Timo Salminen

CORBA LANGATTOMASSA YMPÄRISTÖSSÄ

Tietotekniikan pro gradu -tutkielma

Ohjelmistotekniiikan linja

12/31/01

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Timo Salminen

Yhteystiedot: Sähköposti: tisalmin@st.jyu.fi

Työn nimi: CORBA langattomassa ympäristössä

Title in English: Wireless CORBA

Työ: Pro gradu -tutkielma

Sivumäärä: 69/81

Linja: Ohjelmistotekniikka.

Teettäjä: Jyväskylän yliopisto, tietotekniikan laitos

Avainsanat: Hajautetut järjestelmät, CORBA, GIOP

Keywords: Distributed Systems, CORBA, GIOP

Tiivistelmä: Tässä tutkielmassa käsitellään CORBA-menetelmää sekä sen siirtämistä langattomaan ympäristöön.

Abstract: This report discusses about CORBA standard and it’s performance in wireless environment.

Esipuhe

Hajautettujen järjestelmien arkkitehtuureissa on viimeisen kymmenen vuoden aikana tapahtunut paljon. Standardeista on yleisessä käytössä muutama, joista CORBA lienee tunnetuin ja käytetyin. Tästä huolimatta suomenkielisiä CORBA-kirjoja ei juuri ole saatavilla. Koska langattoman CORBAn ymmärtäminen edellyttää CORBA-standardin tuntemista, on tässä tutkielmassa käytetty aikaa myös sen selvittämiseen.

Tämän tutkielman kirjoittaminen alkoi syyskuussa 2001 ja tavoitteenani oli tutkielman valmistuminen ja samalla tutkinnon saaminen vuoden 2001 lopussa. Olen työssäni tutustunut CORBA-standardiin käytännössä ja tähän perustuu myös tämän tutkielman aihevalinta. Toivon, että tämän tutkielman luettuaan lukijalla on hyvä kuva CORBAsta langattomassa ympäristössä.

Haluan kiittää tämän tutkielman valmistumiseen vaikuttaneita henkilöitä: Kari Liuskaa, Vesa Lappalaista, Kimmo Raatikaista sekä ennen kaikkea Jaakko Kangasharjua, jonka ansioista tutkielmaan kuuluvan koejärjestelmän toteutus ja testaus voitiin suorittaa. Kiitos kuuluu myös opiskeluuni positiivisesti suhtautuneelle työnantajalleni. Merkittävimmän tuen opiskeluissani olen kuitenkin saanut vanhemmiltani, joille haluan osoittaa suurimmat kiitokset. Kiitos myös Mirvalle kärsivällisyydestä.

Termiluettelo

CORBA	Common Object Request Broker Architecture.
DCOM	Distributed Component Object Model. Microsoftin kehittämä hajautusmenetelmä, joka perustuu vahvasti Windows-käyttöjärjestelmään.
GIOP	General Inter-ORB Protocol. Protokolla, jota CORBAn oliovälittimet eli ORBit käyttävät keskinäisessä kommunikaatiossa.
GTP	GIOP Tunneling Protocol. Protokolla, jota käytetään GIOPin tunnelointiin.
Handoff	Kanavanvaihto. Tapahtuma, jossa mobiililaite vaihtaa yhdyssiltaa. Yhdyssilta “luovuttaa” laitteen toiselle yhdyssillalle.
IIOP	Internet Inter-ORB Protocol on GIOP-sovitus TCP/IP-ympäristöön.
LDAP	Lightweight Directory Access Protocol.
MICO	Mico is CORBA. MICO on ns. *open-source* –toteutus CORBA-standardista.
MIWCO	Langaton CORBA-laajennus MICOon.
NDS	Novell Directory Service. Novellin hakemistopalvelu.
OMG	Object Management Group. Yli 800 yrityksen muodostama ohjelmointistandardointia tekevä organisaatio.

OSF DCE	Opengroupin standardi hajautetulle ohjelmistoarkkitehtuurille.
RFP	Request for Proposal. OMG:n dokumentti, jolla haetaan esityksiä OMG-määritelmiksi.
RMI	Remote Method Invocation. Javassa käytetty menetelmä hajautettujen sovellusten toteuttamiseen.
SMS	Short Message Service. Lyhytsanomapalvelu.
SSL	Secure Socket Layer. Tietoturvastandardi.
UDP	User Datagram Protocol. TCP/IP-protokollaperheen yhteydetön kuljetusprotokolla.
UTP	UDP Tunneling Protocol. Käytetään tunnelointiin, jossa UDP-tietosähkeen datakentässä olevan UDP-viestin sisällä viedään GTP-viesti.
WAP	Wireless Access Protocol. Standardi tiedon esittämiseen ja siirtämiseen langattomille päätelaitteille.
WDP	Wireless Datagram Protocol. WAP-standardin kuljetuskerroksen yhteydetön protokolla.
WTP	WAP Tunneling Protocol. Protokolla, jota käytetään tunnelointiin WAPin kanssa (kuten UTP:tä UDP:n kanssa).
QoS	Quality of Service. Palvelunlaatu
X.500	ITU:n määrittelemä hakemistopalvelustandardi.

Sisältö

1 Johdanto

Tietoverkkojen ja päätelaitteiden kehityksen myötä hajautetut sovellukset ovat yleistyneet. Tämän päivän tavoitteita sovelluskehityksessä ovat toimintojen ja hallinnan keskittäminen sekä sovelluksen arkkitehtuurista riippuen ns. liiketoimintalogiikan (engl. *business logic*) hajauttaminen. Tunnetuin esimerkki hajautuksesta lienee kolmikerrosarkkitehtuuri (engl. *3-tier*), jossa erotetaan sovelluksen toiminnallisesti erilaiset osat kolmeen kerrokseen. Esimerkkinä kolmikerrosarkkitehtuurista käsitellään yleensä sovellusta, jossa käyttöliittymä, palvelin ja tietokanta muodostavat omat kokonaisuudet eli kerrokset.

Hajautetun sovelluksen toteuttamiseen on olemassa useita menetelmiä. Sokettiohjelmointia ja RPC-kutsuja (*Remote Procedure Call*) kehittyneempiä menetelmiä ovat Microsoftin DCOM, RMI sekä CORBA. Näistä CORBA on selvästi laajin ja käytetyin menetelmä, jonka etuna muihin on kieli- ja alustariippumattomuus. CORBA-sovellukset koostuvat luokista ja itsenäisesti suoritettavista ohjelmista. Ohjelmien suoritus voidaan hajauttaa eri tietokoneille ja samasta luokasta voi olla suorituksessa useampi ilmentymä. Kukin CORBA-olio toteuttaa rajapinnan, jonka tuntemalla muut oliot voivat sitä kutsua. CORBA-olio voisi olla esimerkiksi verkkokaupan ostoskori, jota käsittelevät tilaus- ja laskutusoliot [TC]. Nykyään CORBAa käytetään laajasti kaupan- ja hallinnon sovelluksissa, tietoliikennesovelluksissa, avaruus- ja ilmailusovelluksissa, liikennejärjestelmissä jne.

Tämä tutkielma käsittelee CORBA-standardin soveltamista langattomaan ympäristöön. Tutkielma perustuu suurimmaksi osaksi OMG:n (*The Object Management Group, Inc.*) spesifikaatioihin. CORBAn laaja käyttö, langattomien verkkojen kehittyminen sekä langattomien päätelaitteiden yleistyminen ovat synnyttäneet tarpeen käyttää CORBAa myös langattomassa ympäristössä. Käytön tulee perustua siihen, että olemassaoleviin CORBAan perustuviin järjestelmiin ei tarvita muutoksia, jotta ne voidaan liittää langattomaan ympäristöön.

Tämän ja monen muun vaatimuksen täyttämistä varten OMG:ssä on tehty paljon spesifiointityötä. Vuoden 2001 lopussa langattomaan CORBAn määrittelevä spesifikaatio oli viimeisteltävänä ja sen toteutus on ollut vapaasti saatavilla marraskuusta 2001 saakka. Määrittelytyössä merkittävä osuus on ollut Nokialla sekä Helsingin yliopistolla.

Langattomille päätelaitteille on ominaista huomattavasti työasemia rajoitetummat resurssit sovelluksen suoritusympäristössä. Mobiililaitteiden yleistyessä käyttäjät odottavat kuitenkin voivansa käyttää samoja palveluja langattomalta päätelaitteelta kuin työasemiltaan. Tämä asettaa uusia haasteita sovelluskehitykselle ja korostaa hajautetun ohjelmistoarkkitehtuurin merkitystä.

1.1 Tutkimusongelma

Tutkielman tarkoituksena on selvittää CORBA-standardiin tarvittavat muutokset käytettäessä sitä langattomassa ympäristössä. Lisäksi tutkielmassa pyritään selvittämään näiden muutosten vaikutuksia CORBAn suorituskykyyn sekä ensimmäisen langattoman CORBA-standardin toteutuksen, MIWCOn, toiminnallisuutta.

Ongelmien selvittämiseksi tutkielmassa toteutetaan esimerkkisovellus, jonka avulla pyritään toteamaan MIWCOn toimivuus sekä langattoman ympäristön vaatimien muutosten vaikutus CORBAn suorituskykyyn

1.2 Tutkielman rakenne

Luvussa 2 esitellään CORBA-standardia, josta uusin versio on 2.5. CORBAn perusteiden tunteminen on välttämätöntä, jotta lukija voi sisäistää langattoman ympäristön aiheuttamat lisävaatimukset olemassaolevalle standardille. Luvussa käsitellään CORBAn määrittelemä arkkitehtuuri ja oliopalvelut. Lisäksi esitellään CORBA-standardin soveltamista rajoitetuissa ja erityisvaatimuksia asettavissa ympäristöissä, sillä langattomiin päätelaitteisiin liittyy nykyisin vielä vahvoja suorituskykyyn ja muistin määrään liittyviä rajoituksia.

Esimerkkinä rajoitetusta ympäristöstä voidaan siis pitää langatonta ympäristöä. Luvussa 3 käsitellään CORBAa langattomassa ympäristössä. Luvussa selvitetään langattoman ympäristön vaatimukset ja erot CORBA-standardille kiinteään verkkoon nähden. Lisäksi tarkastellaan tarkemmin CORBA-arkkitehtuuria langattomassa ympäristössä.

Luvussa 4 määritellään ja toteutetaan yksinkertainen langatonta CORBA-toteutusta käyttävä järjestelmä, jossa selvitetään eroja langattoman ja kiinteän verkon välillä suorituskyvyn kannalta ja todetaan standardia vastaavan toteutuksen toimivuus.

Käytettävissä olevat resurssit eivät valitettavasti mahdollistaneet CORBAn testausta aidossa langattomassa ympäristössä, vaan testauksessa jouduttiin langatonta ympäristöä simuloimaan TCP/IP-verkossa. Saatujen tulosten perusteella vasteajan kasvu langatonta ympäristöä simuloitaessa oli nelinkertainen. Tämän vasteajan jakaantumista kuljetuksen erivaiheisiin selvitettiin verkkoanalysointiohjelman avulla.

2 CORBA-standardi

Tässä luvussa esitellään CORBA 2.5 -standardin määrittelemä arkkitehtuuri ja yleiset piirteet. Luku perustuu pääasiassa lähteisiin [OMG1] ja [OMG2].

CORBA (*Common Object Request Broker Architecture*) on OMG:n (*The Object Management Group, Inc*.) määrittelemä standardi hajautetulle ohjelmistoarkkitehtuurille. CORBAn rakenne sallii erilaisten oliojärjestelmien integroimisen. OMG on yli 800 yrityksen muodostama kansainvälinen organisaatio, joka perustettiin vuonna 1989. OMG:n päätavoitteet ovat uudelleenkäytettävyys, siirrettävyys ja yhteensopivuus olioperustaisessa ohjelmistonkehityksessä.

2.1 Esimerkkejä käyttökohteista

Kieli- ja alustariippumattomuus ovat tehneet CORBAsta käytetyn menetelmän monella osa-alueella, kuten avaruus- ja ilmailualan, puolustusvoimien, tietoliikenteen sekä kaupan- ja hallinnon sovelluksissa. Esimerkiksi Yhdysvaltain avaruushallinnon, NASAn, MOPSS-järjestelmä (*The Mission Operations Planning and Scheduling System*) perustuu CORBAan. MOPSS on järjestelmä, jonka avulla voidaan selviytyä nopeasti monimutkaisten sovellusten suunnittelu- ja aikataulutustehtävistä sekä optimoida käytettävissä olevien resurssien tehokas käyttö.

Myös Berliinin liikenneinfojärjestelmän toteutus perustuu CORBAan. Järjestelmä on verkkoversio olemassaolevasta HAFAS-järjestelmästä, jonka Cd-rom-versio on käytössä yli 40 liikenneyhtiöllä Euroopassa. Tämän järjestelmän avulla ihmiset voivat rakentaa aikataulunsa kotona. HAFAS-sovellusta voitiin aiemmin suorittaa kotikoneelta tai ns. Travel-Info -päätteeltä. Koska päätteet eivät olleet verkossa, oli päivittäminen hankalaa. Uuden järjestelmän ja päätteiden verkotuksen avulla päivitysongelma poistui [CORBA].

