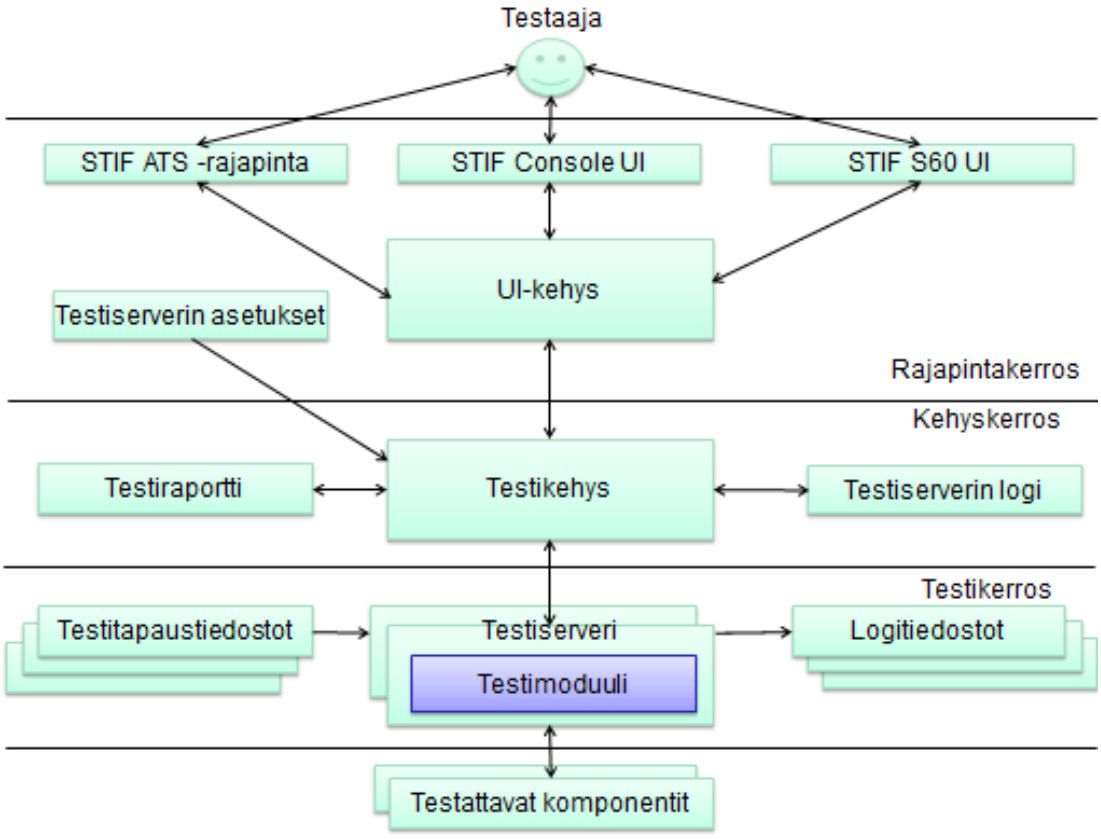
Nigtlyrunner toimii siirtymällä näiden tilojen välillä.

1. Testikehys hakee versionhallinnasta komponenttien uusimmat ohjelmakoodit sekä asiakkaan palvelimelta uusimman järjestelmän uusimman baseline-version.
2. Testikehys kääntää koko järjestelmän.
3. Testikehys kääntää testattavan komponentin testitapaukset.
4. Testikehys instrumentoi testattavan komponentin kattavuustyökalulla.
5. Testikehys ajaa testitapaukset.
    i. Mikäli testiajossa on määritelty lisää testattavia komponentteja, palataan aikaisempaan tilaan (Koko järjestelmän käännös) ja suoritetaan järjestelmän uudelleen käännös, jolla taataan testiajojen riippumattomuus aikaisemmista ajoista.
    ii. Mikäli testiajossa ei ole enää testattavia komponentteja, jatketaan seuraavaan tilaan (Testiraportin muodostaminen).
6. Testikehys kokoaa testiajoista raportin ja lähettää sen sähköpostilla kehittäjille.

6.2.2. STIF-testikehys

STIF-testikehys on Nokian kehittämä Symbian-ohjelmistojen testaukseen tarkoitettu testikehys, joka tarjoaa testaajalle testihaarniskan niin käyttöliittymä- kuin serveritasonkin komponenttien testaukseen. STIF-testikehys tarjoaa työkalut yksikkö- ja integraatiotason testauksen toteuttamiseen.

STIF:n tarjoamat komponentit voidaan jaotella karkeasti kolmeen tasoon. Kuvassa 6.3 esitellään testikehyksen eri komponenttien jakautuminen näille tasoille. Ensimmäinen kerros on rajapintakerros, joka tarjoaa kolme vaihtoehtoista rajapintaa testitapausten ajamiseen. Näistä kaksi on tarkoitettu manuaaliseen testaukseen joko emulaattori tai päätelaiteympäristössä. Kolmas rajapinta on ATS-rajapinta, joka tarjoaa mahdollisuuden komentosarjojen kirjoittamiseen ja automaattisten testitapausten toteuttamiseen.



Kuva 6.3: STIF-testikehyksen kerrokset [STIF, 2010]

Seuraavalla tasolla, eli kehyskerroksella on testimoottori (testengine), joka huolehtii testiajojen suorittamisesta sekä kokoaa raportit testiajoista.

