LAPPEENRANNAN TEKNILLINEN KORKEAKOULU 5.12.1999
Tietotekniikan osasto
Tietoliikennetekniikan laitos

1715 Tietotekniikan erikoistyöt

Loppuraportti

OVOPS++ -protokollasovelluskehyksen käyttö tietoliikenneohjelmistojen toteutuksessa

Tarkastaja: Prof. Jorma Jormakka
Tekijä: Olli Suihkonen, Tite 4

SISÄLLYSLUETTELO

LYHENTEET

ATM	Asynchronous Transfer Mode
B-ISUP	Broadband ISDN User Part
CC	Call Control
CIC	Circuit Identification Code
CORBA	Common Object Request Broker Architecture
FSM	Finite State Machine
FSR	Frame Synchronized Ring
ISDN	Integrated Services Digital Network
ISUP	ISDN User Part
ITU-T	International Telecommunication Union -Telecommunication
LTKK	Lappeenrannan teknillinen korkeakoulu
NI	Network Interface
ORB	Object Request Broker
OVOPS	Object Virtual Operations System
OVOPS++	Object Virtual Operations System ++
PDU	Protocol Data Unit
PF	Protocol Framework
SAAL	Signalling ATM Adaptation Layer
SCCP	Signalling Connection Control Part
SCOMS	Software Configurable Multidiscipline Switch
SF	Scheduler Framework
SI	Servicer Indicator
SS7	Signalling System #7
TCAP	Transaction Capabilities Application Part
TKK	Teknillinen korkeakoulu
TOVE	Toimialaverkot
VTT	Valtion teknillinen tutkimuskeskus

1. JOHDANTO

Object Virtual Operations System ++ (OVOPS++) on oliosuuntautunut protokollasovelluskehys, joka on syntynyt Teknillisen korkeakoulun (TKK) tietoliikenneohjelmistojen ja multimedian laboratorion Toimialaverkot-tutkimusprojektissa (TOVE). TOVE:ssa syntyneen ohjelmiston kehitys jatkuu edelleen projektissa nimeltä Software Configurable Multidiscipline Switch (SCOMS). Näissä projekteissa OVOPS++:lla on toteutettu Valtion teknillisen tutkimuskeskuksen (VTT) Frame Synchronized Ring –kytkimen (FSR) hajautettu ohjausohjelmisto. Jotta sovelluskehystä voitaisiin käyttää muuhunkin, täytyy sen olla riippumaton alkuperäisestä käyttökohteestaan, sekä hyvin dokumentoitu. OVOPS++ on pidetty erillisissä moduuleissa, joten sen siirtäminen muihin projekteihin ja ympäristöihin on periaatteessa yksinkertaista. Sen sijaan dokumentointi on jäänyt tekemättä, ja on syntynyt tarve kirjoittaa kattava dokumentti joka esittelee OVOPS++:n suomia mahdollisuuksia tietoliikenne-ohjelmistojen toteuttamiseen, tehden sen hyvin yksityiskohtaisella tasolla. Dokumentin täytyy perehdyttää vasta-alkaja OVOPS++:aan niin, että tämä kykenee tuottamaan ohjelmakoodia pelkästään dokumentin ja valmiina olevan koodin perusteella. Projektien dokumentointi on hyvin pitkälti hoidettu opinnäytteiden avulla, joten on luonnollista tehdä tästäkin asiasta sellainen.

Tämä erikoistyö siis kertoo OVOPS++:sta, ja perustuu projektissa tehtyyn käyttöohjeeseen. Koska kyseessä on käyttöohje jossa OVOPS++:n käyttöä on havainnollistettu yksinkertaisten koodiesimerkkien avulla, on tässä erikoistyössä otettu hieman enemmän etäisyyttä käytännön työhön, ja tarkasteltu sovelluskehystä teoreettisemmasta näkökulmasta. OVOPS++:n taustalla olevia suunnitteluperiaatteita ja ideoita on pyritty selvittämään, sillä ne ovat varsin mielenkiintoisia ja hyödyllisiä myös muissa oliosuuntautuneissa ohjelmistoissa.

2. OLIOTEKNOLOGIAT

Oliosuuntautunut (object-oriented) ohjelmointi on yleistynyt 90-luvulla, ja oliomemetelmät on todettu käyttökelpoisiksi ja tehokkaiksi. Niiden avulla voidaan mallintaa entistä paremmin todellisen maailman asioita, ja jakaa ongelma-alue helpommin hallittaviin osiin. Olio-ohjelmointia on alettu soveltamaan kaikilla ohjelmistotekniikan alueilla, myös tietoliikennetekniikassa. Oliosuuntautuneet protokollakehitysympäristöt, kuten OVOPS++, ovat hyvä esimerkki tästä. Ne ovat mahdollistaneet tietoliikennejärjestelmien entistä nopeamman kehityksen, ja tuoneet selkeyttä muuten niin monimutkaiseen ja sekavaan aihepiiriin. Vaikka olio-ohjelmointi sinänsä on mullistanut ohjelmistokehityksen, on sen oppiminen ja oikean käytön hallitseminen vaativaa. Tämän vuoksi onkin kehitetty joukko uusia oliomenetelmiä, jotka auttavat oppimisesssa ja käytännön työssä.

2.1. Sovelluskehykset

OVOPS++ on sovelluskehys (framework), joka on suunniteltu erityisesti protokollien ja tietoliikenneohjelmistojen oliopohjaiseen toteuttamiseen. Se on siis runko ohjelmistoille, joissa useat asiat toistuvat samanlaisina tai lähes samanlaisina sovelluksesta toiseen. Sovelluskehys tarkoittaa olio-ohjelmoinnin yhteydessä kiinteästi toisiinsa liittyvien luokkien kokoelmaa, jota täydentämällä ja kokoamalla saadaan valmis sovellus. Sovelluskehys sisältää tietyntyyppisten ohjelmistojen perusratkaisut, esimerkiksi protokollasovelluskehykset sisältävät protokollille tyypillisen ajanjaon. Toisin kuin luokkakirjastot, sovelluskehykset määräävät toteutettavan sovelluksen arkkitehtuurin ja rakenteen, sekä määrittelevät rajoituksia tai vaatimuksia ohjelmiston toiminnan suhteen. Hyvin suunnitellut oliosuuntautuneet sovelluskehykset käyttävät suunnittelumalleja hyväkseen.

2.2. Suunnittelumallit

Oliopohjaisten ohjelmistojen suunnittelu on vaativa prosessi, ja kehitettyjen ratkaisujen tulisi olla uudelleenkäytettäviä, jotta vältyttäisiin pyörän uudelleen keksimiseltä. Kokeneen suunnittelijan erottaakin aloittelijasta siinä, että hän kykenee käyttämään aikaisemmin hyviksi havaitsemiaan ratkaisuja eikä ratkaise jokaista ongelmaa alusta asti. Suunnittelumalleissa (design patterns) on kyse juuri tästä, ne ratkaisevat jonkin tietyn tyyppisen ongelman niin yleisellä tasolla, että ratkaisua voidaan uudelleenkäyttää tarvittaessa. Mallin toimintaa voidaan kuvata esimerkeillä, mutta sitä ei silti tule käsittää ainoastaan tietyn yksittäisen tilanteen ratkaisuna. Suunnittelumalli luokitellaan sen mukaan, onko kyse luovasta (creational), rakenteellisesta (structural) vai käyttäytymiseen (behavioral) liittyvästä mallista. Kun tiedetään ongelman luonne, voidaan luokittelua käyttää hyväksi etsittäessä sopivaa suunnittelumallia.