Tietoliikenteessä CORBAa on käytetty useissa palvelinsovelluksissa (kuten SMS-palvelimissa) ja verkonhallintasovelluksissa. CORBAa käytetään myös erityisiä vaatimuksia asettavissa sovelluksissa, kuten reaaliaikajärjestelmissä ja resursseiltaan

rajoitetuissa järjestelmissä. Näitä suoritusympäristöjä varten CORBA sisältää omat määritykset [ks. luvut 2.7 ja 2.8].

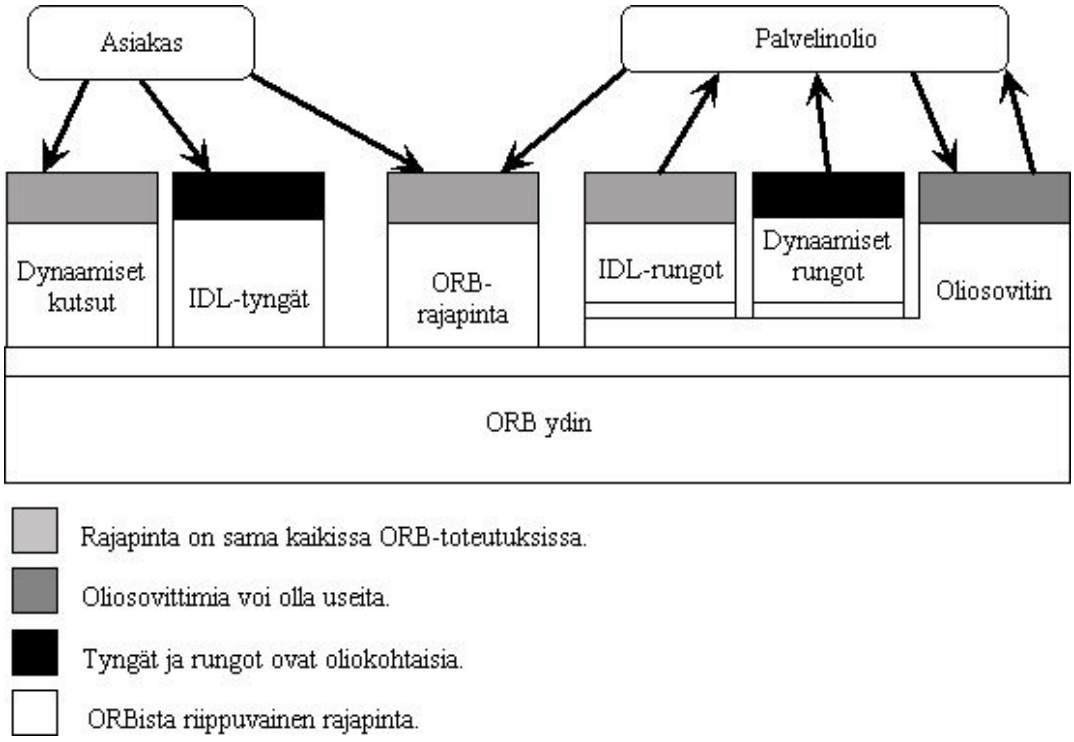
2.2 Käsitteitä

Ennen arkkitehtuurin kuvaamista on syytä esitellä CORBAn kannalta oleellisia käsitteitä. **Asiakas** (engl. *Client*) on kokonaisuus, joka käyttää määriteltyjä **palveluita** (engl. *service*). Palveluiden käyttö tapahtuu **palvelupyynnöillä** (engl. *request*). Palvelut on toteutettu **palvelinoliossa** (engl. *Servant*). Palvelupyynnöt on määritelty asiakkaan ja palvelimen tuntemassa **rajapinnassa** (engl. *interface*). **Oliovälitin** (engl. *Object Request Broker*, ORB) huolehtii palvelupyyntöjen välittämisestä asiakkaalta palvelinoliolle ja vastaavasti palautusarvojen tai poikkeusten välittämisestä palvelinoliolta asiakkaalle. Näihin tehtäviin kuuluu myös palvelinolioiden paikallistamisen, palvelinolion alustuksen sekä datan formaatin muunnoksista (engl. *marshaling*, *remarshaling*) [OMG1]. Oliovälitin voi olla erillinen ohjelma (*Server-based ORB*), ohjelmakirjasto (*Library-based ORB*) tai se voi olla liitettynä suoraan ohjelmakoodiin (*Client- and Implementation-resident ORB*) tai käyttöjärjestelmään (*System-based ORB*) [OMG1]. Tavallisesti ORB-toteutukset ovat ohjelmakirjastoja.

CORBAssa asiakas-palvelin -käsite ei ole perinteisen yksiselitteinen. Olio voi olla asiakas jollekin oliolle ja palvelin toiselle. Tämän lisäksi palvelinolio voi kutsua asiakkaan metodeja. Tätä menetelmää kutsutaan *callback*-menetelmäksi. Tällöin asiakkaan toteutus käsittää olion, joka toteuttaa rajapinnan vastaavasti kuin palvelinolio. Callback-palvelimen olennaisin ero tavalliseen CORBA-palvelimeen on siinä, että asiakkaan on välitettävä callback-palvelimensa viite palvelinoliolle ennen kuin palvelinolio voi suorittaa callback-kutsuja. Palvelinolio ei siis paikallista asiakkaan callback-palvelinta, kuten asiakas paikallistaa palvelinolion.

2.3 Arkkitehtuuri

Kuvassa 1 on esitetty ORBin arkkitehtuuri. Rajapinnat on esitetty erivärisinä suorakulmioina ja nuolet ilmaisevat kutsun suunnan. Rajapinnat määritellään IDL:llä (*Interface Definition Language*). Asiakas voi tehdä palvelinoliolle osoitettuja palvelupyyntöjä dynaamisen kutsurajapinnan tai IDL-tyngän (engl. *IDL stub*) avulla. ORB rajapinta tarjoaa asiakkaalle pääsyn erilaisiin ORB-palveluihin, joita käsitellään tarkemmin luvussa 2.3. Palvelinolio vastaanottaa palvelupyynnön joko dynaamiselta rungolta (engl. *dynamic skeleton*) tai IDL-rungolta (engl. *IDL skeleton*). Myös palvelinolio käyttää ORBin palveluita. Lisäksi se käyttää oliosovitinta (engl. *object adapter*) palvelua suoritettaessa tai kommunikoinnissa oliovälittimen kanssa.



Kuva 1. ORB-arkkitehtuuri.

2.3.1 Rajapinnat

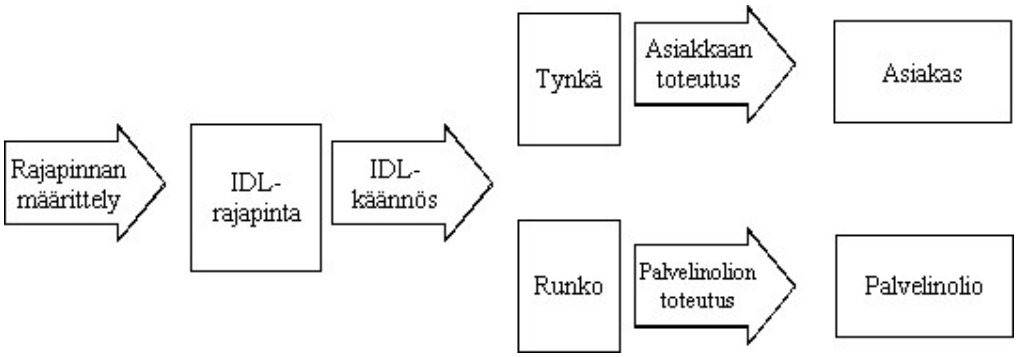
Rajapinnat voidaan määritellä staattisesti OMG:n IDL-kielellä (*Interface Definition Language*). IDL määrittelee rajapinnan kutsut (eli palvelupyynnöt) ja niihin liittyvät parametrit ja palautusarvot sekä poikkeukset. Lisäksi rajapintoja voidaan varastoida rajapintavarastoon (engl. *Interface Repository*), josta rajapintoja vastaavat oliot voidaan saada ajon aikana.

IDL:llä määritellään olion tyyppi sen toteuttaman rajapintamäärittelyn kautta. Tämän rajapinnan avulla palvelinolio kertoo mahdollisille asiakkaille palveluistaan. IDL:llä tehdyistä rajapinnoista käännetään (engl. *mapping*) tyngät ja rungot ohjelmointikielelle tarkoitusta varten tehdyllä kääntäjällä. Mikäli IDL-rajapinta halutaan kääntää esimerkiksi jollekin oliokielelle, tekee kääntäjä tietotyyppien muunnokset ja tuottaa tarvittavat luokat. Näitä luokkia ovat tyngät, rungot, dynaamiset kutsurajapinnat, oliosovittimet ja ORB-rajapinta. Käännökset määrittelevät myös oliokutsujen ja prosessin suorituksesta vastaavien säikeiden välisen suhteen. Tavallisesti suoritusvuoro siirtyy palvelupyynnön mukana palvelinoliolle ja palaa asiakkaalle, kun palvelu on suoritettu.

2.3.2 Palvelupyynnöt

Jotta asiakas voi suorittaa kutsun palvelinoliolle, on sillä oltava palvelinolion objektireferenssi (engl. *Object Reference*). Objektireferenssi sisältää kaiken tarvittavan tiedon palvelinoliosta, jotta ORB voisi sen tunnistaa. Objektireferenssin saamiseen on useita eri menetelmiä. Objektireferenssit voivat poiketa toisistaan eri ORB-toteutuksilla. Asiakkaan on tunnettava myös palvelinolion tyyppi ja palvelupyynnöt. Ohjelmoijan kannalta kutsu tehdään samoin kuin minkä tahansa paikallisen olion ollessa kyseessä. ORB saa palvelupyynnön siis tyngältä tai dynaamisen kutsurajapinnan kautta. Tämän jälkeen ORBin tehtävänä on paikallistaa oikean palvelinolion toteutus, välittää parametrit ja siirtää kutsu sekä suoritusvuoro eteenpäin IDL-rungolle tai dynaamiselle rungolle. Palvelua suorittaessa palvelinolio voi käyttää joitain ORBin palveluja oliosovittimen kautta. Kun palvelupyyntö on suoritettu, palautetaan vaste ja suoritusvuoro asiakkaalle.

Mikäli palvelupyynnöt toteutetaan staattisesti IDL-tyngillä ja -rungoilla, tuottaa IDL-kääntäjä tyngän ja rungon kullekin rajapinnalle. Tynkä edustaa palvelinolioa siten, että palvelua kutsuttaessa kutsutaankin tyngän metodia. Kutsu vastaa paikallista metodikutsua. Tynkä kutsuu edelleen ORBia ORB-kohtaisen rajapinnan kautta [ks. kuva 1]. Tästä johtuen, mikäli käytetään useampaa kuin yhtä ORB-toteutusta, voi kullakin rajapinnalla olla ORB-kohtainen tynkä ja runko. Rungossa on määritelty operaatiot, jotka palvelinolion on toteutettava. ORB kutsuu palvelinolion operaatiota rungon kautta. Toisaalta rungon olemassaolo ei aina takaa vastaavan tyngän olemassaoloa, sillä asiakkaat voivat suorittaa palvelupyyntöjä myös dynaamisen kutsurajapinnan kautta. Kuvassa 2 on esitetty vaiheet asiakkaan ja palvelinolion toteutukselle, kun käytetään staattisia tynkiä ja runkoja.



Kuva 2. CORBA-järjestelmän toteutus.

Dynaamista kutsurajapintaa käytettäessä kutsut palvelinoliolle muodostetaan dynaamisesti. Tällöin asiakas voi määritellä kutsuttavan palvelinolion, palvelupyynnön sekä palvelupyyntöön tai palvelupyyntöihin liittyvät parametrit. Asiakkaan toteutuksen tulisi sisältää tieto suoritettavasta palvelupyynnöstä ja sen parametrien tyypistä. Nämä tiedot voidaan myös hakea rajapintavarastosta tai jostain muusta ajonaikaisesta varastosta. Dynaamiset rungot mahdollistavat dynaamiset kutsut palvelinolion päässä. Näiden avulla voidaan selvittää tarjolla olevat palvelupyynnöt ja niiden parametrit. Palvelinolion toteutuksen on dynaamisessa menetelmässä kuvattava ORBille kaikki palvelupyyntöjen parametrit. Dynaamisia runkoja voidaan kutsua dynaamisen kutsurajapinnan lisäksi myös tynkien kautta. Dynaamisten kutsujen käyttö käytännön toteutuksissa on harvinaista.