Alimmalla tasolla, eli testikerroksella on yksi tai useampi testiserveri-instanssi, jonka avulla testattavaa komponenttia kutsutaan. Testiserveri-instanssi hoitaa myös yksittäisen testitapauksen suorittamisen testitapaustiedostossa määritetyllä tavalla.

STIF-testikehys on julkisesti saatavilla Symbian Foundationin kautta. [STIF, 2010]

6.2.3. Kattavuusmittaukset

Ohjelmistoa testattaessa yksi olennainen osa on kattavuusmittaukset. Kattavuusmittauksilla pystytään mittaamaan kuinka tehokkaasti suoritetut testitapaukset käyvät läpi ajettua koodia. Tässä projektissa kattavuusmittausten suorittamiseen päätettiin käyttää Bullseye coverage -työkalua. Työkalua ei ollut käytetty aikaisemmin yrityksessämme, joten osana testauksen automatisointia suoritettiin myös kattavuustyökalun evaluointi.

Bullseyen toiminta perustuu testattavan koodin instrumentointiin käännösvaiheessa, jolloin Bullseye lisää koodiin omia makrojaan, joilla pystytään seuraamaan ohjelman suorituksen kulkua. Koodia ajettaessa makrot kirjoittavat testilogia, josta pystytään jälkikäteen luomaan kattavuusraportteja sekä selaamaan testitapausten kattamia reittejä ohjelmakoodissa.

6.2.4. Raportointi

Nightlytesterin testiajon tuloksena syntyy komponenttikohtainen raportti, joka lähetetään kyseisen komponentin implementoijalle sähköpostiin. Raportista selviää komponentin käännöksessä mahdollisesti tapahtuneet virheet tai varoitukset, kattavuusmitat sekä testitapausten läpimeno. Lisäksi Nightlytester kerää verkkolevylle jokaisesta testiajosta instrumentaatiotiedot ja kaikki tulosteet, mikäli jälkikäteen halutaan tutkia tarkemmin testiajojen tuloksia.

Tärkeä osa testausta on testitulosten raportointi mahdollisimman luettavassa muodossa. Nightlytester parsii STIF-kehyksen raportit sekä Bullseyen kattavuusraportit yhdeksi raportiksi. Koostetusta raportista pystyy yhdellä silmäyksellä näkemään kokonaiskuvan testiajoista. Säännöllinen ja selkeä raportointi tarjoaa mahdollisuuden huomata mahdollinen regressio ensitilassa.

Valmiit raportit on myös mahdollista lähettää suoraan asiakkaalle ja näin parannetaan asiakkaan näkyvyyttä projektiin.

Esimerkki Nightytesterin testiraportista löytyy liitteenä.

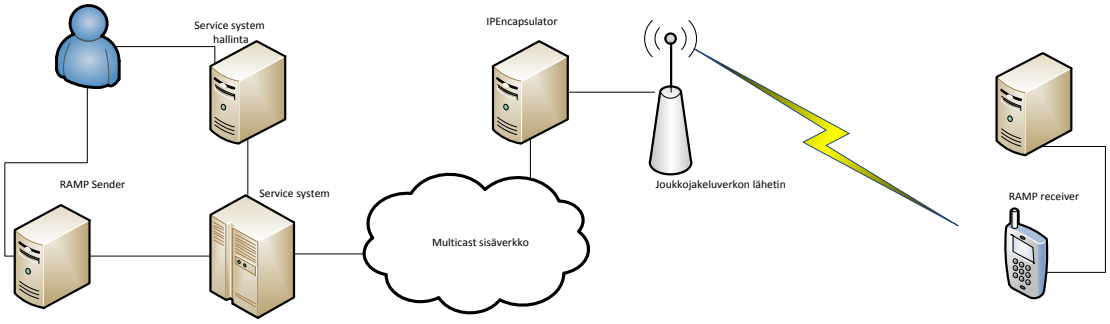
6.3. RampTester

Toteutettavien välikerrosohjelmistokomponenttien yläpuolelle on tarkoitus toteututtaa välikerrosohjelmistokomponenttien julkista rajapintaa käyttävä MobiiliTV-käyttöliittymä, joka muun muassa käyttää välikerrosohjelmistokomponenttien tarjoamia

tiedonsiirtopalveluita. Kuitenkin ennen käyttöliittymäohjelmiston valmistumista oli tarve saada validoitua välikerrosohjelmistojen toiminta oikeaa verkkototeutusta vastaan. Tähän ei kuitenkaan ollut saatavilla valmista työkalua vaan sellaisen toteuttamiselle tuli tarvetta.

Näin ollen päätettiin toteuttaa yksinkertainen testipakettien lähettäjä ja vastaanotto-ohjelmisto. Palvelinpäähän toteutettiin erittäin yksinkertainen haluttuihin multicast-osoitteisiin kasvavalla indeksillä paketteja lähettävä ohjelma ja päätelaitepäähän vastaavaa liikennevirtaa vastaanottava asiakasohjelma.

Kuva 6.4 on kuvattu RampTesterin sijoittuminen järjestelmätason päästä-päähän-arkkitehtuuriin.



Kuva 6.4: RampTesterin järjestelmätason arkkitehtuuri

6.3.1. RampTesterin käyttöönotto

Ramptester mahdollisti välikerrosohjelmistokomponenttien IP-datasiirron validoinnin hyvin aikaisessa vaiheessa implementaatiota. Ramptester toimi myös erinomaisena debug- ja tehokkuusmittaustyökaluna testattaessa allaolevia verkko-ominaisuuksia.