Suunnittelumalleihin liittyy neljä elementtiä:

- Mallin nimi, jonka avulla ongelma identifioidaan lyhyesti.
- Ongelmankuvaus, joka kertoo milloin mallian voidaan käyttää.
- Ratkaisunkuvaus määrittelee tarvittavat osat, niiden vastuut ja yhteistyön.
- Seurauskuvaus kertoo odotetun tuloksen, vaikutukset ja mahdollisesti aiheutuvat ongelmat.

Suunnittelumalleja on käytetty useissa laajoissa projekteissa, ja niiden käytöstä on saatu runsaasti kokemusta. Niiden käytön on todettu selkeyttävän ja parantavan ohjelmistosuunnittelua, sillä ne korostavat ohjelmiston oliopohjaisuutta ja uudelleenkäytettävyyttä. Toisaalta suunnittelumallien käyttö vähentävää työhön tarvittavia resursseja, kun suunnittelussa tapahtuvien virheellisten ratkaisujen määrä vähenee. Muita suunnittelumallien tuomia hyötyjä ovat dokumentoinnin ja

koulutuksen helpottuminen, sekä ohjelmistoprojektin sisäisen kommunikaation paraneminen.

Suunnittelumallien tehokkaasti käyttäminen vaatii kuitenkin kokemusta, eikä niitä tulisi käyttää vain mallien itsensä vuoksi. Ne antavat vain ratkaisuja suunnittelussa havaittuihin ongelmiin, eivätkä muuten takaa sovelluksen laatua. Sääntöjen käyttö aikakriittisissä järjestelmissä voi myös aiheuttaa suorituskyvyn heikentymistä, vaikka säännöt sopisivatkin hyvin tiettyjen ongelmien ratkaisuksi.

2.3. Komponentit

Oliopohjainen ohjelmointi on luonteeltaan ohjelmiston pienten osasten analyysiin, suunnitteluun ja toteutukseen kantaaottava käsite. Suurien ohjelmistojen käsitteleminen ja hahmottaminen pelkkien olioiden ja luokkien avulla on kuitenkin hankalaa, sillä ne eivät kuvaa riittävän suuria kokonaisuuksia. Olioiden väliset rajapinnat monimutkaistuvat ja tuloksena on huonosti hallittavaa ja sotkuista koodia. Niinpä onkin syntynyt tarve komponenteille, jotka hahmottavat useampien olioiden tai luokkien vuorovaikutusta. Komponentti on yksilöitävissä oleva ohjelmiston osa, jolla on selkeät rajat. Komponentilla on tietyt ominaisuudet, ja se on suunniteltu uudelleenkäytettäväksi. Käyttäjä tuntee komponentin tarkoituksen sekä sen tarjoaman rajapinnan, jonka avulla se voidaan liittää muuhun ohjelmistoon. Sen sijaan komponentin sisäistä toimintaa tai toteutusyksityiskohtia ei tarvitse tietää voidakseen käyttää komponenttia. Komponentteja voidaan koota kokonaisuuksiksi, jotka puolestaan ovat suurempien ohjelmistojen komponentteja. Komponentit sopivat sovelluskehyksiin liitettäviksi, tai toisaalta sovelluskehykset voivat perustua komponenttipohjaiseen ajatteluun, kuten Conduits+ -malli johon myös OVOPS++ perustuu. [Toi97], [Num99]

3. CONDUITS+

Conduits+ on malli tietoliikenneprotokollien kehitysympäristöksi, joka hyödyntää tehokkaasti suunnittelumalleja, ja esitelty paperissa [Hün95]. Suunnittelumallien avulla on haluttu parantaa uudelleenkäytettävyyttä, selkeyttää suunnittelu- ja toteutusvaihetta sekä tuda yhtenäisten termien käyttö kommunikointiin ja dokumentointiin. Conduits+ on niinsanottu mustalaatikko-kehitysympäristö (black-box framework), missä sisäinen toiminnallisuus ei näy ulospäin ja sovellukset rakennetaan valmiista komponenteista. Komponenttien yhteensovittaminen on helppoa, ja koko sovelluksen rakenne yksinkertaistuu, kun protokollaspesifiset ratkaisut voidaan toteuttaa valmiiksi määrättyihin komponentteihin. Ympäristössä on kaksi pääkäsitettä: tietoa käsittelevät osat eli kanavat (conduits) ja tietoa esittävät osat (information chunks). Kaikki muut käsitteet johdetaan näistä pääkäsitteistä.

3.1. Protokolla

Protokolla Conduits+:ssa käsittelee tietopalasia, ja vastaa tietoliikenneprotokollan toiminnallisuudesta. Toiminnallisuuden käsittelystä vastaa äärellinen tilakone (FSM, Finite State Machine), joka toteutetaan State-suunnittelumallin avulla. Protokollaolio on yhteyskohtainen, ja tietopalaset reititetään tarvittaessa protokollainstansseille multiplekserin avulla. Protokollalla voi olla yksi kanava kummallakin sivullaan.

3.2. Multiplekseri

Mux eli multiplekseri yhdistää yhdistää yhden kanavan moneen, eli kyseessä on 1:N multipleksointi. Multiplekserin pääasiallinen tehtävä on ohjata saapuvat viestit oikealle protokollalle jonkin tunnisteen mukaan. A-puoli on yhdistetty yhteen kanavaan, ja B-puolella on mielivaltainen määrä kanavia joita lisätään ja poistetaan

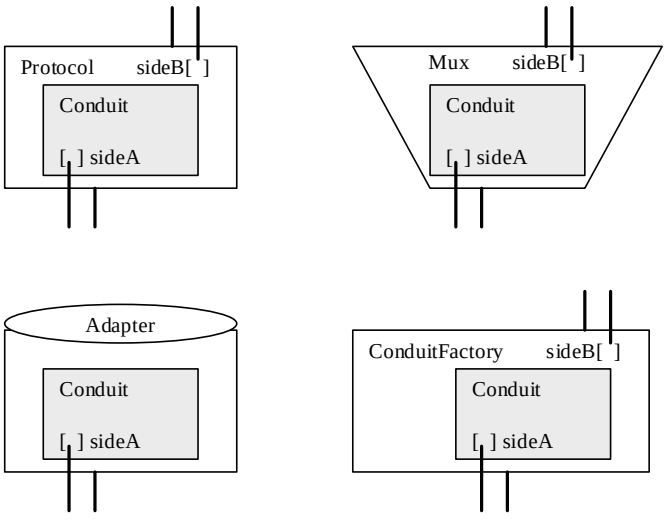
dynaamisesti. Multiplekserillä on lista yhdistetyistä kanavista, ja yhteydet erottava tekijä on protokollan määrittelemä tunniste.

3.3. Kanavatehdas

Kanavatehtaan tehtävänä on luoda uusia protokollia uusille yhteyksille, mikäli niitä ei ole olemassa. Kanavatehdas myös hoitaa näiden yhteyksien lisäämisen multiplekserille. Kanavatehdas on oletuskanava multiplekserille, eli kun multiplekseri toteaa ettei sillä ole tunnisteen mukaista yhteyttä, viesti ohjataan kanavatehtaalle. Uuden protokollainstanssin luomisen jälkeen yhteyden viestit voidaan ohjata sille multiplekseriltä.