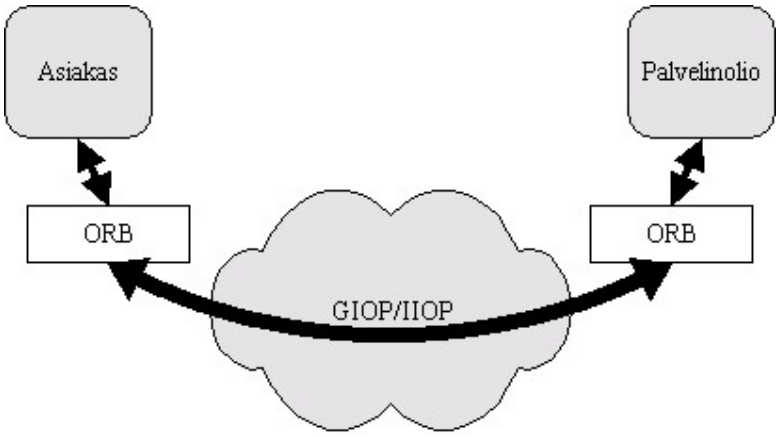
2.3.3 Oliosovitin ja toteutusvarasto

Palvelinolio käyttää ensisijaisesti ORBin palveluita oliosovittimen kautta. Näitä palveluja ovat mm. objektireferenssien muodostaminen ja tulkitseminen, palvelupyyntöjen esittäminen, tapahtumien turvallisuus, olion aktivointi ja deaktivointi, objektireferenssien tulkitseminen olioiksi sekä palvelinolioiden toteutusten rekisteröinti. Oliosovittimien avulla ORB voi ryhmitellä samantyyppisiä palvelinolioita. CORBA-standardi määrittelee POA-oliosovittimen (*Portable Object Adapter*), jota voidaan käyttää useimpien ORBien ja palvelinolioiden kanssa. POA:n tarkoituksena on tarjota oliosovitin, jota voidaan käyttää useiden ORB-toteutusten kanssa [OMG1].

Toteutusvarasto sisältää tietoa, jota ORB tarvitsee palvelinolioiden paikallistamiseen ja aktivoimiseen. Toteutusvarastojen informaatiosta suurin osa on ORB- tai suoritusympäristökohtaista.

2.4 ORBien välinen kommunikointi

Tavallisesti asiakkaan ja palvelinolion väliseen kommunikaatioon liittyy useampi kuin yksi oliovälitin. Kuvassa 3 on esitetty yksinkertaisen CORBA-järjestelmän arkkitehtuuri, jossa asiakas ja palvelinolio sijaitsevat fyysisesti eri laitteilla. Tällöin asiakkaalla ja palvelimella on oma oliovälitin, jolloin oliovälittimien välinen kommunikointi tulee oleelliseksi osaksi järjestelmää.



Kuva 3: Yksinkertainen CORBA-järjestelmä TCP/IP-verkossa.

ORBien välisessä kommunikaatiossa käytetään usein termiä **toimialue** (engl. *domain*). Yleisesti ajatellen toimialue on joukko olioita, joilla on samankaltaisia ominaisuuksia tai jotka noudattavat samoja sääntöjä. Toimialueita voi olla useita tyyppejä, kuten esimerkiksi hallinta-, nimeämis-, kieli-, turvallisuus- ja teknologiatoimialueita. Näin ollen erityyppisten toimialueiden rajat voivat mennä limittäin. Toimialue voidaan mallintaa olioksi ja se voi olla osana toista toimialuetta. Yleisellä tasolla toimialueet jaetaan hallinnollisiin (engl. *administrative*) ja teknologisiin (engl. *technological*) toimialueisiin.

Toimialueiden välinen kommunikaatio on mahdollista, jos toimialueiden välisiin muunnoksiin (engl. *mapping*) on olemassa hyvin määritellyt säännöt. Toimialueita yhdistävät **sillat** (engl. *bridge*), jotka suorittavat tarpeelliset muunnokset toimialueiden välillä. ORBien välisen kommunikoinnin kannalta yksinkertaisimmassa toimialuemallissa jaetaan olio toimialueisiin ORB-toteutuksen mukaan. Jotta toimialueiden (tässä siis ORB-toteutusten) välinen kommunikointi olisi tehokasta on toimialueiden välisen sillan toimittava yhtä hyvin kumpaankin suuntaan [OMG1].

Toimialueiden välillä voi olla eroja objektireferenssien sisäisessä esitystavassa. Jotta objektireferenssien siirto toimialueiden välillä olisi yhdenmukaista on CORBA-standardissa määritelty erityinen objektireferenssi, **IOR** (*Interoperable Object Reference*).

2.4.1 IOR (*Interoperable Object Reference*) ja IOR-profiilit

IOR-objektireferenssit yhdenmukaistavat objektireferenssien siirron toimialueiden välillä. Objektireferensseillä on vähintään yksi tagi-profiili (engl. *tagged profile*). Kukin profiili sisältää riittävästi tietoa palvelupyynnön suorittamiseen profiilin tukemilla protokollilla. Profiili voi tukea yhtä tai useampaa protokollaa. Palvelupyynnön vastaanottava oliovälitin käsittelee itse asiassa profiileja eikä palvelupyynnön parametrinä siirrettyä kokonaista IORia. Oliovälitin luo tämän perusteella referenssin, joka vastaa toimialueen sisäistä esitystapaa. Oliovälitin toimii siis siltana toimialueiden välillä, joissa referenssien esitystavat poikkeavat toisistaan.

Standardeja IOR-profiileja ovat `TAG_INTERNET_IOP` IIOP:lle, `TAG_MULTIBLE_COMPONENTS` sekä `TAG_SCCP_IOP`. IIOP-profiili sisältää

palvelinolion isäntä- ja porttiosoitteet sekä oliotunnisteen (engl. *object key*). Profiilien tarkempi rakenne on kuvattu lähteessä [OMG1].

2.5 GIOP (General Inter-ORB Protocol)

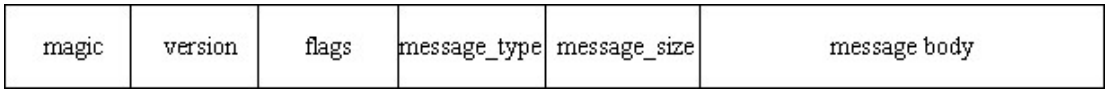
GIOP on protokolla ORBien väliseen kommunikaatioon. Se on suunniteltu toimimaan millä tahansa yhteydellisellä kuljetusprotokollalla. GIOP on pyritty pitämään yksinkertaisena. IDL:n tyyppimääritykset muutetaan siirtoa varten alemman tason esitysmuodoksi (oktettivirta, engl. *octet stream*) CDR-syntaksin (*Common Data Representation*) mukaan. CDR määrittelee esitystavat kaikille OMG IDL:n tietotyypeille, määrittelemättömille tietotyypeille (engl. *any*) sekä toteutusriippuvaisille tietotyypeille.

GIOP:n lisäksi CORBA-standardi määrittelee ORBien väliseen kommunikointiin myös IIOP:n, joka käytännössä sovittaa GIOP:n TCP/IP-verkoille. TCP/IP-protokollaperheestä kerrotaan tarkemmin lähteessä [S1]. IIOP perustuu siis GIOP:iin. Kunkin CORBA-yhteensopivan ORBin on tuettava IIOP:tä tai tarjottava silta ORBin ja IIOP:n välille. IIOP:n avulla asiakas voi tehdä palvelupyyntöjä Internet-verkon yli. IIOP:n lisäksi CORBA-standardissa määritellään ESIOP (*Environment-Specific Inter-ORB Protocol*), jolla saavutetaan ORBien yhteensopivuus erilaisien verkkojen välillä [O1].

2.5.1 GIOP-viestin kehysrakenne

GIOP määrittelee formaatit ORBien välisille viesteille. Seitsemällä viestityypillä hoidetaan ORBien välinen kommunikaatio. GIOP-viestit tukevat kaikkia CORBAn vaatimia toimintoja. GIOP sisältää toiminnallisuuden asymmetrisen yhteyden toteuttamiseksi, jossa GIOP määrittelee kaksi roolia: asiakkaan ja palvelimen. Asiakas muodostaa yhteyden sekä lähettää palvelupyynnön ja palvelin vastaanottaa pyynnön sekä lähettää vastauksen. Mikäli samaan oliovälittimeen liittyy useampi asiakas, mahdollistaa GIOP jaetun yhteyden palvelinolioon asiakkaiden kesken. Tätä menetelmää kutsutaan multipleksaukseksi (engl. *multiplexing*). GIOPissa pyyntö- ja vastausviesteihin liitetään yksikäsitteinen tunniste, jolla vastaukset voidaan liittää oikeisiin pyyntöihin. GIOP sisältää viestit myös yhteydenhallintaan.

Kuvassa 4 on esitetty GIOP 1.2 -viestin kehysrakenne. `Magic`-kenttä identifioi GIOP-viestit. Kentän arvo on aina neljä isoa kirjainta ”GIOP”. `Version`-kentässä ilmoitetaan GIOP-protokollan versio. GIOP-versioissa saattaa olla eroja otsikon suhteen, joten versionumero on ilmoitettava heti otsikon alussa. Tiettyä GIOP-versiota tukevan palvelimen on tuettava myös vanhempia GIOP-versioita. Seuraavat kahdeksan bittiä on varattu lipuille (engl. *flags*). Biteistä kuusi eniten merkitsevää on varattu ja ne saavat versiossa 1.2 aina arvon 0. Vähiten merkitsevä bitti ilmoittaa käytetyn tavujärjestyksen (engl. *byte order*) ja toiseksi vähiten merkitsevä bitti ilmoittaa, oliko kehys viestin viimeinen. Lippujen jälkeen ilmoitetaan viestin tyyppi [ks. taulukko 1] `message_type` -kentän avulla. Viestin koko (pl. otsikon 12 tavua) ilmoitetaan oktetteina `message_size`-kentässä. `Request`-, `LocateRequest`- ja `LocateReply` -viestien `message_size`-kenttä on varattu tulevaisuuden käyttöä varten ja saa aina arvon 0 [ks. luku 2.5.2]. Itse GIOP-viesti kuljetetaan kehyksen runko-osassa (engl. *message body*). GIOPin versiossa 1.0 ei käytetä lippuja, mutta myös siinä ilmoitetaan otsikossa käytetty tavujärjestys.



Kuva 4. GIOP 1.2 –kehys.

2.5.2 GIOP-viestit

Kukin GIOP-viesti alkaa siis GIOP-otsikolla. Lisäksi GIOPin viestityypillä voi olla oma otsikko-osansa ja viestirunko. Näissä otsikoissa on tietoa mm. suoritettavasta operaatiosta ja palvelinolion identiteetistä. Lisäksi viestikohtaiset otsikot sisältävät tiedon, jolla vastaukset liitetään oikeisiin pyyntöihin. GIOP 1.2 -standardin määrittelemät viestit on kuvattu taulukossa 1. Asiakas ja palvelin eivät voi lähettää kaikentyyppisiä viestejä. Viestin tyyppiä edustaa viestin otsikossa kuljetettava kokonaisluku (tyyppiarvo).

Viesti	Lähettäjä	Tyyppiarvo
`Request`	Asiakas	0
`Reply`	Palvelin	1
`CancelRequest`	Asiakas	2
`LocateRequest`	Asiakas	3
`LocateReply`	Palvelin	4
`CloseConnection`	Palvelin	5
`MessageError`	Molemmat	6
`Fragment`	Molemmat	7

Taulukko 1. GIOP-viestit.

`Request`-viestillä kuljetetaan palvelupyynnöt asiakkaalta palvelinoliolle. `Request`-viesti koostuu kolmesta elementistä, jotka ovat GIOP-otsikko, request-otsikko sekä request-runko. Request-otsikko sisältää pyynnön tunnisteen, jolla vasteet osataan liittää oikeisiin pyyntöihin. Lisäksi request-otsikossa on lippukenttä, palvelinolion osoite, palvelupyyntö sekä siihen liittyvä informaatio. Request-runko sisältää palvelupyyntöön liittyvät parametrit.

`Request`-viestiin vastataan `Reply`-viestillä, jossa on myös kolme elementtiä. Nämä ovat GIOP-otsikko, reply-otsikko sekä reply-runko. Reply-otsikkoon liitetään Request-viestin palvelupyyntötunniste sekä informaatio operaation onnistumisesta. Lisäksi reply-otsikko sisältää oliovälitinpalveluihin liittyvää tietoa. Vastauksen rungossa palautetaan joko poikkeus tai operaation mahdolliset parametrit ja palautusarvo.

`CancelRequest`-viestillä ilmoitetaan palvelinoliolle, ettei asiakas enää odota vastausta palvelupyyntöön. Viesti koostuu GIOP-otsikon lisäksi CancelRequest-otsikosta Tämä sisältää tunnisteen palvelupyynnölle, jota viesti koskee.

`LocateRequest`-viestillä asiakas voi selvittää olioviitteestä, pystyykö palvelinolio vastaanottamaan pyyntöjä suoraan olioviitteellä ja jos ei, niin mihin osoitteeseen olioviitteelle tarkoitetut viestit tulee lähettää. Vastauksena palvelin lähettää `LocateReply`-viestin, joka koostuu GIOP-otsikosta, LocateReply-otsikosta sekä

rungosta. LocateReply-otsikossa ilmoitetaan pyynnön tunniste sekä informaatiota vastauksesta. Viestin runko on yleensä tyhjä, mutta tietyissä virhe- ja välitystilanteissa se sisältää informaatiota.

`CloseConnection`-viestillä ilmoitetaan asiakkaan ja palvelimen välisen yhteyden sulkemisesta. Viesti koostuu pelkästä GIOP-otsikosta. `MessageError`-viestillä vastataan mihin tahansa GIOP-viestiin, jossa GIOP-versio on tuntematon tai GIOP-otsikko on virheellinen. Myös tämä viesti koostuu pelkästä GIOP-otsikosta.