Ramptester on toteutettu käyttämään suoraan välikerrosohjelmistokomponenttien funktioita, eikä se siis siten käytä julkisen rajapinnan tarjoamia abstrahoituja palveluita. Tämä on oleellista siinä mielessä, että RampTesteriä voidaan paremmin siis pitää testiajurina kuin testausohjelmistona. RampTestin ongelmana onkin, että sen suoraviivainen toteutus rajaa tietyn toiminnallisuuden pois. Esimerkiksi RampTester ei itse osaa hakea verkkoa, asettaa kanavaa tai hakea ESG-dataa. Järjestelmässä on ensin manuaalisesti kutsuttava edellämainitut ominaisuudet toteuttavia välikerrosohjemistokomponentteja.

6.3.2. RampTesterin eri toimintamoodit

Ramptesterin asiakasohjelmaan toteutettiin neljä eri moodia. Jatkuvassa moodissa testeri vastaanottaa sovitulla vauhdilla paketteja ja raportoi sekvenssistä puuttuvat paketit. Tässä moodissa pystytään parhaiten testaamaan pakettien läpimenoa. Epäjatkuvassa moodissa Ramptester asiakasohjelma avaa ja sulkee IP-filttereitä

satunnaisessa järjestyksessä. Tämän testimoodin tarkoituksena on testata filtterien avaus- ja sulkulogiikka ja etsiä mahdollisia muistivuotoja filtterien käsittelyn priorisaatiologiikassa. Kaatumis-moodissa verkkorajapinnan liitäntöjä suljetaan väkisin välikerrosohjelmistokomponenttien alapuolisessa käyttöjärjestelmärajapinnassa. Tällöin pystytään testaamaan välikerrosohjelmistokomponenttien virheensietokykyä tilanteessa, jossa filtterien luonti tai ylhäälläpito epäonnistuu. Manuaalisessa moodissa pystytään asiakasohjelman käyttöliittymässä valitsemaan avattavat ja suljettavat verkkoliitännät. Tämä moodi mahdollistaa välikerrosohjelmistokomponenttien priorisointilogiikan testauksen.

6.3.3. RampTester esimerkkitestitapaus

RampTester konfiguroidaan luomalla lähetyspäähän halutun nopeuksinen filecast-sessio. Tässä esimerkissä käytetään aikaisemmin luotua 800 kbps nopeuksista filecast-sessiota. Lähetyspäässä RampTester-palvelimelle luodaan testikonfiguraatio vastaamaan luotua filecast-sessiota.

```
# dst6 port from to step packets/size byterate dst4
FF15:0000:0000:0000:0000:0000:0001:0A04 10006 1000 1350 1 10
12000 232.1.10.4
FF15:0000:0000:0000:0000:0000:0001:0A05 10007 1000 1350 1 10
12000 232.1.10.5
FF15:0000:0000:0000:0000:0000:0001:0A06 10008 1000 1350 1 10
12000 232.1.10.6
FF15:0000:0000:0000:0000:0000:0001:0A07 10009 1000 1350 1 10
12000 232.1.10.7
```

Asiakaspäässä RampTesterille luodaan lähetyspään kanssa vastaava testikonfiguraatio, paitsi että IPv4 osoite on korvattu prioriteetillä, jota käytetään mikäli halutaan testata priorisaatiologiikkaa.

```
# dst6 port from to step packets/size byterate prio
FF15:0000:0000:0000:0000:0000:0001:0A04 10006 1000 1350 1 10
12000 0
FF15:0000:0000:0000:0000:0000:0001:0A05 10007 1000 1350 1 10
12000 1
FF15:0000:0000:0000:0000:0000:0001:0A06 10008 1000 1350 1 10
12000 1
FF15:0000:0000:0000:0000:0000:0001:0A07 10009 1000 1350 1 10
12000 0
```

6.3.4. RampTesterin testiraportit

Ramptester-ajoista kerätään ajonaikaista logia päätelaitteen normaalista debug-tulostevirrasta. Koska logia saattaa kertyä useita kymmeniä megatavuja, ei niitä

ole mielekästä analysoida käsin. Logien analysointiin kirjotettiin tämän takia Perl-ohjelma, joka käy login läpi ja piirtää tiedoista Excel-graafeja.

RampTesterin testilogin kiinnostavat rivit tilastotiedon keräämisen kannalta ovat seuraavanlaiset viisi tulostetta:

[16:44:46.796] xti1:MCU_ASCII_PRINTF; channel:0xE0; msg:CPacketQueue::RunL: 7824 bytes forwarded to the stack

IP-pinolle DVB-H ajurilta siirrettyjen tavujen ja pakettien määrä.

[16:44:46.796] xti1:MCU_ASCII_PRINTF; channel:0xE0; msg:CPacketQueue::RunL: 0 bytes and 0 packets remain in Nif's main buffer

IP-pinon puskuriin siirron jälkeen jääneiden tavujen ja pakettien määrä.

[17:31:29.671] xti1:MCU_ASCII_PRINTF; channel:0xE0; msg:CPacketQueue::AddPacket: 0 bytes and 0 packets in Nif's main buffer

IP-pinon puskuriin lisättyjen tavujen ja pakettien määrä.