3.4. Adapteri

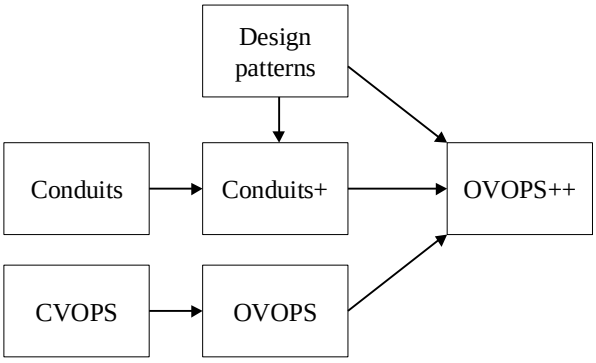
Adapterilla on ainoastaan A-puoli johon voidaan liittää muita kanavia. B-puoli liitetään käyttöjärjestelmän palveluihin, esimerkiksi tiedostoihin tai laitteisiin. Adapteri osaa muuntaa A-puolelta saapuvien viestien rakenteisen tiedon esimerkiksi sokettien vaatimaksi tietovirraksi.



Kuva 1. Conduits+ -mallin pääkomponentit.

4. OVOPS++

OVOPS++ perustuu ideoiltaan Conduits+:aan, ja käytännön toteutukseltaan osittain Lappeerannan teknillisen korkeakoulun (LTKK) OVOPS-projektin ensimmäiseen versioon. OVOPS++:n pääasiallisena tarkoituksena on vähentää protokollakehitysympäristön kompleksisuutta, helpottaa protokollien laajentamista ja uudelleenkäyttöä sekä olla tehokas. Conduits+:n ideat eivät eroa radikaalisti OVOPS:n ideoista. Tärkeimmille käsitteille (multiplekseri, protokolla, kanavatehdas, adapteri) löytyy suora vastine OVOPS:in käsitteistöstä. Tästä huolimatta OVOPS:lla ja OVOPS++:lla ei ole kovin paljoa yhteistä, sillä Conduits+:aa on noudatettu varsin tarkasti, mikä on pakottanut luopumaan OVOPS:n protokollaratkaisuista. OVOPS++ eroaa OVOPS:sta eniten siinä, että sen tarjoaa mustalaatikkokehitysympäristön OVOPS:n valkolaatikkokehitysympäristön sijaan, mikä tarjoaa tiettyjä etuja protokollan kehittäjälle. Tästä kuitenkin saattaa seurata suorituskyvyn heikkeneminen aikakriittisissä järjestelmissä, sillä kyse on C++ -ohjelmointikielen perinnän ja virtuaalifunktioiden käytöstä, sekä suunnittelumalleista joita kehittäessä tehokkuus ei ole ollut pyrkimyksenä. Suunnittelumallien käyttö on tärkeä osa myös Conduit+ -mallia, mutta OVOPS++:ssa niiden käyttöä on pyritty lisäämään entisestään.



Kuva 2. OVOPS++:n kehittyminen.

4.1. Ympäristö

OVOPS++ toimii UNIX-ympäristössä. Se on toteutettu C++ -kielellä, ja vaatii riittävän modernin kääntäjän toimiakseen. Kehitysympäristönä ja suositeltavana ympäristönä on käytetty Linux 2.2 –kerneliin pohjautuvaa käyttöjärjestelmää, ja kääntäjänä on ollut egcs tai g++. Standard Template Library (STL) täytyy olla asennettuna, sillä OVOPS++:ssa käytetään sen määrittelemiä tietorakenteita. Sovelluskehys on myös portattu IRIX-käyttöjärjestelmään menestyksekkäästi, ja periaatteessa se voisi toimia missä tahansa UNIX-järjestelmässä, mikä kuitenkin saattaa vaatia muutoksia koodiin. OVOPS++ osaa myös käsitellä CORBA:n (Common Object Request Broker Architecture) tapahtumajonoa samassa prosessissa protokollien käsittelyn kanssa. CORBA:n käyttäminen vaatii ORBacus version 3.x asentamista.

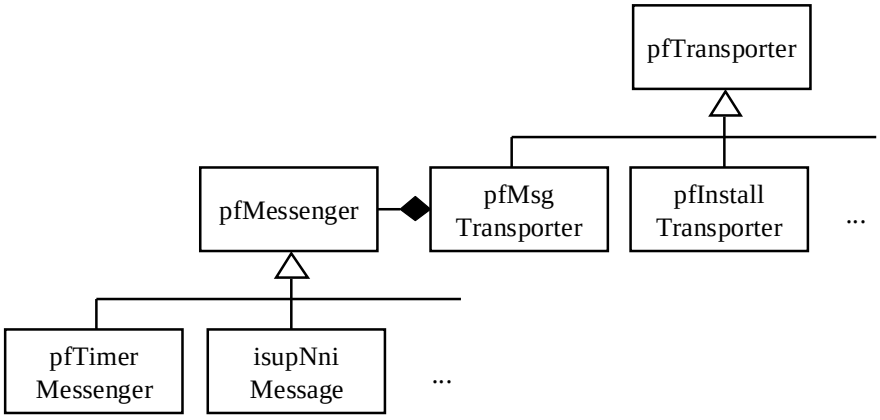
4.2. Perusarkkitehtuuri

OVOPS++ voidaan jakaa karkeasti kahteen osaan, scheduler framework (SF) ja protocol framework (PF), jotka toimivat keskenään. SF on vastuussa ajanjaosta, ja sen toiminta on nähtävissä ainoastaan PF:n kantaluokissa, missä PF-komponentit pyytävät skedulerilta suoritusaikaa. PF puolestaan jakautuu Conduit+:n mukaisesti kanaviin ja tietoa esittäviin osiin. Kanavien rinnalle on tuotu idea proxyistä, jotka kätkevät kanavat sisäänsä. Kanavia käytetään proxyjen kautta, ja ainoastaan proxyjä pystyy kytkemään toisiinsa. Näin on mahdollistettu automaattinen muistinhallinta, jossa pidetään kirjaa kanavaa kohti luotujen proxyjen määrästä. Kun proxyjä ei ole jäljellä, tuhotaan kanavaoliokin.

Tietoa esittävät osat:

- Messenger, jota käytetään tiedon, kuten PDU:n (Protocol Data Unit), primitiivin tai protokollan sisäisen signaalin välittämiseen.

- Transporter, joka kuljettaa Messengerin tai muuta informaatiota kanavien välillä.
- Timeout, viesti joka otetaan vastaan protokollan tilakoneessa ajastimen lauettua.



Kuva 3. pfMsgTransporter kuljettaa viestejä.

4.3. Käytetyt suunnittelumallit

OVOPS++:n tärkeimmät ratkaisut ja toiminnot perustuvat suunnittelumalleihin. Käyttäjän tulisi yrittää hahmottaa suunnittelumallien ideoita, jotta vältyttäisiin tilanteelta jossa sovelluskehystä käytetään ilman että ymmärretään sitä. Tämä johtaa helposti siihen, ettei sovelluskehyksen tarjoamia palveluita käytetä, tai ratkaisut vain kopioidaan valmiista koodista ja toivotaan parasta. Tässä on esitelty lyhyesti kunkin suunnittelumallin sijoittuminen ohjelmistoon.