GIOP 1.2:ssa voidaan `Request`, `Reply`, `LocateRequest` ja `LocateReply` viestit voidaan pilkkoa paketteihin. `Fragment`-viesti lähetetään edellä lähetetyn pyyntö- tai vasteviestin jälkeen, mikäli viestin pilkkominen on osoitettu lipuilla.

2.5.3 GIOP-viestin kuljetus

GIOP on suunniteltu siten, että se voidaan toteuttaa usealle eri kuljetusprotokollalle. GIOP olettaa alla sijaitsevan kuljetusprotokollan olevan yhteydellinen ja luotettava. Lisäksi se olettaa että siirto nähdään oktettivirtana, kuljetuskerros ilmoittaa virhetilanteista ja että yhteyden muodostus vastaa jokseenkin TCP/IP:n mallia [S1]. Tämän mukaan palvelinpää ei alusta yhteyksiä, mutta voi hyväksyä tai hylätä asiakkaan lähettämiä yhteyspyyntöjä. Kun yhteys on avattu, kumpi tahansa voi sulkea sen. CORBAssa asiakas alustaa yhteyden IORin avulla. GIOP-yhteydellä voidaan välittää ainoastaan GIOP-viestejä [ks. taulukko 1].

GIOP sisältää tuen palvelinolioiden paikantamiseen. Viestin kohdeosoite ei välttämättä ole palvelinolion osoite, vaan se voi olla jonkin agentin osoite, jonka kautta yhteys muodostetaan. Agentti on siis eräänlainen välikäsi, jolla voi olla seuraavanlaisia rooleja [OMG1]:

- Agentti voi ottaa vastaan palvelupyyntöjä ja palauttaa vasteita niihin. Se voi omistaa palvelinolion toteutuksen tai se voi toimia siltana ja välittää palvelupyynnön eteenpäin. Oleellista on, että palvelupyyntö voidaan lähettää suoraan agentille.
- Agentti ei voi ottaa palvelupyyntöjä vastaan, mutta se voi toimia paikantajana (engl. *location service*).
- Agentti voi ottaa vastaan joidenkin olioiden palvelupyyntöjä ja toimia paikantajana toisille olioille.
- Agentti voi jonain ajankohtana vastaanottaa palvelupyyntöjä ja toisena ajankohtana toimia paikantajana.

2.5.4 Kaksisuuntainen GIOP

GIOPin versioissa 1.0 ja 1.1 GIOP-yhteys on asymmetrinen. Tämä nousi huomattavaksi rajoitteeksi etenkin käytettäessä palomuureja. GIOP 1.2:ssa muutettiin yhteyskäytäntöä siten, että palvelinolio saattoi tehdä kutsuja asiakkaalle samalla GIOP-yhteydellä. Tätä menetelmää kutsutaan kaksisuuntaiseksi GIOPiksi (engl. *Bi-Directional GIOP*). Jos yhteyttä käytetään kaksisuuntaisena, tulisi asiakkaan ylläpitää yhteyttä niin kauan, kuin palvelimen pyytämät palvelut on suoritettu. Tällöin asiakas voi kutsua palvelimelle tarkoitettuja viestejä ja vastaavasti palvelin asiakkaalle tarkoitettuja [ks. taulukko 1]. Jotta palvelin voi kutsua asiakkaan olion metodeja, täytyy sen tuntea kyseisen olion IOR. Menetelmää, jossa palvelinolio kutsuu asiakkaan metodeja, kutsutaan *callback*-menetelmäksi [OMG1].

2.6 CORBA-palvelut

OMG on määritellyt 15 palvelustandardia, joita esitellään tässä luvussa yleisellä tasolla. CORBA-palvelut (engl. *CORBAservices*) ovat järjestelmätason palveluita, jotka on määritelty IDL-tiedostoissa. Palveluita voidaan käyttää komponenttien luomiseen, nimeämiseen ja niiden esittelyyn järjestelmälle. Lisäksi ne tarjoavat erilaisia mahdollisuuksia palvelinolioiden paikallistamiseen. CORBA-palveluita ovat

- nimipalvelu,
- tapahtumapalvelu,
- POS,
- elinkaaripalvelu,
- rinnakkaisuudenhallintapalvelu,
- suhdepalvelu,
- ulkoistamispalvelu,
- transaktiopalvelu,
- kyselypalvelu,
- lisensointipalvelu,
- ominaisuuspalvelu,
- aikapalvelu,
- turvallisuuspalvelu,
- liikepalvelu sekä
- oliokokoelmapalvelu.

Palvelumäärittelyt tekevät CORBA-standardista varsin laajan. Usein CORBA-toteutuksista on jätetty toteuttamatta joitain tässä luvussa käsiteltyjä palvelustandardeja. Tämä onkin välttämätöntä, kun halutaan pitää ORBin koko pienenä. Tämä tarve esiintyy käytettäessä CORBAa rajoitetussa ympäristössä (esim. sulautetut järjestelmät ja langattomat päätelaitteet). Tarkemmat määritykset palveluista ja niiden sisältämistä rajapinnoista on luettavissa lähteestä [OMG2].

2.6.1 Nimipalvelu

Nimipalvelu (engl. *Naming Service*) on eräs keskeisimpiä CORBAn määrittelemiä palveluita. Sen avulla palvelinolio voidaan sitoa **nimikontekstiin**. Nimikonteksti on olio, jossa kukin nimi on yksikäsitteinen. Nimipalvelusta saadaan nimellä haettua referenssi sitä vastaavaan palvelinolioon. Nimipalvelun nimeämiskäytäntö voi perustua esimerkiksi ISO x.500 –standardiin [SU1], OSF DCE:hen [OSF1], Novellin NDS:ään[N1] tai LDAP:iin [IETF1]. Palvelinolio sitoo (engl. *bind*) jonkin nimen nimipalvelussa itseensä ja tämän

jälkeen se löydetään tällä nimellä. Nimipalvelu ei ota kantaa palautettavan viitteen objektityyppiin, joten asiakkaan on tunnettava palvelinolion tyyppi.

Nimikonteksteja voidaan liittää toisiinsa, jolloin saadaan hyvin paljon hakemistojärjestelmää muistuttavia tietorakenteita. Tässä nimikontekstit vastaavat hakemistoja. Itse nimi on itse asiassa järjestetty komponenttijono, jossa komponentit viimeistä lukuun ottamatta ovat nimikontekstien nimiä (vrt. tiedoston hakemistopolku) [OMG2].

2.6.2 Tapahtumapalvelu

Palvelupyyntö johtaa synkroniseen operaation suoritukseen palvelinoliossa. Jos palvelupyyntöön liittyy parametrejä, siirretään asiakkaan ja palvelinolion välillä dataa. Palvelupyyntö välitetään tietylle palvelinoliolle. Jotta palvelun suoritus olisi onnistunut, on sekä asiakkaan, että palvelimen oltava käytettävissä. Jos palvelinolioon ei saada yhteyttä, saa asiakas tästä poikkeuksen, joka sen on käsiteltävä.

Joskus on syytä erotella edellä kuvatun kommunikaation osia. Tapahtumapalvelu (engl. *Event Service*) suorittaa tätä. Sen avulla komponentit voivat rekisteröidä kiinnostuksensa tiettyjä tapahtumia kohtaan.

Tapahtumapalvelu määrittelee kaksi roolia olioille, jotka ovat varustaja (engl. *supplier*) ja kuluttaja (engl. *consumer*). Varustajat tuottavat tapahtumainformaatiota ja kuluttajat käsittelevät sitä. Tapahtumainformaatio liikkuu siis varustajien ja kuluttajien välillä. Tapahtumakommunikaation voi aloittaa joko varustaja lähettämällä tapahtumia kuluttajille, tai kuluttaja kyselemällä tapahtumia varustajilta. Tapahtumapalvelun **tapahtumakanava** (engl. *event channel*) sijaitsee varustajien ja kuluttajien välissä. Se mahdollistaa useiden varustajien ja kuluttajien asynkronisen kommunikoinnin [OMG2]. Tapahtumakanava siis kerää ja jakaa tietoa tapahtumista. [O1].

2.6.3 POS

Vaikka CORBA mahdollistaa pysyvän olioreferenssin, ei se tarjoa mahdollisuutta palvelinolioiden tilojen tallentamiseen. POSin (*Persistent Object Service*) tarkoituksena on täyttää tämä tarve. POSin tehtävänä on siis säilyttää tietoa olioiden tilasta. Palvelua käytetään yhdessä jonkin muun palvelun, kuten nimipalvelun, kanssa.

Tilat jaetaan kahteen luokkaan, jotka ovat dynaaminen ja pysyvä (engl. *persistent*) tila. Dynaaminen tila ei yleensä säily koko olion elinaikaa. Dynaamisen tilan informaatio saattaa kadota esimerkiksi järjestelmävirheen seurauksena. Pysyvää tilatietoa käytetään dynaamisen tilan uudelleenmuodostamiseen.

Tilojen tallettamista varten palvelinolion on kyettävä määrittelemään tilat, jotka halutaan tallettaa. Vastaavasti palvelinolion on pystyttävä tilatiedon perusteella siirtymään vastaavaan tilaan. POS-arkkitehtuuriin kuuluu varasto, jonne tilatieto talletetaan. Tietovarastoja voivat olla esimerkiksi POSIX-tiedostot tai SQL-tietokannat. POS tarjoaa asiakkaalle mahdollisuudet hallita palvelinolion tiloja tai vaikuttaa niiden hallintaan. Asiakkaan kannalta mielenkiintoista on tilan tallettamisen tai palauttamisen ajoitus sekä palvelinolion tilat [OMG2].

2.6.4 Elinkaaripalvelu

Elinkaaripalvelu (engl. *Life Cycle Service*) määrittelee käytännöt olioiden luomiseen, tuhoamiseen, kopioimiseen ja siirtämiseen. Asiakas voi suorittaa näitä ns. elinkaarioperaatiota olioille niiden sijainnista riippumatta elinkaaripalvelun avulla.

Asiakkaan kannalta olioiden luomiseen liittyvät tehdasoliot (engl. *factory object*), jotka luovat olioita. Tehdasoliot toteuttavat IDL-rajapinnan. Elinkaaripalvelu-standardi määrittelee rajapinnan tällaiselle geneeriselle tehtaalle. Asiakkaalla on siis oltava viite tehdasolioon, jotta se voi suorittaa elinkaarioperaatiota. Tehdasolioihin saadaan viitteitä samoin kuin mihin tahansa palvelinolioon (esimerkiksi nimipalvelun avulla).

Elinkaaripalvelu–standardissa määritellään myös `LifeCycleObject`-rajapinta, jossa määritellään olion poisto-, kopiointi- ja siirto-operaatiot. Poistaakseen olion, on asiakkaalla

oltava viite `LifeCycleObject`-rajapinnan toteuttavaan olioon. Olioiden siirtäminen ja kopioiminen tapahtuu samankaltaisesti kuin lisäys- ja poisto-operaatiot. Näissä operaatioissa tarvitaan lisäksi viite palveluun (*factory finder*), joka löytää tehdasolion halutusta paikasta. Tehdasolio luo ”siirretyn” tai kopioidun olion [OMG2].

2.6.5 Rinnakkaisuudenhallintapalvelu

Joissain järjestelmissä asiakkailla voi olla jaettuja resursseja, joihin pääsyä voidaan hallita rinnakkaisuudenhallintapalvelun (engl. *Concurrency Control Service*). Näin voidaan estää ristiriitaisuuksiin johtavien operaatioiden suorittaminen asiakkaiden välillä ja säilyttää jaettu resurssi eheänä. Jaetun resurssin käyttöä hallitaan lukoilla, joista kukin liittyy yhteen resurssiin ja yhteen asiakkaaseen. Mikäli jollain asiakkaalla on lukko tiettyyn resurssiin, ei toinen asiakas voi saada lukkoa samaan resurssiin, jos se voisi johtaa ristiriitaisuuksiin. Asiakkaan on huolehdittava lukkojen vapautuksista, mikäli rinnakkaisuudenhallintapalvelun kanssa ei käytetä transaktiopalvelua [ks. luku 2.6.8]. Tällöin transaktiopalvelu vastaa lukkojen vapauttamisesta. Rinnakkaisuudenhallintapalvelu tarjoaa monenlaisia lukkotyyppejä, jotka vastaavat erilaisia käyttöoikeuksia (esimerkiksi luku- ja kirjoitusoikeus) [OMG2].

2.6.6 Suhdepalvelu

Hajautettuja olioita mallinnetaan usein kokonaisuuksina. Hajautetut oliot ovat liittyneitä toisiin olioihin. Suhdepalvelun (engl. *Relationship Service*) avulla voidaan esittää kokonaisuuksia ja suhteita. Kokonaisuudet ovat CORBA-olioita. Suhdepalvelu määrittää kaksi uudentyyppistä olioa, jotka ovat **rooli** ja **suhteet**. Yksinkertaisena esimerkkinä rooleista ja suhteista voisi mainita tilanteen, jossa *yliopisto työllistää Kallen*. Tässä yliopisto ja Kalle ovat rooleja, joiden välillä on työllisyyssuhde. Vastaavasti olioilla voi olla erilaisia suhteita toisten olioiden kanssa. Rooli edustaa siis CORBA-olioa suhteessa. Suhteille voidaan määritellä myös sääntöjä.