[19:18:57.094] xti1:MCU_ASCII_PRINTF; channel:0xE0; msg:RampTest: Bitrate: 294661

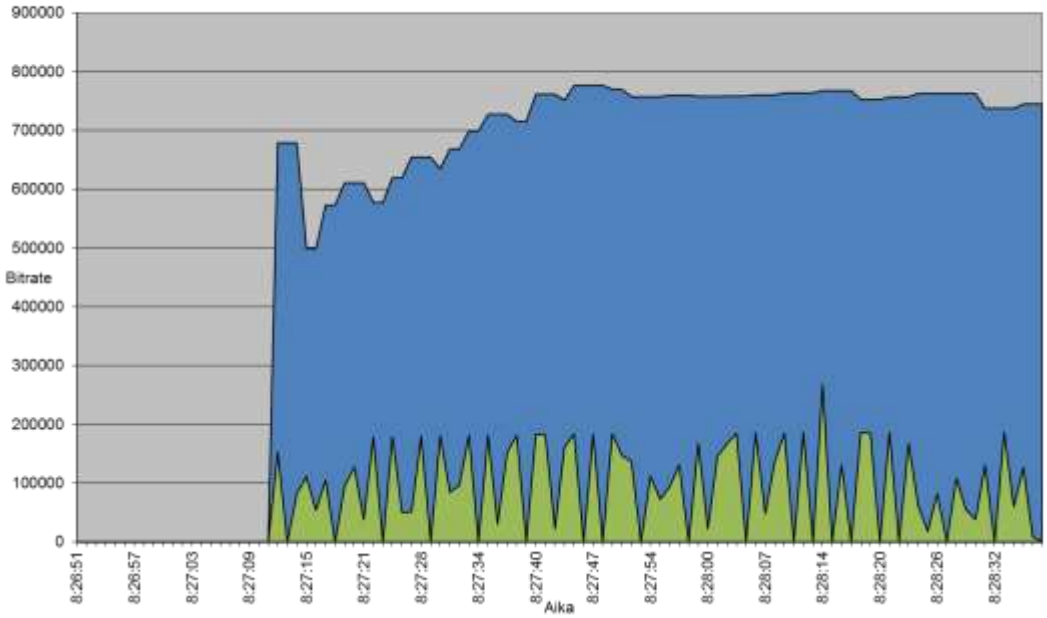
DVB-H ajurilta saatu Bitrate-nopeus.

[19:18:57.094] xti1:MCU_ASCII_PRINTF; channel:0xE0; msg:RampTest Exp: 479400, Got: 479414

RampTester-ohjelman havaitsema pakettien häviäminen siirrossa, Exp=odotettu indeksi, Got=saatu indeksi.

Edellämainittujen testitulosteiden arvoista analyzeramp.pl komentosarja parsii tulokset ja piirtää excel-graafeja.

Kuvassa 6.5 on analysoitu 800 kilotavun RampTester-ajo, jossa sovelluspuolen kuormituksen takia ei pystytä kaikkia paketteja käsittelemään, vaan ne joudutaan pudottamaan. Kuvassa sininen alue on DVB-H-ajurilta saatu Bitrate-nopeus ja vihreä kuormituksen takia pudonneiden pakettien määrä.



Kuva 6.5: Esimerkki analysoidusta testiajosta

6.4. PumppuTester

PumppuTester on projektin ulkopuolelta hankittu IP-liikenteen testaukseen tarkoitettu työkalu. Erona RampTesteriin PumppuTester toimii STIF-testikehyksen mukaisena testimodulina, joten PumppuTesterin avulla pystytään toteuttamaan erilaisia dynaamisempia testitapauksia jollaisiin RampTester ei sellaisenaan kelpaa. PumppuTesterin testitapauksista pystytään helposti keräämään testikokoelma eli testsuite, jolla voidaan kattavasti testata erilaisia testiajoja. Myös testauksen automatisointi on helpompaa STIF-testikehyksen ansiosta.

6.4.1. PumppuTesterin käyttöönotto

Kuten aikaisemmin todettiin RampTester oli testauksen kontekstissa testiajuri, koska se kutsui suoraan testattavien komponenttien funktioita. PumppuTester eroaa RampTesteristä siten, että testitapaus kirjoitetaan käyttäen järjestelmän julkisen rajapinnan, STIF-testien sekä PumppuTesterin tarjoamia funktioita. Julkisen rajapinnan testitapausten kautta pystytään tekemään järjestelmäkutsuja välikerrosohjelmistoille ja esimerkiksi hakea verkko, asettaa kanava tai hakea ESG-data. PumppuTester tarjoaa järjestelmäkutsut IP-yhteyden avaamiseen ja IP-pakettien vastaanottamiseen

PumppuTesterin etu RampTesteriin verrattuna on, että se ei tarvitse erillistä palvelinpuolen testausohjelmaa. PumppuTesterillä voidaan haluttaessa ottaa normaalia DVB-H-verkon RTP-liikennettä vastaan. Tällöin myöskään erillisiä sessioita ei tarvitse konfiguroida DVB-H -Service Systeemiin. Tämä tarjoaa mahdollisuuden testata myös

tuotantoverkkossa tapahtuvaa liikennöintiä. PumppuTesterille on myös olemassa oma lähetyspään palvelin PumppuSender, mutta sitä ei tämän projektin puitteissa käytetty.

Vaikka pumpputester osaa vastaanotettavan RTP-liikenteen sekvenssinumeroista päätellä pakettien hukkumisen, ei normaalin videostreamin käyttäminen sen bittivirran vaihtelevuuden takia aina ole suositeltavaa.