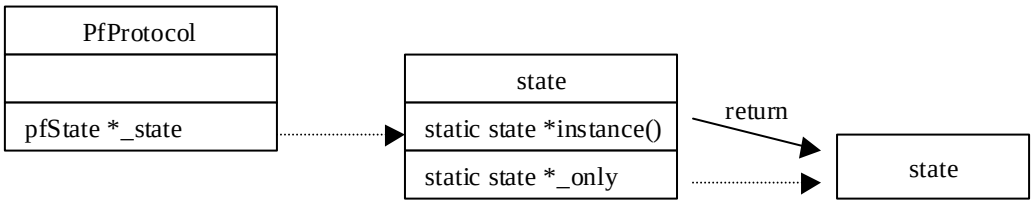
4.3.1. State

OVOPS++:n tilakone on suunniteltu State-suunnittelumallia käyttäen. Siinä jokaista tilakoneen tilaa kohden on oma olio. Protokolla voi vaihtaa omaa tilaansa, jolloin se käytännössä siirtää tilaosoittimensa osoittamaan uuteen tilaan. Protokollan toiminnallisuus on toteutettu tilaobjekteihin, ja protokolla voi

muuttaa käyttäytymistään vaihtamalla tilaa. Itse protokolla sisältää ainoastaan keinot viestien lähettämiseen ja vastaanottamiseen.

4.3.2. Singleton

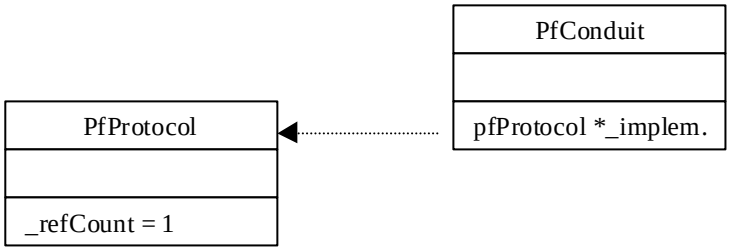
Singleton-malli varmistaa, että luokasta on muodostettu ainoastaan yksi olio, ja siihen on globaali pääsy. OVOPS++:ssa tilaolioissa ei säilytetä mitään protokollakohtaista tietoa, vaan ne sisältävät toiminnallisuuden eri tilanteissa. Näinollen protokollaolioilla voi olla yhteiset tilaoliot, eikä jokaiselle luoda turhaan omia tiloja. Tämä on toteutettu tilakoneissa niin, että tilaluokalla on staattinen instance() –metodi, joka palauttaa osoittimen ainoaan luotuun olioon.



Kuva 4. Singleton-suunnittelumallin käyttö tilakoneessa.

4.3.3. Proxy

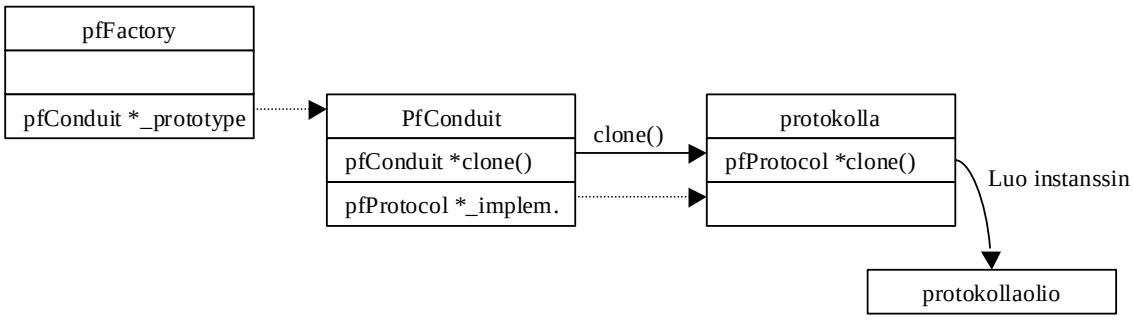
Proxy on korvike tai "kuori" joka peittää olion, ja valvoo pääsyä kyseisen sen metodeihin ja ominaisuuksiin. OVOPS++:ssa kanavaolioita käytetään pfConduit -proxyjen kautta. Kanavia ei voida yhdistää suoraan toisiinsa, vaan niihin osoittavat proxyt yhdistetään. Proxyjä voidaan luoda useita yhtä kanavaoliota kohti, ja niiden määrästä pidetään kirjaa.



Kuva 5. Proxyjen käyttö OVOPS++:ssa.

4.3.4. Prototype

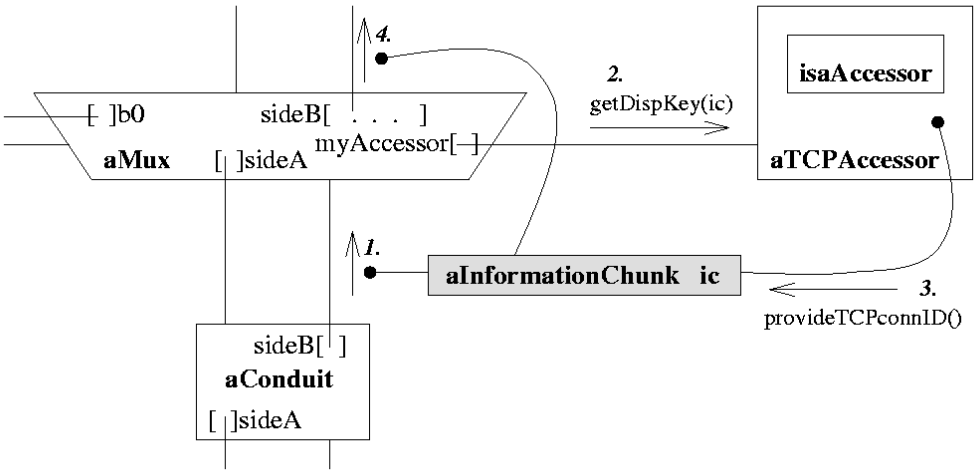
Kanavatehdas käyttää prototyyppiä protokollainstanssien luomiseen multiplekserin päälle. Prototyypillä on clone() –metodi, joka luo oliosta kopion ja palauttaa sen kutsujalle. Kanavatehdas käyttää prototyyppinä olevaa protokollaa pfConduit-proxyn kautta. Operaatiossa luodaan sekä uusi protokolla, että sen peittävä proxy. Tehtaalle palautetaan proxy, joka voidaan sitten yhdistää multiplekseriin.



Kuva 6. Tehdas kloonaa prototyypin.