Suhteille ja rooleille molemmille on määritelty rajapinnat, joita voidaan laajentaa attribuuteilla ja metodeilla. Suhdepalvelussa määritellään kolme palvelutasoa, jotka ovat perus- (engl. *base*), kaavio- (engl. *graph*) ja erityissuhdetaso (engl. *specific*). Perustasolla

ovat suhde- (engl. *Relation Interface*) ja roolirajapinnat (engl. *Role Interface*). Perussuhteet määritellään perustasolla. Kaaviotasolla muodostetaan kaavioita keskenään suhteessa olevista olioista. Kaaviotasolla on määritelty `traversal`-rajapinta, joka tarjoaa menetelmät kaaviossa kulkemiseen. Erityissuhdetasolla laajennetaan näkökulmaa suhteiden tarkasteluun määrittämällä erityyppisiä suhteita [OMG2].

2.6.7 Ulkoistamispalvelu

Ulkoistamispalvelussa (engl. *Externalization Service*) olion tila voidaan tallentaa tietovirran (engl. s*tream*) kautta esimerkiksi tiedostoon tai muistiin. Tämän jälkeen tilatieto voidaan sisäistää (engl. *internalize*) uuteen olioon ulkoisesta varastosta. Operaatio on samankaltainen olion kopioimisen kanssa. Ulkoistamispalvelulla onkin samoja piirteitä suhdepalvelun [ks. luku 2.6.6] ja elinkaaripalvelun [ks. luku 2.6.4] kanssa.

Asiakkaalla, joka haluaa ulkoistaa (engl. *externalize*) jonkin olion, on oltava viite `Stream`-rajapinnan toteuttavaan olioon, jolla on pääsy ulkoistettavaan olioon. Asiakas voi luoda `Stream`-olion myös `StreamFactory` –olion kautta. Kun viite on saatu, voidaan suorittaa ulkoistamisen käynnistävä palvelupyyntö. Ulkoistettavan olion on toteutettava `Streamable`-rajapinta, jossa on metodimääritykset ulkoistamiselle ja sisäistämiselle [OMG2].

2.6.8 Transaktiopalvelu

Transaktio (engl. *transaction*) on tapahtuma, jolla on seuraavat ominaisuudet:

- Transaktio on atominen. Ts. jos virhe keskeyttää sen, kaikki transaktion aiemman tapahtumat mitätöidään (paluu takaisin).
- Transaktio tuottaa yhdenmukaisia tuloksia.
- Transaktio on eristynyt muista transaktioista.
- Transaktio on kestävä, eli sen seuraukset ovat pysyviä.

Transaktio päättyy joko täytäntöön tai takaisinpaluuseen. Mikäli transaktio pannaan täytäntöön kaikki sen tekemät muutokset tehdään pysyviksi. Vastaavasti jos transaktio keskeytyy ja joudutaan palaamaan takaisin, peruutetaan transaktion siihen mennessä tekemät muutokset.

Transaktiopalvelu (engl. *Transaction Service*) määrittelee rajapinnat, joilla useat oliot voivat toteuttaa atomisuuden. Nämä rajapinnat määrittelevät operaatiot, joilla voidaan panna transaktio täytäntöön tai peruuttaa se. Transaktiopalvelu ei edellytä vaatimuksia muille kuin rajapinnan toteuttaville olioille [OMG2].

2.6.9 Kyselypalvelu

Kyselypalvelun (engl. *QueryService*) avulla voidaan suorittaa oliokyselyjä. Kyselyt perustuvat standardeihin kyselykieliin, kuten SQL-92, OQL-93 ja OQL-93 Basic. Kyselyihin saadaan vasteena oliojoukko, jonka oliot vastaavat kyselyn kriteerejä. Kyselypalvelun arkkitehtuurissa määritellään `Query Evaluator`-komponentti, joka suorittaa kyselyt olioille. Kyselypalvelun avulla voidaan muodostaa oliokokoelmia, joille kyselyt kohdennetaan. Kokoelmat (engl. *collection*) ovat olioita, joille voidaan lisätä jäseniä tai poistaa niitä. Kyselyillä voidaan hakujen lisäksi suorittaa olioiden lisäystä, päivitystä ja poistamista kokoelmista.

Kyselypalvelu määrittelee kaksi palvelutyyppiä, jotka ovat **kokoelmat** sekä **kyselykehykset** (engl. *Query Frameworks*). Kokoelmatyypissä tarjotaan rajapinnat oliokokoelmien hallintaan. Kokoelmalle on määritelty `Collection`-rajapinta. Kokoelmaa voidaan selata `Iterator`-rajapinnan metodeilla. Kyselykehyksen rajapinnat määrittelevät joustavan kehyksen oliokyselyiden käsittelyyn [OMG2].

2.6.10 Lisensointipalvelu

Ohjelmistojen tuottajat voivat toteuttaa lisensointipalvelun (engl. *Licensing Service*) omien tai asiakkaidensa tarpeiden mukaan. Lisensointipalvelulla voidaan hallita ohjelmiston käyttöä. Tätä varten palvelu määrittelee kolme lisenssityyppiä. Aikalisenssille voidaan

määrittää voimassaoloaika, ns. *Value Mapping* -tyyppi sallii tuottajien toteuttaa lisensointimalleja ja kuluttajalisenssi voidaan määrittää jollekin kokonaisuudelle (esim. työasemalle). Lisensointipalvelu koostuu kahdesta rajapinnasta, jotka ovat `LicenseServiceManager` ja `ProducerSpecificLicenceService`. Näiden lisäksi lisensointipalvelu erottaa lisensointijärjestelmän omaksi kokonaisuudekseen, jossa määritellään lisenssityypit [OMG2].

2.6.11 Ominaisuuspalvelu

Kaksi olioa on CORBAn kannalta samantyyppisiä, jos ne toteuttavat saman rajapinnan. Ominaisuudet ovat tyypitettyjä ja nimettyjä arvoja, jotka liittyvät olioon. Ominaisuuksien perusteella olioita voidaan esimerkiksi luokitella ja seurata niiden käyttöastetta.

Palvelu toteuttaa `PropertySet`- tai `PropertySetDef`-rajapinnan. `PropertySet`-perustaisessa palvelussa ominaisuus on nimi-arvo -pari (engl. *name-value*), jossa nimi on tyypiltään merkkijono ja arvon tyyppi voi olla mikä tahansa (eli sen tyypiksi on määritelty *any*). `PropertySetDef`-rajapintaa voidaan pitää `PropertySet`-rajapinnan aliluokkana. Siinä ominaisuuksille voidaan määritellä piirteitä (esim. luku- ja kirjoitusoikeus tai vain lukuoikeus). Ominaisuuspalvelun avulla voidaan `PropertySet`- tai `PropertySetDef`-olioita luoda tehdasolion avulla.

Asiakas voi lukea tai asettaa olioihin liittyviä ominaisuuksia. Kaikkien ominaisuuspalvelua tukevien olioiden on toteuttava joko `PropertySet`- tai `PropertyDefSet`-rajapinta. Palvelinolion kannalta `PropertySet` tarjoaa metodit ominaisuuksien määrittelemiseen, poistamiseen, nimeämiseen sekä niiden olemassaolon tarkastamiseen. Ominaisuudet ovat dynaaminen vastine CORBA-attribuuteille [OMG2].

2.6.12 Aikapalvelu

Aikapalvelu (engl. *Time Service*) koostuu kahdesta palvelusta. Se siis määrittelee myös kaksi rajapintaa. `TimeService` –rajapinnan toteuttava olio käsittelee UTO- (*Universal Time Object*) ja TIO-olioita (*Time Interval Object*). Tämän palvelun avulla saadaan kellonaika, johon liitetään arvio virheestä. Lisäksi kellonaikaa voidaan manipuloida.

`TimerEventService`-rajapinnan avulla voidaan käyttää ajastintoimintoja. Em. palvelut ovat riippuvaisia niiden alla sijaitsevasta järjestelmästä, joka tuo palveluille kellonajan. Tälle järjestelmälle on vaatimuksena, että se pystyy palauttamaan kellonajan ja siihen liittyvän virheparametrin [OMG2].

2.6.13 Turvallisuuspalvelu

CORBAssa käytetään nimitystä toimeksiantaja (engl. *principal*) palveluiden käyttäjistä. Toimeksiantaja voi olla joko asiakasolio tai asiakassovelluksen käyttäjä. Turvallisuuspalvelu (engl. *Security Service*) tarjoaa menetelmät toimeksiantajien tunnistamiseen ja autentikointiin. Lisäksi turvallisuuspalvelu tarjoaa menetelmät käyttöoikeuksien hallintaan, tapahtumien auditointiin, turvalliseen kommunikointiin, olioiden kiistämättömyyteen sekä turvallisuusinformaation hallintaan.

Turvallisuuspalvelun määrittelemisellä on suora vaikutus oliovälittimeen ja useimpiin oliopalveluihin sekä oliototeutuksiin. Itse turvallisuuspalvelun määritelmä tarjoaa ainoastaan kehyksen ja rajapinnat, jonka perusteella palvelu voidaan toteuttaa. Turvallisuuspalvelu nojautuu vahvasti olemassaoleviin turvallisuusmekanismeihin ja –standardeihin, kuten esimerkiksi SSL (*Secure Socket Layer*). Turvallisuusominaisuuksia voidaan käyttää järjestelmässä joko sovellukselle näkyvästi tai näkymättömästi [SA1].

CORBA-standardissa määritellään lisäksi SECIOP-protokolla (Secure Inter-ORB Protocol), jolla voidaan salata ORBien välinen kommunikaatio. SECIOP sijaitsee protokollapinossa GIOPin alla [OMG2].

2.6.14 Liikepalvelu

Liikepalvelun (engl. *Trading Object Service*) avulla palvelinoliot voivat mainostaa palvelujaan, kuten yritykset puhelinluettelon keltaisilla sivuilla. Tätä kutsutaan liikepalvelussa tuonniksi (engl. *import*). Liikepalvelun toteuttaa liikepalveluolio (engl. *Trader*). Liikepalveluolio voi palveluiden tarjoamisen ja mainostamisen lisäksi yrittää löytää asiakkaan tarpeita vastaavia palveluita. Tätä palvelua kutsutaan vastaavasti vienniksi (engl. *export*).

Tuontia varten palvelinolion on annettava liikepalveluoliolle kuvaus palveluistaan sekä palvelut toteuttavan rajapinnan sijainti. Viennissä liikepalveluoliolta kysytään palvelua, jolla on jotain tiettyjä ominaisuuksia. Liiketoimintaolio vertaa kysyttyjä ominaisuuksia tuotujen palveluiden kuvauksiin ja palauttaa valitun palvelurajapinnan sijainnin.

Koska palveluita voi olla isommissa järjestelmissä huomattava määrä, on liikepalvelun kuorma syytä jakaa useammalle palvelinoliolle. Liiketoimintaolioiden on siis pystyttävä kommunikoimaan myös toisten liiketoimintaolioiden kanssa.

Liiketoimintaolion toteutus voi perustua esimerkiksi tietokantaan tai muistinkäyttöön. Molemmissa vaihtoehdoissa on hyvät ja huonot puolensa, joten järjestelmän arkkitehtuuri on ratkaiseva tekijä liikepalvelutoteutusta valittaessa. Liikepalvelu-standardi määrittelee abstrakteja rajapintoja liiketoimintaolioiden luomiseksi sekä toiminnallisia rajapintoja [OMG2].

2.6.15 Oliokokoelmapalvelu

Kokoelmat ovat oliojoukkoja, jotka puolestaan tukevat joitain yhteisiä operaatiota. Lisäksi oliojoukot esittelevät ominaisuuksia, jotka liittyvät joukon luonteeseen. Kokoelma voi olla esimerkiksi jono, pino, lista tai puu. Oliokokoelmapalvelun (engl. *Object Collections Service*) tarkoituksena on toteuttaa yleinen ja geneerinen tapa luoda ja manipuloida kokoelmia. Kokoelmien avulla voi siis suorittaa operaatioita kokoelman olioille samanaikaisesti tai vain yhdelle kokoelman oliolle.

Oliokokoelmapalvelun määrittelemät rajapinnat jaetaan kolmeen luokkaan. Ensimmäiseen luokkaan kuuluvat kokoelmaolioiden ja kokoelmatehdasolioiden toteuttamat rajapinnat. Toisessa luokassa on rajapinnat kokoelmien läpikäymiseen ja kolmannessa luokassa ovat toiminnalliset rajapinnat, joilla voidaan välittää käyttäjän määrittelemää informaatiota kokoelman toteutukselle [OMG2].