6.4.2. PumppuTester esimerkkitestitapaus

PumppuTesterin esimerkkitestitapaus koostuu järjestelmän alustamisesta, datan vastaanottamisesta ja järjestelmän sulkemisesta.

```
[Test]
title PumppuTest 5000/200 RTP

create IpdcApiTest tstApi
tstApi Constructor
pause 3000

tstApi RegisterReceiverStateObserver

tstApi GetReceiverState 1 // Inactive

tstApi Activate
print activate
waittestclass tstApi // activation started
waittestclass tstApi // activation completed

print PumppuTester -Activation done
tstApi GetReceiverState 3 // NoPlatform

print SCAN STARTS
tstApi SignalScan
waittestclass tstApi
waittestclass tstApi
print PumppuTester -Scan done
tstApi GetReceiverState 3 // NoPlatform

print PumppuTester -SetPlatform
tstApi SetPlatform 1234 // Test platform
waittestclass tstApi
waittestclass tstApi // platform set ready

print PumppuTester -Start receiving

create PumppuTestModule pumppuTester

pumppuTester ReceiveIPData 2 5000 200 0 1 ff15::0050 10300
```

```
waittestclass pumppuTester // Wait data
pumppuTester StopReceivingIPData
delete pumppuTester

tstApi Deactivate

pause 3000

tstApi Destructor
pause 5000
delete tstApi
[Endtest]
```

Edellämainitussa testitapauksessa järjestelmä hakee kanavat ja asettaa vastaanottimen kanavalle 1234. Tämän jälkeen PumppuTester avaa yhteyden IP-osoitteeseen FF15::0050 ja porttiin 10300, josta se ottaa vastaan 5000 pakettia RTP-liikennettä. PumppuTester tulostaa 200 vastaanotetun paketin väliajoin tilastotietoa vastaanotosta. Testitapauksen lopuksi avatut resurssit suljetaan.

6.4.3. PumppuTesterin testiraportit

PumppuTester tuottaa testiajoistaan STIF-testimoduleille tyypillisesti normaalin testiraportin. Käytännössä tulosten analysointi tapahtuu käymällä läpi debug-tuloste päätelaitteesta testiajon ajalta ja tutkimalla testitapauksen tulostamat tilastotiedot.

PumppuTester tulostaa vastaanotettujen ja pudotettujen pakettien määrän sekä keskimääräisen siirtonopeuden.

6.4.4. Testien toistettavuus

Jotta PumppuTesterillä ja samalla myös RampTesterillä pystyttiin toistamaan testitilanteita useampaan otteeseen, tarvitsi järjestelmästä pystyä ottamaan lähetetystä tietovirrasta otoksia ja toistamaan ne tiettynä ajanhetkenä.

Testien toistettavuutta varten järjestelmään pystytettiin toinen instanssi, jonka tarkoituksena oli toistaa aikaisemmin suorasta lähtetysvirrasta tallennetut Transport Stream-tallenteet.

Näiden tallenteiden avulla pystytiin toistamaan myös muilla lähetyspaikoilla tallennettuja otoksia ja näin testaamaan eri bugeja, joita niiden toistossa ilmeni.

6.5. TestResponder

Projektin myöhäisemmässä vaiheessa tuli tarvetta integraatiotestata komponentteja, jotka suorittavat asiakas-palvelin–mallin mukaisia tietoliikenneyhteyksiä. Integraatiotestausvaiheessa ei ollut kuitenkaan mahdollista, eikä myöskään toistettavuuden kannalta sopivaa, käyttää todellista palvelinyhteyttä, joten tuli tarve implementoida testipalvelin.

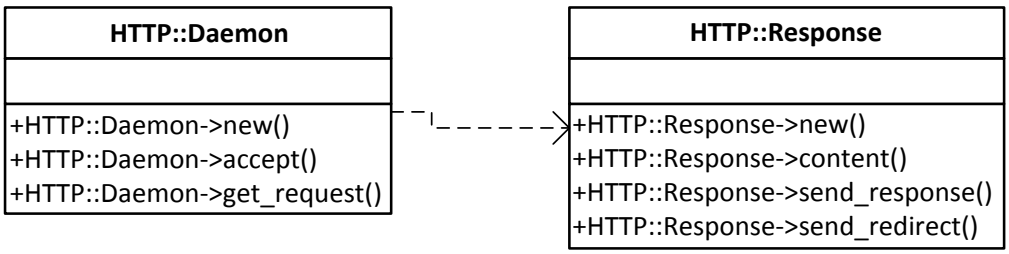
TestResponder-testipalvelin määriteltiin toimimaan siten, että siihen voidaan edeltäkäsin määritellä testitapauksen mukaisia vastaussekvenssejä joissa serveri vastaa sekvenssin mukaisesti asiakasohjelman lähettämiin viesteihin standardin mukaisilla viesteillä.

Hyvä esimerkki tilanteessa, jossa TestResponder-palvelin tuli tarpeeseen oli Ecommerce-välikerrosohjelmistokomponentin testaus. Kuten luvussa 2 kuvattiin on Ecommerce-palvelin ulkoisen operaattorin hallinnassa, joten ellei erillistä testi-instanssia konfiguroida ei kyseisiä palveluita pystytä luotettavasti testaamaan. Testitapausten toistettavuus ei ollut myöskään mahdollista tuotantokäytössä olevia palvelimia vastaan.

Koska asiakasohjelman ja palvelimen välinen viestintä on XML-koodattuja viestejä HTTP/HTTPS protokollan päällä kuljetettuna, on tämän protokollan toteuttaminen hyvin yksinkertaista TestResponder-palvelimella.