4.3.5. Strategy

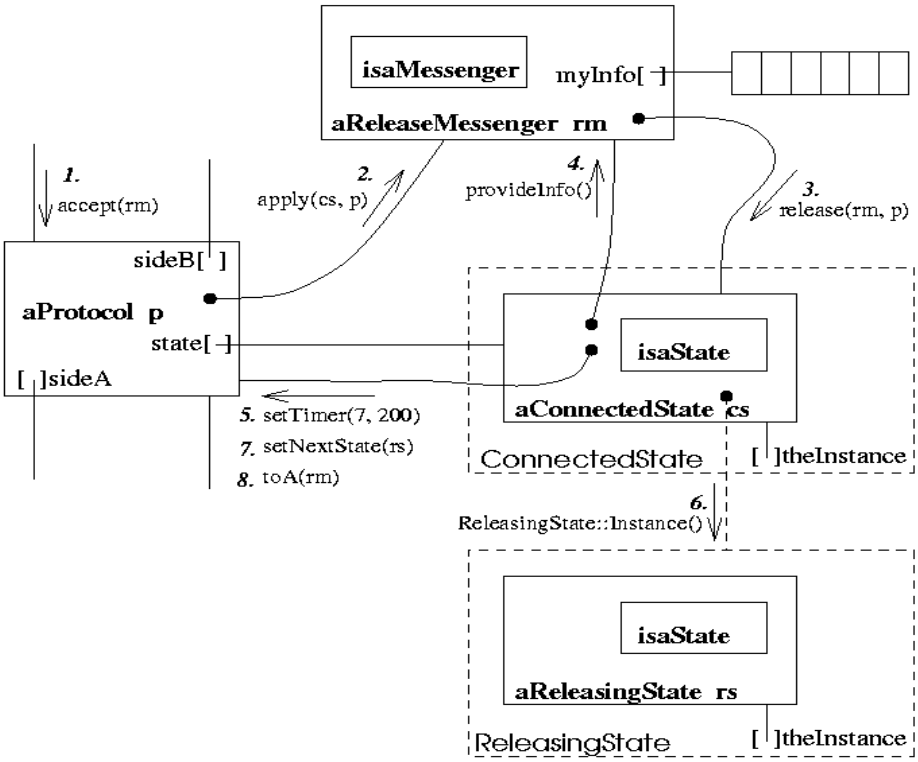
Strategy-suunnittelumalli määrittelee joukon algoritmeja, joista kukin koteloidaan ja tehdään vaihdettavaksi. Näin algoritmia voidaan vaihtaa clientin tietämättä. OVOPS++:ssa halutaan käyttää multiplekseriä vapaasti kanavagraafissa, ilman että siihen tarvitsee lisätä mitään protokollaspesifistä koodia. Strategyä on käytetty multiplekserin yhteydessä niin, että multiplekseri pystyy erottamaan tunnisteen saapuvasta viestistä vakiomenetelmällä. Kullekin protokollalle tehdään spesifinen accessor-luokka, joka sisältää kyseisen protokollan vaatiman toiminnan.



Kuva 7. Accessor irrottaa protokollan yhteystunnisteen.

4.3.6. Command

Command-suunnittelumallin tarkoituksena on koteloida pyyntö olioksi, ja mahdollistaa sen manipulointi jollain tavalla. Jos protokollan viestit käsitettäisiin ainoastaan bittijonoina, ei niille tietenkään voitaisi tehdä funktiokutsuja tai muita oliomaailman toimintoja. OVOPS++:ssa määritellään luokka pfMessenger, joka sisältää informaatiopalasen sekä apply-operaation jota voidaan kutsua sen saapuessa kanavaan. pfMessenger kuvaa tapahtumaa tai toimintoa joka käynnistetään protokollan tilakoneessa, joten se on esimerkki Command-suunnittelumallista. Command mahdollistaa minkä tahansa viestin käsittelyn missä tahansa kanavassa. Viesti voidaan siirtää naapurikanavalle mikäli sille ei aiota tehdä operaatioita, mutta protokollakanavat käsittelevät viestiä apply-operaation avulla, antaen viestille protokollan senhetkisen tilan parametrina. Seurauksena viesti puolestaan herättää oikean toiminnon tilaoliossa.

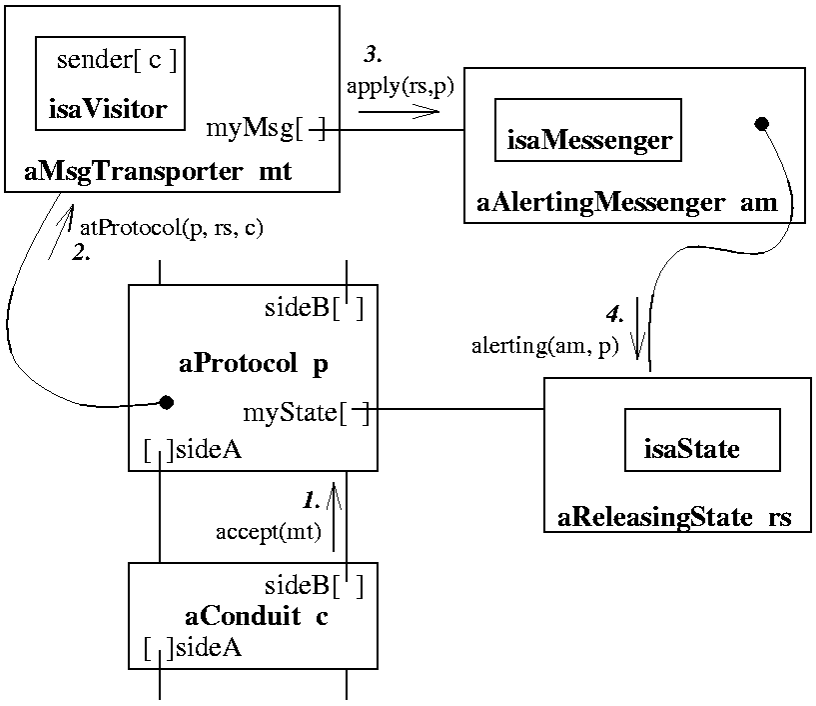


Kuva 8. Command, State ja Singleton toiminnassa

4.3.7. Visitor

Kanavien välillä tulee pystyä kuljettamaan muutakin informaatiota, kuin protokollien omia viestejä tai primitiivejä. Esimerkiksi kanavien asentaminen tai poistaminen suoritetaan viesteillä, joiden ei ole tarkoitus olla tekemisissä protokollien tilakoneiden kanssa. Niinpä on määriteltävä tapa, jolla viestin saapuessa kanavaan päätetään, suoritetaanko sille apply-operaatio vai tehdäänkö jotain muita toimintoja. Visitor-suunnittelumallin tarkoituksena on esittää operaatio, joka suoritetaan oliomallin elementeille. Sen avulla voidaan määritellä uusi toiminto, ilman että vaihdetaan operoitavien elementtien luokkia. OVOPS++:ssa tämä toteutetaan määrittelemällä abstrakti pfTransporter-luokka (visitor), josta sitten peritään tarvittavia aliluokkia kuten pfMessageTransporter.

Kanavat ovat tekemisissä pfTrasporterin kanssa, joten ne eivät riipu pfMessenger-luokkahierarkiasta. Kanavat herättävät toiminnon pfTransporter-oliossa, ja kutsuttavan metodin nimestä pfTransporter saa selville kutsuvan kanavan tyypin. Näin voidaan määritellä eri toiminnot eri kanaville, esimerkiksi multiplekserille saapuva pfMessageTransporter ei kutsu apply-metodia viestistä, koska se on ainoastaan tilakonetta käyttävälle kanavalle tarkoitettu toiminto. [Hün95] [GHJV95]



Kuva 9. Transporter on suunniteltu Visitor-mallin avulla.

5. TOTEUTUKSET

OVOPS++ -käyttöohjeessa on koodiesimerkkien lisäksi esitelty kaksi valmista toteutusta, jotka esittelevät sovelluskehyksen toimintaa. OVOPS++ -testi on ohjelma, joka käyttää PF:n tärkeimpiä komponentteja, ja testaa samalla myös ajanjaon toiminnan. ISUP (ISDN User Part) on puolestaan yhteiskanavamerkinantoverkon (SS7, Signalling System #7) protokolla, jolla on demonstroitu kerrosprotokollan toteutusta. Itse protokolla on tehty SCOMS-ohjelmistoon osana moniprotokollakytkimen ohjausta.