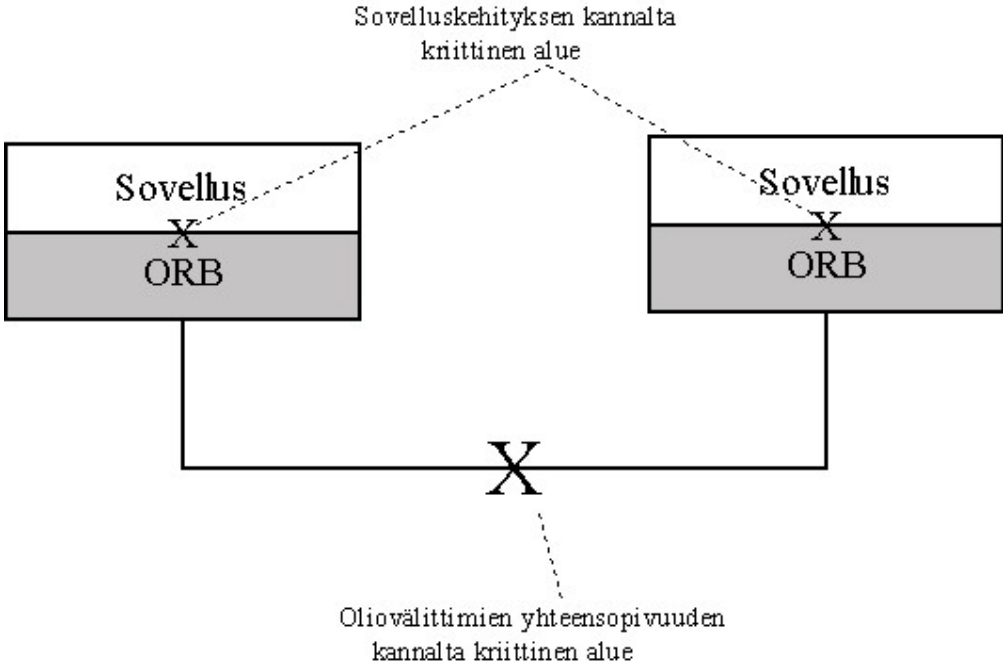
2.7 CORBA rajoitetuissa ympäristöissä – minimumCORBA

Luvussa 2.5 esitetyt palvelut tekevät CORBAsta laajan standardin. Tästä johtuen toteutukset voivat viedä resursseja suoritusympäristöltä. CORBAn yleistyessä sulautetuissa järjestelmissä ORB-toteutuksista ja -palveluista on osa jouduttu riisumaan pois, jotta laitteiden koko ja hinta voitaisiin pitää kohtuullisina. Tämä on johtanut *minimumCORBA*-standardin määrittelyyn, joka on suunniteltu erityisesti resursseiltaan rajoitettujen järjestelmien käyttöön. Tällaisia voivat olla esimerkiksi kämmentietokoneet, puhelimet sekä mittarit.

MinimumCORBA sisältää täyden IDL-tuen. Lisäksi minimumCORBA-sovellusten tulee olla yhteensopivia CORBA-sovellusten kanssa. Poisjätettyjä CORBAn ominaisuuksia voidaan tuoda minimumCORBA järjestelmiin ulkoisina resursseina, jos sovellus niitä tarvitsee. CORBA tarjoaa siis käyttäjäystävällisemmän kehitysympäristön kuin minimumCORBA, mutta vaatii myös huomattavasti enemmän resursseja.

ORB-toteutuksesta riippumatta myös CORBAn ja olio-ohjelmoinnin perusominaisuudet lisäävät sovelluksen aiheuttamaa kuormitusta. Näitä ovat esimerkiksi any-tyypin käyttäminen, omien poikkeusten määritteleminen ja IDL-rajapintojen moniperinnän käyttö. Vaikka minimumCORBA-standardissa edellä mainittuja ominaisuuksia ei olekaan jätetty pois, voidaan ne kuitenkin jättää pois yhteensopivasta minimumCORBA-toteutuksesta. Sen sijaan dynaamiset palvelupyynnöt (dynaaminen kutsurajapinta ja dynaamiset rungot) on jätetty minimumCORBAsta pois. Lisäksi rajapintavarastosta on suuri osa jätetty pois. POA-rajapinnasta tuetaan vain osaa sen määrittelemiä palvelupyyntöjä [OMG1].

MinimumCORBAssa ORBille asetetaan samat yhteensopivuusvaatimukset kuin CORBAssakin. Kuvassa 5 on esitetty yhteensopivuuden kannalta kriittiset alueet CORBA-arkkitehtuurissa. Sovelluskehityksen kannalta kriittiset alueet käsittävät sovellusohjelmointirajapinnat eli APIt (*Application Programming Interface*), kun taas ORBien välisen kommunikaation kriittiset alueet käsittävät kommunikointiprotokollan.



Kuva 5. Oliovälittimien yhteensopivuus.

2.8 CORBA reaaliaikaisessa arkkitehtuurissa – Real-Time CORBA

Arkkitehtuuristandardeja määriteltäessä joudutaan usein tilanteisiin, jossa yleisen ratkaisun ja erikoisjärjestelmien vaatimukset muodostavat ristiriitoja. Real-Time CORBA laajentaa CORBA-standardin vastaamaan reaaliaikajärjestelmien asettamia vaatimuksia. Reaaliaikajärjestelmien tuottajien on kiinnitettävä erityistä huomiota resurssien varaamiseen ja vapauttamiseen sekä järjestelmän luotettavaan toimintaan. Real-Time CORBA tarjoaa järjestelmien kehittäjille mahdollisuudet resurssienkäytön ja luotettavuuden hallintaan joidenkin CORBAn perusominaisuuksien kustannuksella.

Reaaliaikajärjestelmät jaetaan reaaliaikaisuusvaatimusten perusteella kahteen luokkaan, koviin (engl. *hard*) ja pehmeisiin (engl. *soft*) reaaliaikajärjestelmiin. Real-Time CORBA tarjoaa tuen molempien järjestelmien kehittämiseen. Real-Time CORBAn tavoitteena on siis tukea reaaliaikajärjestelmien kehittäjiä toteuttamaan päästä-päähän luotettavuutta ja tarjoamalla tuen resurssienkäytön hallintaan. Se tuo reaaliaikajärjestelmien kehitykseen

samat edut joustavuudessa, siirrettävyydessä sekä yhteensopivuudessa, kuin mitä CORBA toi asiakas-palvelin –sovelluksiin.

Reaaliaikajärjestelmän on toimittava odotuksenmukaisesti ja ennustettavasti, jotta järjestelmässä voidaan asettaa suoritukselle prioriteetteja. Ennustettavaa toimintaa edistää järjestelmän komponenttien deterministinen käyttäytyminen. Real-Time CORBAn rajapinnat ja mekanismit helpottavat ORBin ja sovelluksen toiminnan ennustamista. Sovellus käyttää resurssienhallintaan Real-Time CORBAn rajapintoja ja ORBin mekanismit koordinoivat sovelluksen muodostavia toimintoja. Real-Time ORB on riippuvainen alla sijaitsevasta reaaliaikakäyttöjärjestelmästä eli RTOS:ista (*Real-Time Operating System*).

Reaaliaikajärjestelmissä on usein pystyttävä laukaisemaan toimintoja (engl. *activites*) muutenkin kuin CORBAn pyyntö- ja vastausviesteillä. Toiminnan laukaisijana voi olla esimerkiksi jokin I/O-operaatio (*Input/ Output*). Real-Time CORBAssa toimintoja ei kuvata IDL:llä, vaan siinä käsitellään IDL:llä määriteltyjen operaatioiden kutsumista (engl. *invocation*). Toiminnan ja operaation ero on, että toiminta voi sisältää useita (mahdollisesti sisäisiä) operaatioita.

Real-Time CORBA -järjestelmän neljä pääkomponenttia ovat Real-Time ORB, käyttöjärjestelmän skedulointi (engl. scheduling), viestien siirto sekä sovellukset. Kunkin toteutuksella voidaan vaikuttaa järjestelmän päästä-päähän ennustettavuuteen.

Real-Time CORBAssa ei määritellä erillistä RT-IOP:ia, sillä Real-Time CORBAa voidaan soveltaa useissa, luonteeltaan erilaisissa, järjestelmissä. Lisäksi järjestelmissä voidaan käyttää eri kuljetuskerroksen protokollia. RT-IOP:in määrittelyn sijaan käytetään hyväksi IIOP:n laajennusmekanismeja, joita ovat palvelukontekstit (*GIOP ServiceContexts*), IOR-profiilit (*IOR Profiles*) sekä IOR-tagikomponentit (*IOR Tagged Components*). Näiden avulla voidaan rakentaa tuki reaaliaikajärjestelmän protokollia varten IIOP:hen. Näin saavutetaan myös kahden Real-Time CORBA toteutuksen välinen yhteensopivuus. Tosin yhteensopivuus ei ole reaaliaikajärjestelmissä yhtä tärkeä vaatimus kuin perinteisissä hajautetuissa järjestelmissä. Rajapintamäärittelyt (etenkin IIOP:n laajennus) mahdollistavat sen, että Real-Time CORBAn komponentit voivat kommunikoida CORBA-komponenttien

kanssa. Useat ohjelmistovalmistajat ovat toteuttamassa Real-Time CORBAan perustuvia ratkaisuja tietoliikenteen alalla uusimpiin optisiin kytkimiin. Näiden käyttöönotto on huomattavasti helpompaa operaattoreille, joilla verkonhallintajärjestelmä perustuu CORBAan.

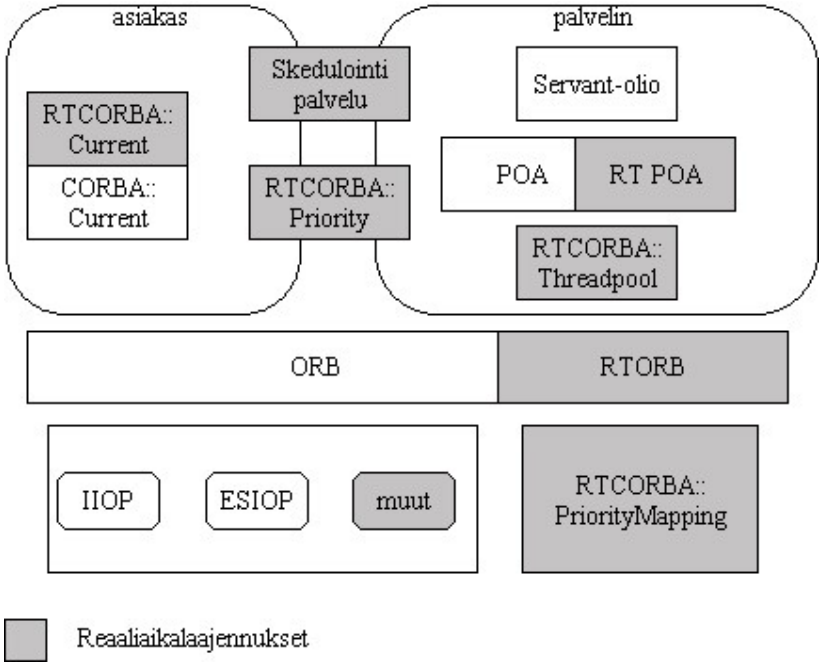
2.8.1 Real-Time CORBAn arkkitehtuuri

Kuvassa 6 on esitetty Real-Time CORBAn arkkitehtuuri. Kaikki Real-Time CORBA IDL-määritykset ovat `RTCORBA`- ja `RTPortableServer`-moduuleissa.

Real-Time CORBA jakaa suoritusaikaa säikeille. Tätä kutsutaan skeduloinniksi. Toimintojen ajastusta hallitaan koordinoimalla toiminnon muodostavia säikeitä. Real-Time CORBA tarjoaa säikeiden hallintaan rajapinnat (`Threadpool` ja `Real-Time CORBA Current`).

Real-Time CORBA määrittelee prioriteettimallin (*Real-Time CORBA Priority*), joka piilottaa Real-Time CORBAn alla olevien käyttöjärjestelmien prioriteettimallit. Tätä varten tarvitaan rajapinta, joka suorittaa prioriteettimallien muunnokset (engl. *mapping*). Tämä rajapinta on `PriorityMapping`. Prioriteettimallin avulla CORBA-sovellukset voivat suorittaa priorisoituja CORBA-kutsuja.

Skedulointipalvelulla (engl. *Scheduling Service*) voidaan piilottaa Real-Time CORBAn skedulointiparametrien (esimerkiksi CORBA-prioriteetit ja Real-Time POA menettelyt) koordinointi [OMG1].



Kuva 6. Real-Time CORBA.

2.9 Vikasietoisuus ja CORBA

Useissa hajautetuissa sovelluksissa on vaatimuksena vikasietoisuus. Tällaisia järjestelmiä voivat olla esimerkiksi lennonjohtosovellukset, hätäkeskukset, sulautetut järjestelmät sekä kommunikaatiojärjestelmät. Standardi, joka sopisi kaikkiin sovelluksiin, täyttäisi todennäköisesti järjestelmien tarpeet huonosti. Tämän vuoksi CORBAn vikasietoisuusmääritelmä (*Fault Tolerant CORBA*) sisältää lukuisia kompromissejä.