6.5.1. Testipalvelimen toteutus

TestResponder-testipalvelin toteutettiin Perl-kielellä käyttäen HTTP::Daemon- ja HTTP::Response-kirjastojen palveluita, joiden rajapinnat on kuvattu kuvassa 6.6. Testipalvelin on siis hyvin yksinkertainen HTTP-palvelin, joka vastaa asiakaspalvelimen HTTP/POST-viesteihin ennaltamääritellyin HTTP/POST-viestisekvenssein.



Kuva 6.6: TestResponderin luokkakaavio. [HTTP::Daemon, 2003]

Testipalvelimeen haluttiin lisäksi toteuttaa konfiguraatiorajapinta. Tämä päätettiin toteuttaa HTTP/GET-viestein, näin palvelinta pystytään konfiguroimaan normaalilla

nettiselaimella antamalla osoitekenttään serverin konfigurointisivun osoite ja parametrina sekvenssi.

6.5.2. Konfiguraatiorajapinnan protokolla

Testiserverin konfiguraatiorajapinta on toteutettu HTTP/GET viestein. Palvelimen konfiguraatiosivulle annetaan parametrina haluttu sekvenssi, esimerkiksi ”http://server-ip/config.html?seq=1,2,3,2,3”. Esimerkissä määritellään viiden viestin sekvenssi, jossa palvelin vastaa asiakkaalta tulleisiin viesteihin esimääritellyin viestein 1, 2, 3, 2 ja 3.

Palvelimelle voi asettaa vain yhden sekvenssin kerrallaan, jonka jälkeen palvelin palaa vastaamaan kaikkiin viesteihin ”OK”-vastauksen. Sekvenssi voidaan nollata kesken asettamalla sekvenssiksi 0 viestillä ”http://server-ip/config.html?seq=0”.

6.5.3. Asiakasrajapinnan protokolla

Asiakasohjelmiston kanssa keskusteltaessa serveri toteuttaa standardin mukaisen protokollan vastaten halutun sekvenssin mukaisesti asiakkaan viesteihin. Palvelin vastaa HTTP/POST-viestinä protokollan mukaisen XML-viestin asiakkaalle ja jää odottamaan seuraavaa viestiä saapuvaksi.

Esimerkiksi palvelin vastaa onnistuneessa tapauksessa asiakkaalle seuraavalla viestillä.

```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<Response xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xsi:noNamespaceSchemaLocation=\"Response.xsd\">
<Response_Code>SUCCESS</Response_Code>
<Value>2388</Value>
</Response>
```

Testipalvelimen mahdolliset vastaukset eräässä testikonfiguraatiossa asiakkaan viesteihin ovat kuvan 6.7 taulukon mukaiset. Sekvenssit 1-3 ovat asiakkaan normaalin protokollan mukaisien tapausten testaukseen. Loput sekvenssinumerot on varattu virhetilanteiden testausta varten. Sekvenssi 4 on normaali ”SUCCESS”-viesti jonka XML-osa on virheellinen. Tällä viestillä voidaan testata asiakasohjelman XML-parserin virhesietokykyä. 301-307 sekvenssit ovat HTTP-protokollan uudelleenohjausviestien testaukseen ja 500 HTTP-protokollan systeemivirheen testaukseen.

Konfiguraatio sekvenssi	Response_code	Käyttötapaus	HTTP-paluukoodi

1	SUCCESS	Validointi onnistui	200
2	INVALID	Validointi epäonnistui	200
3	SYSTEM ERROR	Validoitaessa tapahtui virhe	200
4	SUCCESS	Virheellinen XML-viesti	200
301	-	HTTP-Uudelleenohjauksen testaus	301
302	-	HTTP-Uudelleenohjauksen testaus	302
307	-	HTTP-Uudelleenohjauksen testaus	307
500	-	HTTP-Järjestelmävirheen testaus	500

Kuva 6.7: Testiserverin konfiguraatioesimerkki

Testipalvelin voidaan tästä esimerkistä poiketen tarpeen mukaan konfiguroida vastaamaan hyvinkin suureen määrään erilaisia viestejä.

6.5.4. Testiserverin yhdistäminen integraatiotesteihin

Komponentin integraatiotestit on toteutettu kuten yksikkötestitkin STIF-testikehyksen avulla. Testiserveri otettiin osaksi komponentin automatisoituja integraatiotestejä. Testikehykseen toteutettiin konfiguraatiorajapinnan protokollan toteutus, jonka avulla integraatiotestitapaus asettaa palvelimelle halutun sekvenssin ennen testausta ja suorittaa sen jälkeen testattavan protokollan palvelimen kanssa. Näin pystyttiin helposti automatisoimaan halutujen käyttötapausten testit.

Yhdistämällä TestResponder-instanssi intergraatiotesteihin pystytään myös nopeasti validoimaan toteutetut uudet ominaisuudet lisäämällä TestResponder-palvelimelle määrittelyn mukaiset viestit ja toteuttamalla haluttu sekvenssi testitapaukseen.

7. YHTEENVETO

Tässä diplomityössä kehitettiin Sasken Finland Oy:n asiakasprojektin testausta. Projektissa suoritettiin siirtyminen ketteriä menetelmiä käyttävään ohjelmistotuotantoprosessiin, mikä aiheutti tarpeen projektin testauksen automatisointiin.

7.1. Testauksen kehittämisen arviointi

Aikaisemmassa prosessimallissa testausta suoritettiin harvoin ja siten testattavana oli suuri otos muutoksia. Tämä aiheutti, että mahdollisen regression ilmetessä oli hyvin vaikeaa osoittaa tiettyä yksittäistä muutosta ohjelmakoodissa virheen syyksi.