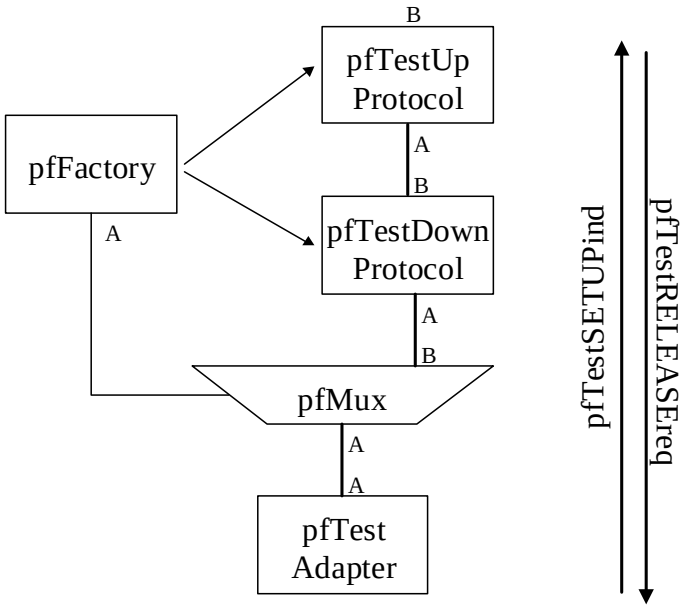
5.1. OVOPS++ -testi

Alunperin uusien käyttäjien perehdyttämiseksi tarkoitettu testi/harjoitusohjelma koostuu kahdesta protokollasta, jotka luodaan tehtaan avulla multiplekserin päälle, sekä alla olevasta adapterista. Testin esimerkkitoteutuksesta käy ilmi OVOPS++:n tärkeimmät toiminnot. Sitä voidaan käyttää OVOPS++:n ja sen ympäristön testaamiseen, esimerkiksi muistivuotojen etsimiseen, erilaisten tilanteiden simulointiin ja PF:n uusien ominaisuuksien tai muutosten testaamiseen. Ohjelma sisältyy OVOPS++ -pakettiin, ja sen tarkoituksena on mahdollistaa asennetun sovelluskehyksen toiminnan nopea testaaminen, ja esitellä sen tärkeimmät toiminnot.

5.1.1. Rakenne ja toiminta

Käytetyt PF:n luokat:

- pfConduit
- pfProtocol
- pfFactory
- pfMux
- pfAccessor
- pfAdapter
- pfMessenger
- pfUnInstallTransporter
- pfTimerMessenger



Kuva 10. OVOPS++ -testiohjelman rakenne.

pfTestAdapter simuloi protokollapinoa linkkitasolla. Adapterille annetaan luotaessa määrä, joka uusia yhteyksiä tulee luoda multiplekserin päälle. Yhteys luodaan lähettämällä pfTestSETUPind multiplekserille, joka puolestaan kutsuu tehdasta ja aikaansaa näin uusien protokollaolioiden luomisen. Kun pfTestUpProtocol saa pfTestSETUPind-viestin, se vastaa pfTestRELEASEreq-viestillä joka aiheuttaa yhteyden purkamisen.

5.1.2. Toteutusyksityiskohtia

Kun tehtävän ohjelmiston rakenne on kanavatasolla selvitetty, voidaan sitä alkaa toteuttamaan. Toteuttamisvaihe OVOPS++:lla kannattaa aloittaa rajapintojen määrittämisellä protokollien välillä. Rajapinnan mahdolliset primitiivit ja viestit määritellään abstraktissa luokassa, joka sitten peritään protokollan tilakoneen luokkiin. Toisin sanoen rajapintaluokassa määritellään ne metodit, joita Command-suunnittelumallin mukaisesti voidaan kutsua eri messenger-olioista. Myös ajastimien timeout-viestien vastaanotto määritellään samalla tavoin. Multiplekseriä ja tehdasta käytetään sellaisenaan ilman periyttämistä, eikä niiden

rajapintoja määritellä koska niillä ei ole tilakonettakaan. Sen sijaan adapterille määritellään tilakone samalla tavalla kuin protokollillekin.

```
// Alemman protokollan tilojen kantaluokka. Input-luokat sisältävät
// eri viestejä vastaavien metodien esittelyt.

class pfTestDownProtocolState : public pfState,
                                public pfTestUpInputs,
                                public pfTestDownInputs,
                                public pfTestTimeoutInputs
{
    public:
        pfTestDownProtocolState(void);
        virtual ~pfTestDownProtocolState(void);

        virtual void pfTestSETUPindAct(
            pfTestSETUPind *primitive_,
            pfProtocol *protocol_);

        virtual void pfTestRELEASEreqAct(
            pfTestRELEASEreq *primitive_,
            pfProtocol *protocol_);

        virtual void pfTestTimeoutAct(
            pfTestTimeout *timeout_,
            pfProtocol *protocol_);
};
```

Tilakoneen toteuttaminen vaatii luonnollisesti sitä, että toteutettavan protokollan standardista on selvitetty tilakoneen toiminta. State-suunnittelumallin mukaisesti jokaista tilaa kohti on oma luokka, mutta jotta rajapintaluokkien idea toimisi paremmin, määritellään protokollan tiloille pääluokka. Tässä pääluokassa toteutetaan mahdollisimman järkevät oletustoiminnot eri viestien vastaanottamiselle. Perityissä luokissa sitten korvataan tarpeen mukaan näitä metodeja, mutta joissain tapauksissa kuten yhteyden purkamista ilmaisevissa viesteissä toiminta saattaa olla sama joka tilassa.

Viestien toteuttaminen liittyy läheisesti tilakoneen toimintaan. Jokaisella viestillä on apply-metodi, jota kutsutaan kun viesti saapuu kanavalle jolla on tilakone. Metodi saa parametrina senhetkisen tilan, josta kutsutaan oikeaa metodia toiminnon herättämiseksi.

```
// Viestin tärkein metodi: apply. Parametrina saatava tilaosoitin
// muutetaan cast-operaatiolla Input-luokan tyyppiseksi, varmistaen
// samalla sen oikea tyyppi. Tämän jälkeen voidaan herättää oikea
// toiminto tilassa.

void pfTestSETUPind :: apply(pfState *state_, pfProtocol *protocol_)
{
    pfTestUpInputs *input = dynamic_cast<pfTestUpInputs*>(state_);
    THROW_IF_DYNAMIC_CAST_FAILED(input);
    input->pfTestSETUPindAct(this, protocol_);
    return;
}
```