CORBAn vikasietoisuusstandardi pyrkii tarjoamaan vahvan tuen sovelluksille, jotka edellyttävät korkeaa toimintavarmuutta. Vikasietoisuus riippuu järjestelmän entiteettien (engl. *entity*) redundanssista, virheiden havaitsemisesta sekä virheistä toipumisesta. Entiteetin redundanssilla tarkoitetaan käytännössä palvelinolioiden replikoimista eli monistamista. Vikasietoisuusstandardi tarjoaa tuen myös sovelluksille, jotka edellyttävät vikasietoisuutta oliokopioiden eli replikoiden hallintaan. Standardi sisältää replikoinnin lisäksi myös muita vikasietoisuusmenetelmiä, kuten virheiden havaitseminen, virheistä

ilmoittaminen, palvelupyyntöjen uudelleenlähetys ja pyyntöjen välittäminen toiselle palvelimelle.

Hajautettuun vikasietoiseen järjestelmään kuuluu perusarkkitehtuurin lisäksi virheiden havaitsemiseen, niistä ilmoittamiseen ja toipumiseen tarvittavat mekanismit. Lisäksi tarvitaan kokonaisuus, joka vastaa replikoinnista ja replikoiden hallinnasta.

3 CORBA langattomassa ympäristössä

Tässä luvussa käsitellään CORBAn liittämistä langattomaan ympäristöön ja sen aiheuttamia vaatimuksia. Langattomassa ympäristössä voidaan suorittaa niin asiakasta kuin palvelintakin langattomalla päätelaitteella. Olennaisimpana asiana siirrettäessä CORBA-sovelluksia langattomaan ympäristöön on protokollatasolla tapahtuvat muutokset, jonka tarkasteluun on kiinnitetty erityistä huomiota tässä luvussa. Luku perustuu pääasiassa lähteisiin [OMG1], [OMG3] ja [S1].

3.1 Taustaa

Tietoverkkojen kehittyminen mahdollisti hajautettujen sovellusten kehittämisen. CORBA tarjosi avoimen ja alustariippumattoman teknologian hajautettujen järjestelmien kehittämiseen, jossa olioperustaisilla tekniikoilla peitetään hajautuksen mukana tullut kompleksisuus. Langattomien verkkoteknologioiden, kuten GSM, WLAN, GPRS ja UMTS, myötä on syntynyt luonnollisesti tarve siirtää hajautettuja järjestelmiä (tai niiden komponentteja) langattomaan ympäristöön [OMG4]. Koska mobiililaitteilla käytössä olevat prosessointi- ja muistiresurssit ovat yleensä huomattavasti työasemia rajoitetumpia, tulee etäolioiden palveluiden merkitys tärkeämmäksi. Mobiililaitteiden yleistyessä käyttäjät odottavat voivansa käyttää samoja palveluja kuin työasemiltaan.

CORBA-standardi sisältää oletuksia, jotka eivät päde langattomassa ympäristössä (lähinnä luotettava kuljetusprotokolla). Tästä johtuen CORBAa ei pidetty järin sopivana ratkaisuna sellaisenaan langattomaan ympäristöön. Vuonna 1999 OMG julkisti RFP:n (*Request For Proposal*) ”Wireless Access and Terminal Mobility”, jossa pyydettiin selvittämään teknologioita, joilla mobiililaitteiden sovellukset voivat käyttää tai tarjota CORBA-palveluita. Mobiililaitteiden tuli siis pystyä toimimaan asiakkaana tai palvelimena. RFP:ssä haettiin ratkaisuja seuraaviin asioihin:

- OMA-yhteensopiva (*Object Management Architecture*) arkkitehtuurikehys mobiililaitteiden liittymiseksi mobiilitoimialueeseen (engl. *mobility domain*),
- GIOP-viestien siirtäminen langattomilla siirtoprotokollilla,
- mekanismit, joilla CORBA-palvelimien liikkuvuus (engl. *mobility*) piilotetaan CORBA-asiakkailta,
- mekanismit, joilla tarvittavat CORBA-palvelut voidaan paikallistaa mobiilitoimialueessa,
- mekanismit, CORBA-palveluiden tarjoamiseen mobiilitoimialueella sekä
- mobiilitoimialueiden välinen liikenne.

Mobiilitoimialueella tarkoitetaan tässä toimialuetta, johon mobiilipäätelaitteet voivat liittyä ja jossa ne voivat käyttää tai tarjota palveluita. Päätelaitteen liikkuvuus mobiliteettitoimialueen sisällä on ORBille näkymätöntä [OMG4].

RFP:n vaatimukset osoittautuivat varsin kattaviksi langattomiin ympäristöihin, joka vaikutti vastausten lähestymistapaan. Langaton CORBA -spesifikaatio ”Wireless Access and Terminal Mobility in CORBA” (OMG document dtc/01-06-02) on hyväksytty spesifikaatio ja se oli joulukuussa 2001 vielä viimeisteltävänä. Versio 1.0 ilmestyy vuonna 2002 [OMG3]. Suurin osa spesifikaation liittyvistä toteutuksista on tehty Helsingin yliopistossa ja niitä käytetään tässä tutkielmassa koejärjestelmän toteutuksessa. Tarkempaa tietoa spesifikaatiosta ja siihen liittyvistä IDL-rajapinnoista on lähteessä [OMG3].

3.1.1 OMG:n spesifiointiprosessi

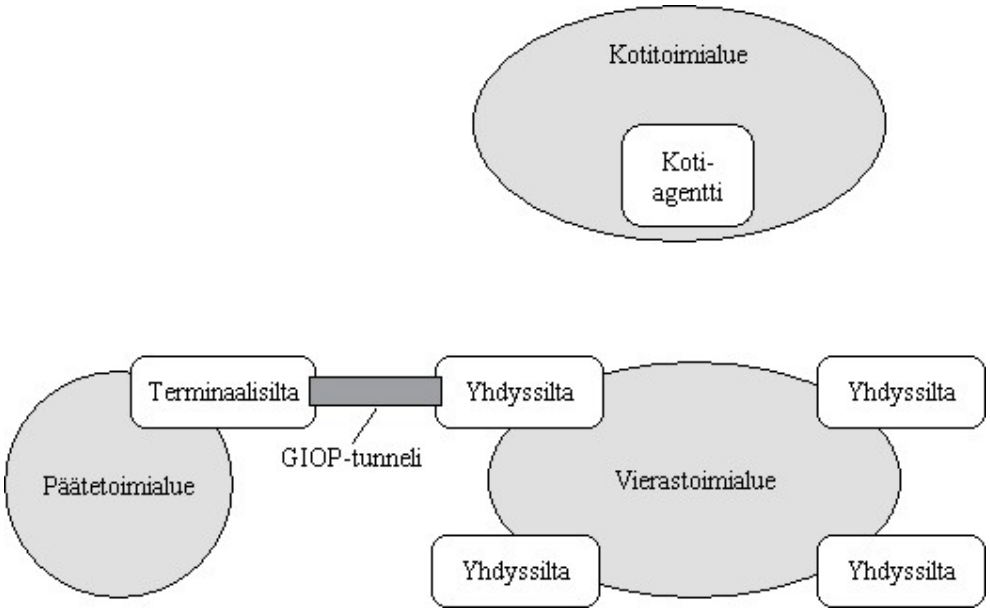
OMG hyväksyy spesifikaatioita äänestyksellä. OMG perustaa päätöksensä liiketoiminnallisiin ja teknologisiin näkökulmiin. Kun spesifikaatio on hyväksytty, on se julkisesti saatavilla.

OMG:n julkistaman RFP:n vastauksille annetaan päivämäärä, johon mennessä alkuehdotukset (engl. *Initial Submission*) on oltava valmiina. Alkuehdotuksien odotetaan olevan täydellisiä ratkaisuehdotuksia RFP:hen. Lisäksi niiden mukaisten toimivien toteutusten tulee valmistua ehdotusvaiheen aikana.

Ehdotusten jättämisen jälkeen ne arvioidaan. Tällä välin ehdotuksia tuottaneilla yhtiöillä on mahdollisuus tarkastaa ehdotuksiaan sekä mahdollisesti yhdistää ehdotuksia. Korjatuille ehdotuksille (engl. *Revised Submission*) asetetaan päivämäärä, jolloin niiden on oltava valmiina. Kun rekisteröityneet äänestäjät kokevat ymmärtävänsä korjatun ehdotuksen sisällön, järjestetään spesifikaatioäänestys. Tämän jälkeen spesifikaatio viimeistellään, jonka jälkeen se julkistetaan versiolla 1.0 [OMG4].

3.2 Arkkitehtuuri

”Wireless Access and Terminal Mobility in CORBA” -spesifikaatio määrittelee arkkitehtuurin ja rajapinnat, joilla CORBA-menetelmää voidaan käyttää langattomassa ympäristössä. Spesifikaatiossa ei edellytetä muutoksia kiinteissä verkoissa toimiviin ORB-toteutuksiin, jotta ne voisivat kommunikoida langattomien asiakkaiden tai palvelinsovellusten kanssa. Kiinteille ORBeille (engl. *non-mobile ORB*) mobiili-ORBien liikkuminen ei siis näy mitenkään. Langattoman CORBA-ympäristön perusarkkitehtuuri on esitetty kuvassa 7 [OMG3].



Kuva 7. Langaton perusarkkitehtuuri.

Langattoman CORBA-arkkitehtuurin peruskonseptit ovat

- Mobiili-IOR (engl. *Mobile IOR*),
- Kotiagentti (engl. *Home Location Agent*),
- Yhdyssilta (*Access Bridge*),
- Terminaalisilta (engl. *Terminal Bridge*) sekä
- GIOP-tunneli.

3.2.1 Toimialueet

Kuvassa 7 on kolme toimialuetta, jotka ovat kotitoimialue (engl. *Home Domain*), vierastoimialue (engl. *visited domain*) ja päätetoimialue (engl. *terminal domain*). Kotitoimialueessa sijaitsee päätelaitteen kotiagentti. Vierastoimialueessa on yksi tai useampi yhdyssilta, joiden kautta voidaan kommunikoida liikkuvien päätelaitteiden (engl. *mobile terminal*) kanssa. Päätetoimialueessa on päätelaite, jossa on oliovälitin ja terminaalisilta. Tämän avulla oliot voivat kommunikoida toisessa verkossa sijaitsevien olioiden kanssa.

3.2.2 Mobiili-IOR

Langattomalla päätelaitteella sijaitsevalla palvelimella on erityinen objektireferenssi, Mobiili-IOR. Se piilottaa päätelaitteen liikkumisen asiakkailta, jotka käyttävät langattomalla päätelaitteella olevia palvelinolioita. Päätelaitteen liikkuminen ei näy myöskään asiakkaiden oliovälittimille. Mikäli kutsut tapahtuvat ainoastaan langattomasta verkosta kiinteään verkkoon (eli palvelin sijaitsee kiinteässä verkossa ja asiakkaat langattomalla päätteellä), ei mobiili-IORilla ole käytännössä merkitystä.

Mobiili-IOR sisältää IIOP-profiilin lisäksi mobiiliterminaaliprofiilin. Yhdessä mobiili-IORissa voi olla useampi IIOP-profiili, mutta vain yksi mobiiliterminaaliprofiili. Asiakkaan oliovälitin käyttää mobiili-IORin (yhtä) IIOP-profiilia asiakkaan pyyntöjen välittämiseen palvelinolion päätelaitteen yhdyssillalle.

Mobiili-IORin IIOP-profiilin isäntä- ja portti-informaatio eivät ilmaise palvelinolion osoitetta, vaan palvelinolion päätelaitteen kotiagentin tai päätelaitteeseen liittyvän yhdyssillan osoitetta. Kun mobiili-IOR luodaan päätelaitteessa, päättää päätelaitteen oliovälitin viitataanko IIOP-profiilin osoitteella kotiagenttiin tai yhdyssiltaan. Jos osoiteinformaatiolla viitataan kotiagenttiin, välittää kotiagentti yhdyssillan viitteen palvelupyynnön esittäjälle. Tämä on viite yhdyssiltaan, johon kotiagentti uskoo päätelaitteen olevan liittynyt. Vastaavasti jos osoiteinformaatiolla viitataan yhdyssiltaan, eikä päätelaite ole enää liittynyt kyseiseen siltaan, välitetään pyyntö päätelaitteen uudelle yhdyssillalle. Näihin ilmoituksiin käytetään GIOPin `Reply`-viestillä (`LOCATION_FORWARD`–tilatiedolla) [ks. luku 2.5.2].

Mobiili-IORin mobiiliterminaaliprofiilissa on informaatiota kotiagentista ja yhdyssillasta. Tämä informaatio piilottaa päätelaitteiden liikkuvuuden. Mobiiliterminaaliprofiilia tarvitaan ORBeissa, joiden avulla toteutetaan kotiagentit ja yhdyssillat. Sitä ei siis tarvita oliovälittimissä, joita ei ole tarkoitettu langattomaan ympäristöön [OMG3].

3.2.3 Kotiagentti

Kotiagentissa ylläpidetään tietoa yhdyssillasta, johon päätelaite on liittynyt. Näin tiedetään mitä yhdyssillan olioita voidaan kutsua. Kotiagentti tarjoaa operaatiot päätelaitteen sijainnin selvittämiseen ja päivittämiseen. Lisäksi siltä voidaan kysellä kotitoimialueeseen (engl. *home domain*) liittyviä palveluja ja viitteitä.