Testauksen automatisoinnilla pystyttiin testaussykli lyhentämään yhteen vuorokauteen, joten regressiotapauksissa oli hyvin pieni määrä muutoksia läpikäytävänä. Myös testaukseen kuluva työmäärä pystyttiin tiputtamaan minimiin automaation myötä.

Testauksen kehittämisen avulla saatiin kerättyä myös kattavuustietoa ja evaluoitua yrityksen käyttöön koodikattavuutta mittaava ohjelmisto. Kattavuustiedon mittauksen avulla projektissa pystyttiin helposti identifioimaan kriittisimmät osat, joiden testausta parannettiin. Koodikattavuutta mittaava ohjelmisto on otettu myös muissa projekteissa käyttöön tämän projektin hyvien kokemusten perusteella.

Integraatio- ja järjestelmätestauksen kehittämisellä pystyttiin aikaistamaan useiden komponenttien testausta. Esimerkiksi RampTesterin avulla päästä-päähän-tietoliikenneyhteyksien testaus aikaistui huomattavasti, koska puuttuva käyttöliittymä voitiin korvata testerillä. RampTesterin avulla pystyttiin lisäksi löytämään useita virheitä toteutuksessa, joita todellista verkkoliikennettä vastaan liikennöitäessä tuskin olisi helposti pystytty todentamaan. Esimerkkinä hitailla tietovirroilla tapahtunut puskurien köyhtyminen sekä erilaiset priorisaatiologiikan virheet.

TestResponder mahdollisti projektissa aivan uudenlaisen integraatio- ja järjestelmätestauksen tason, kun aikaisemmin mahdottomia testisekvenssejä pystyttiin ajamaan kerta toisensa jälkeen testipalvelinta vasten.

Kokonaisuudessaan testauksen kehittäminen onnistui oikein hyvin. Testiautomaation luomiseen käytetty aika pystyttiin saamaan moninkertaisesti takaisin nopeutuneen kehitystyön ansiosta.

7.2. Jatkokehitysideoita

Vaikka testausta onnistuttiin kehittämään erittäin hyvin tuloksin, jäi vielä joitakin ideoita, jotka jäivät toteuttamatta.

Moduulitestausta voitaisiin kehittää enemmän jatkuvan integroinnin periaatteiden mukaiseksi. Tämä tarkoittaisi, että aina kun versionhallintaan tehtäisiin muutos, ajettaisiin kyseisen komponentin testeistä ainakin jokin alijoukko.

RampTesterin tai PumppuTesterin testausta voitaisiin automatisoida ottamalla käyttöön Service Systeemin tarjoamat SOAP-rajapinnan palvelut. SOAP-rajapinnan kautta on mahdollista uudelleenkonfiguroida esimerkiksi lähetyspään kaista-allokaatio ja näin mahdollistaa hyvinkin erilaisten testien automaattinen ajaminen. Tosin RampTesteriä käytettäessä myös sitä tarvitsisi laajentaa tukemaan dynaamista konfiguraatiota.

TestResponderin konfiguroitavuutta voitaisiin kehittää lisää. Tällä hetkellä, jos halutaan lisätä uusia XML-viestejä järjestelmään, joudutaan ne lisäämään suoraan palvelimen lähdekoodiin. TestResponderille tarvitsisi siis toteuttaa jonkinlainen yksinkertainen konfiguraatiotiedostojärjestelmä.

LÄHTEET

[Burnstein, 2003]
Ilene Burnstein. 2003. Practical software testing: a process-oriented approach. Springer Professional Computing. 714 s.

[DVB Document A099, 2008]
European Telecommunications Standards Institute. 2008. IP Datacast over DVB-H: Electronic Service Guide (ESG). DVB Document A099 Rev.1 .

[ETSI TS 102 005, 2007]
European Telecommunications Standards Institute. 2007. ETSI TS 102 005 V1.3.1 (2007-07). Digital Video Broadcasting (DVB); Specification for the use of Video and Audio Coding in DVB services delivered directly over IP protocols. ETSI Technical Standard. 49 s.

[ETSI TR 102 469, 2006]
European Telecommunications Standards Institute. 2006. ETSI TR 102 469 V1.1.1. (2006-05). Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Architecture . ETSI Technical Report. 38 s.

[ETSI TS 102 470, 2006]
European Telecommunications Standards Institute. 2006. ETSI TS 102 470 V1.1.1 (2006-04). Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Program Specific Information (PSI)/Service Information (SI). . ETSI Technical Standard. 37 s.

[ETSI TS 102 472, 2006]
European Telecommunications Standards Institute. 2006. ETSI TS 102 472 V1.2.1 (2006-12). Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Content Delivery Protocols . ETSI Technical Standard. 76 s.

[ETSI TS 102 474, 2006]
European Telecommunications Standards Institute. 2006. ETSI TS 102 474 V1.1.1. (2007-11). Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Service Purchase and Protection. ETSI Technical Standard. 272 s.

[Faria et al., 2006]
Gerald Faria, Jukka A. Henriksson, Erik Stare, Pekka Talmola. 2006. DVB-H: Digital Broadcast Services to Handheld Devices. Proceedings of the IEEE. v94 i1. 194-209. 16 s.

[Glenford, 2004]
Glenford J. Myers,Tom Badgett,Todd M. Thomas,Corey Sandler. 2004. The art of software testing., 2nd Edition. John Wiley and Sons Ltd. 256 s.