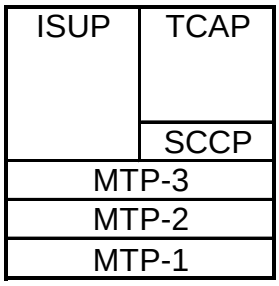
Kanavagraafin mukainen oliorakenne pystytetään pääohjelmassa. Siinä initialisoidaan OVOPS++-ympäristö, luodaan kanavat ja liitetään ne toisiinsa. Ohjelmisto voi luonnollisesti sisältää dynaamisia osia, joissa kanavat luodaan tarpeen mukaan, mutta jonkinlainen pääohjelmassa koottava perusrakenne on aina olemassa. Mikäli kanavaluokat on toteutettu oikein, niiden konstruktorit palauttavat proxyn joka pitää sisällään varsinaista kanavaa. Pääohjelmassa ei yleensä ole tarvetta käyttää suoraan kanavien metodeja, vaan toiminnot herätetään nimenomaan viestejä käyttämällä.

```
// Pääohjelmassa luodaan tarvittavat kanavat, ja liitetään ne yhteen
// ennen skedulerin käynnistämistä. Kanavien create-metodit
// palauttavat suoraan proxy-olion.

   // Luodaan kanavaoliot (proxyt)
   pfConduit adapterProxy = pfTestAdapter :: create(counter,
       timerForAdapter);
   pfConduit downprotocolProxy = pfTestDownProtocol :: create();
   pfConduit upprotocolProxy = pfTestUpProtocol :: create();

   // Luodaan kanavatehdas, joka saa kloonattavan protokollan proxyn
   // prototyypiksi. Molemmat prototyypit annetaan tehtaalle joka luo
   // ja yhdistää ne.
   pfConduit factoryProxy = pfFactory ::
       createFactory(downprotocolProxy, upprotocolProxy);

   // Luodaan multiplekseri ja sen accessor (testAccessor)
   pfTestAccessor *accessor = new pfTestAccessor(maxValue);
   pfConduit muxProxy = pfMux :: createMux(accessor);

   // Yhdistetään kanavat (adapter, mux, factory)
   adapterProxy.connectToA(muxProxy);
   muxProxy.connectToA(adapterProxy);
   muxProxy.connectToB(factoryProxy);
   factoryProxy.connectToA(muxProxy);
```

```
// Käynnistetään skeduleri (sf)
try
{
    pfSystem::instance()->run();
}
```

5.2. ISUP-protokolla

OVOPS++ on parhaimmillaan kerrosprotokollien toteutuksessa, joten käyttöohjeeseen on lisätty esimerkiksi sellaisen toteutus. ISDN User Part on yhteiskanavamerkinantoverkon protokolla, joka tarjoaa signalointitoiminnat ISDN-verkon (Integrated Services Digital Network) palveluille ääni- ja datasiirtoon. ISUP on määritelty ITU-T:n suosituksissa Q.761- Q.764, sekä Q.766.



Kuva 11. SS7 -protokollapino.

5.2.1. Lähtökohta

ISUP käyttää MTP:n (Message Transfer Part) palveluita tiedon kuljettamiseen ISDN-käyttäjien välillä. SCOMS-ohjelmistossa ISUP-protokollan yläpuolinen sovellus on Call Control (CC), joka yhdistää sisääntulevan yhteyden protokollapinon uloslähtevään. Ennen toteutusta tiedossa olivat käytettävät ylä- ja alarajapinnat. Vastaava signalointiprotokolla laajakaistapuolelta, eli Broadband ISDN User Part (B-ISUP) oli jo totetettu, joten selvitettäväksi lähinnä jäi, kävisikö sama rakenne myös kapeakaistapuolelle. B-ISUP eroaa ISUPista erityisesti

viestiensä puolesta, joiden rakenne on täysin erilainen. Viestisekvenssit ja tilakoneet ovat pääosin samoja, tosin ISUP:n standardi ei tilakonetta edes määrittele, joten toteutuksessa se täytyi selvittää varsin sekavan standardin avulla. ISUP-standardi on toteutuksen kannalta ongelmallinen, sillä se käsittelee protokollan toimintaa lähinnä kokonaisen puhelinkeskuksen kannalta. SCOMS-ohjelmistossa toimintoja piti käsitellä enemmänkin tulevan ja lähtevän yhteyden kannalta.

B-ISUPin suunnittelussa oli käyty läpi erilaiset rakennevaihtoehdot. Niistä oli valittu ratkaisu, joka jakaa protokollan rakenteen kahteen isompaan kokonaisuuteen. Niistä alempi on viestien koodauksen ja dekoodauksen hoitava osa, ja ylempi puolestaan protokollan tilakone. Ratkaisu on siitä järkevä, että kaikki alemmalta tasolta tulevat viestit käsitellään ensin yhteisessä osassa, ja jaetaan sen jälkeen yhteyttä kuvaaville protokollainstansseille. [Raa99] Samaa rakennetta päätettiin käyttää myös ISUP-toteutuksessa selkeyden vuoksi.

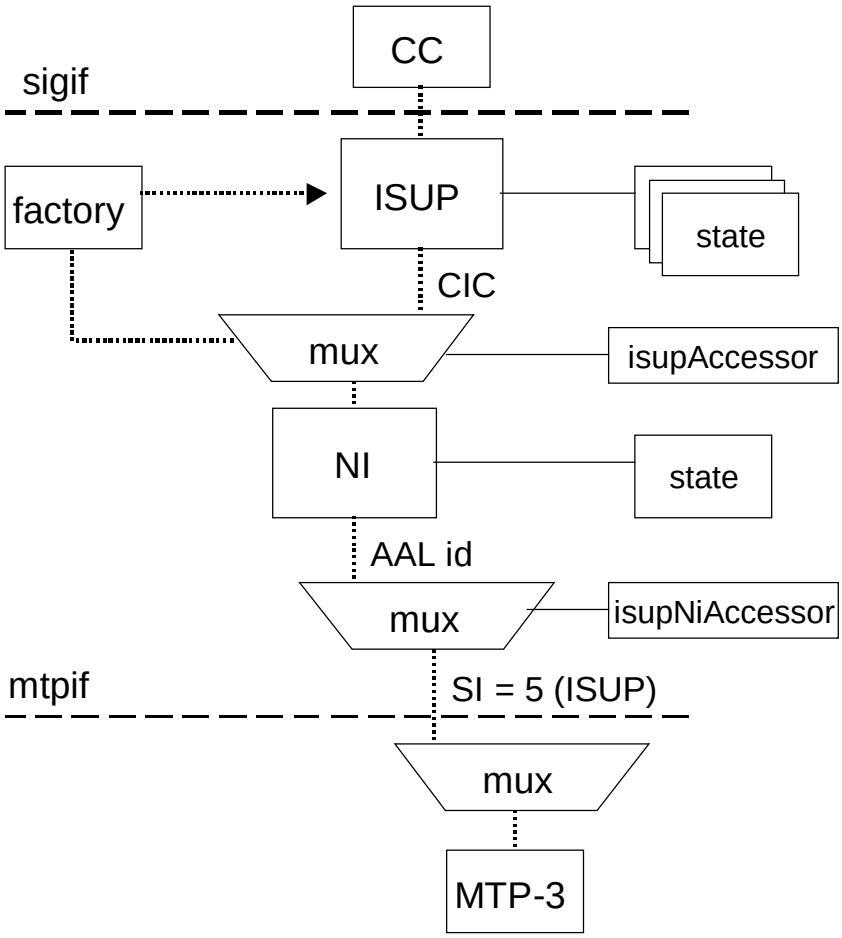
5.2.2. Rakenne

Kuva 12 sisältää tärkeimmät komponentit ISUP-toteutuksessa. Jokaisella MTP-3:n käyttäjällä on oma SI-arvo (Signalling Identifier), jonka perusteella MTP-kerros tunnistaa sen. ISUP:in SI-arvo on 5. Rajapinta ISUP-kerroksen ja MTP-kerroksen välillä (mtpif) määrittelee primitiivit MTP-3:n ja sen käyttäjien väliseen tiedon vaihtoon.