[Haikala ja Märijärvi, 2006]
Haikala, Ilkka. Märijärvi, Jukka. 2006. Ohjelmistotuotanto. 11. painos. Jyväskylä, Talentum Media Oy. 440 s.

[Harju, 2000]
Jarmo Harju. 2000. 83501 Tietoliikenneprotokollat. Luentomoniste osa 1. syksy 2000. 40 s.

[Harrison, 2003]
Richard Harrison. 2003. Symbian OS C++ for Mobile Phones. John Wiley&Sons, Ltd.. 750 s.

[HTTP::Daemon, 2003]
HTTP:Daemon API. Gisle Aas 2003. [WWW] [Viitattu 23.5.2010.] Saatavissa: http://kobesearch.cpan.org/htdocs/libwww-perl/HTTP/Daemon.html

[HTTP RFC, 1999]
R. Fielding, J. Gettys, et al. Hypertext Transfer Protocol -- HTTP/1.1. RFC 2616. IETF. 1999. 114 s.

[HSPA Whitepaper, 2007]
BASIC CONCEPTS OF HSPA. White Paper. 2007 Ericsson AB. [WWW] [Viitattu 23.5.2010.] Saatavissa: http://whitepaper.techworld.com/mobile-wireless-telecom/5230/basic-concepts-of-hspa

[Kumar, 2007]
Amitabh Kumar. 2007. DVB-H, DMB, 3G Systems and Rich Media Applications. Focal Press. 512 s.

[SDP RFC, 2006]
M. Handley, V. Jacobson, et al. SDP: Session Description Protocol. RFC 4566. IETF. 2006. 50 s.

[Stichbury, 2005]
Jo Stichbury. 2005. Symbian OS Explained. John Wiley & Sons, Ltd. 392 s.

[STIF, 2010]
S60 5.x STIF User's Guide, version 33. 2010. Nokia Corporation. [WWW] [Viitattu 23.5.2010.] Saatavissa: http://developer.symbian.org/wiki/images/0/04/S60_STIF_UI_Users_Guide_D.pdf

[STIF UI, 2009]
S60 5.x STIF UI User's Guide, version 2.0. 2009. Nokia Corporation. [WWW] [Viitattu 23.5.2010.] Saatavissa: http://developer.symbian.org/wiki/images/8/80/S60_STIF_Users_Guide_D.pdf

[Sales, 2005]
Jane Sales. 2005. Symbian OS Internals: Real-time Kernel Programming. John Wiley & Sons, Ltd. 940 s.

[XML RFC, 2008]
Tim Bray, Jean Paoli, et al. Extensible Markup Language (XML) 1.0 (Fifth Edition). 2008. W3C Recommendation 26 November 2008.

LIITTEET

Esimerkkitestiraportti komponentille, jolle on ajettu 5 testiä.

```
HttpCommunicationsTester
Covered functions:          102 / 132
Function coverage:          77%
Covered conditions/decisions:  2808 / 3499
Condition/decision coverage:   80%
Covered decisions:          2321 / 2763
Decision coverage:          84%

-------------------------------------------------

**********************************************************************
Tuesday 31st March 2010
7:59:21 am
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
SUMMARY:
        Passed cases: 5
        Failed cases: 0
        Timeout cases: 0
        Crashed cases: 0
        Total cases: 5

-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENVIRONMENT INFO:
HW Info:
        Manufacturer: 0x4, MachineUid: 0x10005f62, Model: 0x4d24
        HW Rev: 0x1, CPU: 0x2, CPU Speed: 1712 MHz
        Language: 1
SW Info:
        SW Rev: 0x100, SW Build: 0x250
Memory Info:
        RAM: 32 MB, RAM Free: 20 MB

-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
TESTCASE SUMMARY:
[testscripter_HttpCommunicationsTester][c:\testframework\HttpCommunicationsTeste
r.cfg][1] Title:[HTTPCT-001a]
        StartTime: 7:59:21.5531 am, EndTime: 7:59:22.1406 am
        Result: 0 [] ==> PASSED
- - - - - - - - - - -

[testscripter_HttpCommunicationsTester][c:\testframework\HttpCommunicationsTeste
r.cfg][2] Title:[HTTPCT-001b]
        StartTime: 7:59:22.1606 am, EndTime: 7:59:22.3437 am
        Result: 0 [] ==> PASSED
```

```
- - - - - - - - - - -

[testscripter_HttpCommunicationsTester][c:\testframework\HttpCommunicationsTeste
r.cfg][3] Title:[HTTPCT-001c]
        StartTime: 7:59:22.3837 am, EndTime: 7:59:22.5625 am
        Result: 0 [] ==> PASSED
- - - - - - - - - - -

[testscripter_HttpCommunicationsTester][c:\testframework\HttpCommunicationsTeste
r.cfg][4] Title:[HTTPCT-001d]
        StartTime: 7:59:22.5781 am, EndTime: 7:59:22.7343 am
        Result: 0 [] ==> PASSED
- - - - - - - - - - -

[testscripter_HttpCommunicationsTester][c:\testframework\HttpCommunicationsTeste
r.cfg][5] Title:[HTTPCT-001e]
        StartTime: 7:59:22.7543 am, EndTime: 7:59:22.9531 am
        Result: 0 [] ==> PASSED
- - - - - - - - - - -

-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
TESTMODULE SUMMARIES:
Module: [testscripter_HttpCommunicationsTester]
        Passed cases: 5
        Failed cases: 0
        Timeout cases: 0
        Crashed cases: 0
        Total cases: 5
```