Mahdollisia ISUP-linkkejä voisi ohjelmistossa olla useampiakin, joten ne täytyy erottaa toisistaan, tässä AAL-tunnisteen (ATM Adaptation Layer) avulla. IsupNIaccessor erottaa tunnisteen MTP-3:n sanomasta. NI-protokolla (Network Interface) on osa joka koodaa ja dekoodaa viestit, ja sen tilakoneessa on ainoastaan yksi tila. Seuraava multiplekseri erottelee eri yhteydet, joille luodaan omat ISUP-oliot kanavatehtaan avulla. ISUP-yhteydet erotellaan Circuit Identification Code:n (CIC) avulla, joka käytännössä tarkoittaa tiettyä

aikaviipaletta aikajakokanavoinnissa. IsupAccessor erottaa CIC:n jo dekoodatusta ISUP-sanomasta. Ylimmän multiplekserin päällä olevat ISUP-oliot sisältävät standardin perusteella laaditun tilakoneen.

CC:n ja ISUP-kerroksen välinen rajapinta (sigif) on SCOMS-ohjelmistossa laadittu yleinen signalointirajapinta protokollien ja CC:n välisen tiedon vaihtamiselle.



Kuva 12. ISUP-protokollan rakenne SCOMS-ohjelmistossa.

JOHTOPÄÄTÖKSET

Sovelluskehyksen käyttö ohjelmakehityksessä nopeuttaa ja vähentää merkittävästi tarvittavaa ohjelmointityötä, vähentää virhemahdollisuuksia eikä rasita ohjelmoijaa triviaaleilla asioilla kuten kerrasta toiseen toistuvalla kopioinnilla. Ohjelmiston toteuttaja pääsee siis suunnitteluvaiheen jälkeen lähes suoraan toteuttamaan varsinaisia kokonaan uusia ohjelman osia, joita rakenteilla oleva sovellus vaatii.

Sovelluskehyksen rakentamiseen kannattaa käyttää aikaa vasta kun varmasti tiedetään, että sen avulla voidaan toteuttaa useampia uusia sovelluksia. Sovelluskehyksen käyttöönotto vaatii aikaa henkilöstön kouluttamiseen uusiin ajattelutapoihin ja välineisiin. Kaupallisessa ohjelmistotuotannossa onkin siis tarkasti mietittävä sovelluskehyksen rakentamisen kannattavuutta ennen ajatuksen varsinaista toteuttamista.

OVOPS++ on kypsynyt muutaman vuoden aikana ohjelmistoksi, joka sopii esimerkiksi tietoliikenneohjelmistojen toteuttamista helpottavista ohjelmistoista sekä kehittyneiden oliomenetelmien käytöstä. OVOPS++ on osoittanut käyttökelpoisuutensa tutkimusprojekteissa, ja se julkaistaan avoimen lisenssin alla vuoden 2000 alkupuolella. Sen tiimoilta tullaan todennäköisesti vielä näkemään opinnäytteitä. Eräs mielenkiintoinen aihe olisi OVOPS++:n suorituskyvyn vertailu johonkin kaupalliseen tai muuten suosittuun protokollasovelluskehykseen. Tästä voitaisiin paremmin päätellä oliomenetelmien käytön vaikutus suorituskykyyn.

Erikoistyö herätti keskustelua mahdollisesta jatkokehityksestä. Conduits+ on erinomainen mustalaatikkomalli, mutta mikäli OVOPS++:stakin haluttaisiin tehdä helppokäyttöisempi sovelluskehys jossa toteuttajalta peitetään kaikki toteutuksen kannalta epäoleellinen, pitäisi toteutusprosessia yksinkertaistaa enemmän.

Suunnittelumallien käyttö on johtanut siihen, että hyvin monet OVOPS++:n toiminnot toteutetaan aina vakiomenetelmällä. Olisi varmasti mahdollista toteuttaa jonkinlainen graafinen käyttöliittymä, jossa tilakoneiden luominen, kanavagraafin pystytys ja vastaavat yleiset, vakiomenetelmillä suoritettavat tehtävät suoritettaisiin yksinkertaisesti niitä kuvaavia symboleja yhdistelemällä. Generoituun koodiin voitaisiin sitten lisätä tarpeelliset protokollaspesifiset toiminnot. Ehkäpä tulevaisuudessa näemme jonkin jatkoprojektin joka mahdollistaa tämän.

Varsinainen erikoistyöprojekti, OVOPS++:n käyttöohjeen kirjoittaminen, pysyi suhteellisen hyvin aikataulussa. Alustava aikataulu olikin varsin joustavaksi laadittu, ja kun OVOPS++ oli jo alunperin tuttu, ei sen opettelemiseen tarvinnut käyttää aikaa. Enemmänkin aikaa vaati sopivien koodiesimerkkien etsiminen. Kysymyksessä on kuitenkin melko laaja ohjelmisto, joka perustuu mm. suunnittelumalleihin joissa toimintoja ei pakata yksittäisiin funktioihin tai luokkiin, vaan ne määrittelevät useiden luokkien yhteistoiminnan. Luonnollisesti tällaisten asioiden kuvaaminen lyhyissä esimerkeissä on vaikeaa, ja käyttöohjeessa viitataankin Design Patterns: Elements of Reusable Object-Oriented Software – kirjaan [GHJV95], joka lukijalla tulisi olla oppimisen tukena. Alustavan aikataulun toteutumisen seuraaminen oli hankalaa, koska tein muita töitä manuaalin kirjoittamisen yhteydessä. Manuaalin kirjoittaminen oli myös iteratiivinen prosessi, jossa OVOPS++:n parissa työskennelleet työntekijät arvioivat ja ehdottivat muutoksia “kehitysversioihin”, mistä syystä työn ei voida katsoa menevän suoraviivaisesti alusta loppuun. Esimerkkiohjelmien kirjoittamiseen käytettyä aikaa ei ole aikataulussa lainkaan, niiden lisääminen todennäköisesti tuplaisi ajan.

Kuva 13. Alustava aikataulu.

LÄHTEET

[Sui99a] Suihkonen, O.: OVOPS++ manual. TKK. Tietoliikenteen ja Multimedian laboratorio. 1999.

[Sui99b] Suihkonen, O.: ISUP implementation in the SCOMS project. TKK. Tietoliikenteen ja Multimedian laboratorio. 1999.

[Raa99] Raatikainen S.: B-ISUP protokollan toteutus. Tietotekniikan erikoistyöt, loppuraportti. LTKK. Tietotekniikan osasto. 1999.

[Toi97] Toivanen, I.: Yhteiskanavamerkinantoverkon siirtoyhteys-kerroksen oliopohjainen emulointi. Diplomityö. LTKK. Tietotekniikan osasto. 1997.

[Num99] Nummisalo, P.: Uudet olioteknologiat älyverkkototeutuksessa. Diplomityö. LTKK. Tietotekniikan osasto. 1999.

[Hün95] Hüni H., Johnson R., Engel R.: A Framework for Network Protocol Software. GLUE Software Eng., Bern, Switzerland, 1995. SIGPLAN Notices, Vol: 30, Iss: 10, pp 358-369.

[GHJV95] Gamma E., Helm R., Johnson R. Vlissides J.: Design Patterns, Elements of Reusable Objecct-Oriented Software. Addison-Wesley Publishing Company. 1995.