Kuhunkin päätelaitteeseen liittyy yksikäsitteinen päätelaitetunnistin (engl. *terminal id*). Kun yhdyssilta kadottaa tai saa yhteyden päätelaitteeseen, lähettää se kotiagentille päivitysviestin. Viestiin liitetään päätelaitetunnistin ja sijainti (jos yhteys kadotetaan, sijaintia ei tunneta).

Kun kotiagentti vastaanottaa GIOP-viestin, se tutkii onko viestin kohdeolio liittynyt yhdyssiltaan. Mikäli liitos löydetään vastataan `Reply`-viestillä (joka sisältää `LOCATION_FORWARD`-tilatiedon) ja palautetaan yhdyssillan mobiili-IOR. Jos päätelaitteeseen liittyvää yhdyssiltaa ei löydetä, palautetaan poikkeus (`OBJECT_NOT_EXIST`) [OMG3].

3.2.4 Yhdyssilta

Yhdyssilta on GIOP-tunnelin kiinteän verkon pää. Se kapseloi (engl. *encapsulate*) GIOP-viestit terminaalisillalle ja purkaa terminaalisillalta saatujen viestien kapseloinnin. Yhdyssillan kautta voidaan selvittää vierastoimialueen palvelut ja viitteet palveluihin. Yhdyssilta voi tukea myös kanavanvaihtoa (engl. *handoff*) ja päätelaitteiden liikkuvuuteen liittyviä viestejä.

Yhdyssillalta voidaan kysyä yhdyssillan osoiteinformaatiota tai onko jokin päätelaite liittynyt yhdyssiltaan. Yhdyssilta toimii siis linkkinä asiakkaan ja palvelinolion välillä. Se ylläpitää sidoksia päätelaitteen tunnisteen (*terminal_id*) ja päätelaitteeseen liittyvän GIOP-tunnelin osoitteen välillä. Lisäksi ylläpidetään tilatietoa GIOP-viesteistä, joihin odotetaan vastetta. Kun yhdyssilta vastaanottaa päätelaitteelle tarkoitetun viestin, se kapseloi viestin GTP:n sisään ja lähettää sen päätelaitetunnisteeseen liitettyyn GIOP-tunneliin. Jos yhdyssillalla ei ole tunneliassosiaatiota päätelaitteeseen, voidaan päätelaitteen sijainti kysyä kotiagentilta tai korvata IORin profiilitieto kotiagentin tiedoilla. Jos IORissa ei ole kotiagentti tietoa ja yhdyssilta ei tunne päätelaitteen kotiagenttia, ei yhdyssilta voi paikallistaa haettua olioa.

Mikäli kohdeolion päätelaite on vaihtanut yhdyssiltaa, voidaan käyttää luvussa 3.3.3 esitettyä välitysmekanismia. Näitä välitysmekanismeja ei kuitenkaan välttämättä tueta yhdyssillassa. Yhdyssillat voivat tukea myös erilaisia tapahtumia (engl. *event*). Päätelaitteen siirtyessä toiseen yhdyssiltaan, voidaan tästä ilmoittaa tapahtumilla. Vastaavasti voidaan tapahtumilla ilmoittaa myös päätelaitteen liittymisestä yhdyssiltaan.

Yhdyssilloille on määritelty seuraavat tapahtumat [OMG3]:

- `HandoffArrivalEvent` muodostettaessa GIOP-tunneli päätelaitteeseen,
- `HandoffDepartureEvent` purettaessa GIOP-tunneli,
- `AccessDropoutEvent` havaittaessa yhteyskatko päätelaitteeseen sekä
- `AccessRecoveryEvent` GIOP-tunnelin uudelleenmuodostuksen jälkeen.

3.2.5 Terminaalisilta

Terminaalisilta on GIOP-tunnelin päätelaitteen pää. Se kapseloi yhdyssillalle lähetettävät viestit ja purkaa yhdyssillalta saadut viestit. Terminaalisilta voi lisäksi tarjota myös tapahtumakanavan (engl. *moblility event channel*), jossa voidaan kuljettaa yhteyteen liittyviä ilmoituksia [OMG3]. Tapahtumat ovat samankaltaisia kuin yhdyssillalla. Terminaalisillan laukaisemat tapahtumat ovat

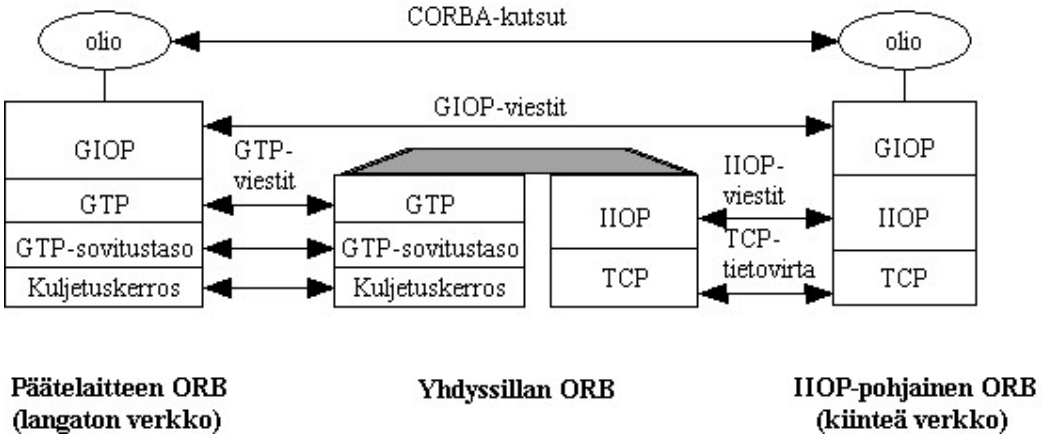
- `TerminalDropoutEvent` menetettäessä yhteys yhdyssiltaan,
- `TerminalRecoveryEvent` yhteyden uudelleenmuodostuksessa sekä
- `TerminalHandoffEvent` yhteyden muodostuksessa uuteen yhdyssiltaan.

3.2.6 GIOP-tunnelointi

GIOP-tunnelissa kuljetetaan GIOP-viestejä yhdyssillan ja terminaalisillan välillä. Yhdyssillan ja terminaalisillan välillä voi olla ainoastaan yksi tunneli, mutta päätelaite voi vaihtaa näkymättömästi tunnelin toiseen yhdyssiltaan muodostettuun tunneliin. Tunneli jaetaan kaikkien päätelaitetta koskevien GIOP-viestien kesken. Tunnelin on oltava siis multipleksattavissa. Multipleksauksella tarkoitetaan usean loogisen yhteyden sovittamista samalle fyysiselle yhteydelle.

3.3 Kommunikointi

GIOP-tunnelointiprotokolla eli GTP määrittelee GIOP-viestien siirron tunnelissa. GTP määrittelee myös tunnelin muodostuksen, purkamisen ja uudelleenmuodostuksen. Lisäksi se määrittelee GIOP-yhteyksien muodostamisen ja purkamisen yhdyssiltojen kautta. GTP on kuljetuskerroksesta riippumaton abstrakti protokolla, jolle 01-06-02 –spesifikaatio määrittelee toteutukset TCP-, UDP- ja WAP:n WDP-kuljetusprotokollien päälle. GIOP-tunnelointi on esitetty kuvassa 8.



Kuva 8. GIOP-tunnelointi.

GTP on suunniteltu siten, että konkreettisen tunnelointiprotokollan määrittely on yksinkertaista. Konkreettinen tunnelointiprotokolla asettuu kuljetuskerroksen ja GTP:n väliin ikään kuin sovituskerrokseksi.

GIOP-tunnelin muodostus koostuu kahdesta vaiheesta: kuljetuksen alku- ja loppupään tunnistuksesta sekä GIOP-tunnelin muodostamisesta näiden välille. GTP olettaa, että sovituskerros tarjoaa saman luotettavuuden kuin minkä GIOP olettaa viestin kuljetukselta. Mikäli käytettävä kuljetusprotokolla ei tarjoaa luotettavaa kuljetusta, on sovituskerroksen tehtävä tämä.

3.3.1 GTP-viestin rakenne

GTP-viestin kehysrakenne on esitetty kuvassa 9. GTP määrittelee erilaisia viestityyppejä, joihin liittyvä informaatio sisältyy kuvan viestirunkoon (engl. *message body*). Viestityypit käsitellään myöhemmin tässä luvussa.

seq_no	last_seq_no_received	gtp_msg_type	flags	content_length	message body

Kuva 9. GTP-viestin rakenne.

GTP-viesti alkaa kahdeksan oktetin mittaisella otsikolla. Otsikossa on kentät viestin numeroille, joista `seq_no`-kentässä ilmoitetaan viestin järjestysnumero. Vastaanottaja

ilmoittaa `last_seq_no_received`-kentässä viimeksi vastaanotetun viestin numeron. GTP-viestin tyyppi ilmoitetaan `gtp_msg_type`-kentässä [ks. taulukko 2]. Otsikon vasemman puoleisimmalla lippubitillä ilmoitetaan GTP-otsikossa ja GTP-hallintaviestissä (engl. *GTP Control Message*) käytetty *Endian*-tyyppi (*Big-Endian* tai *Little-Endian*). Loput seitsemän bittiä on varattu tulevaisuuden käyttöön. GTP-viestin pituus ilmoitetaan *content_length*-kentässä.

3.3.2 GTP:n viestityypit

GTP määrittelee 17 erilaista viestityyppiä eri tarkoituksiin. Suurin osa viestityypeistä koskee yhteyden hallintaa. GTP-viestit on kuvattu taulukossa 2.

Viesti	**Tyyppiarvo**	**GTP versio**
`IdleSync`	0x00	1.0, 2.0
`EstablishTunnelRequest`	0x01	1.0, 2.0
`EstablishTunnelReply`	0x02	1.0, 2.0
`ReleaseTunnelRequest`	0x03	1.0, 2.0
`ReleaseTunnelReply`	0x04	1.0, 2.0
`HandoffTunnelRequest`	0x05	2.0
`HandoffTunnelReplyCompleted`	0x06	2.0
`OpenConnectionRequest`	0x07	1.0, 2.0
`OpenConnectionReply`	0x08	1.0, 2.0
`CloseConnectionRequest`	0x09	1.0, 2.0
`CloseConnectionReply`	0x0A	1.0, 2.0
`ConnectionCloseIndication`	0x0B	1.0, 2.0
`GIOPData`	0x0C	1.0, 2.0
`GIOPDataReply`	0x0D	1.0, 2.0
`GTPForward`	0x0E	2.0
`GTPForwardReply`	0x0F	2.0
`Error`	0xFF	1.0, 2.0

Taulukko 2. GTP-viestit.

`IdleSync`-viestillä ei ole varsinaista viestirunkoa (engl. *message body*). Viesti lähetetään terminaalisillasta tai yhdyssillasta. Viestiä käytetään yhteydenhallintaan. `EstablishTunnelRequest`-viestillä terminaalisilta lähettää tunnelin muodostus- tai uudelleenmuodostuspyynnön yhdyssillalle. Viestiin on liitettävä päätelaitetunnistin sekä viite kotiagenttiin. Lisäksi viestissä viedään `time_to_live_request`-parametri. Jos `EstablishTunnelRequest`-viestillä pyydetään tunnelin uudelleenmuodostusta, liitetään viestiin vielä informaatio viimeisestä yhdyssillasta sekä siltä vastaanotetusta viimeisestä viestistä.

Vastauksena yhdyssilta lähettää `EstablishTunnelReply`-viestin, jossa ilmoitetaan tunnelinmuodostuksen onnistumisesta. Tunnelinmuodostuksen onnistuessa viestissä kerrotaan, onko tunneli uusi vai uudelleenmuodostettu sekä onko päätelaitteella kotiagenttia (ns. *kodittomien päätelaitteiden* tapauksessa). Jos tunnelinmuodostus epäonnistuu, ilmoitetaan epäonnistumisen syy. Näitä voivat olla kotiagentin puuttuminen (jos silta ei tue kodittomia päätelaitteita) tai jokin geneerinen virhetilanne, kuten resurssien loppuminen.

Jos pyyntö koskee tunnelin uudelleenmuodostusta, tarkastetaan viimeisen onnistuneen lähetyksen numero (`last_seqno_received`) ja lähetetään tämän jälkeen lähetetyt viestit uudelleen. Tämä tarkastus suoritetaan luonnollisesti tunnelin molemmissa päissä ja sitä varten on tiedettävä viimeisen vastaanotetun viestin `last_seqno_received`-kenttä. Mikäli tunneli muodostetaan ensimmäistä kertaa sisääntulonsillan kanssa, on terminaalisillan lähetettävä tähän tunneliin mahdollisesti kadonneet viestit. Tunnelinmuodostuspyynnössä käytetään `time_to_live_request`-kenttää esittämään toivomus tunnelinylläpitoajaksi. Aika ilmoitetaan sekunteina. Vastaus sisältää `time_to_live`-kentän, jossa siis ilmoitetaan yhdyssillan hyväksymä elinaika tunnelille. Tämä voi olla yhtä suuri tai vähemmän kuin pyynnössä esitetty aika.

`ReleaseTunnelRequest`-viestillä terminaalisilta tai yhdyssilta pyytää tunnelin purkamista. Viestin ainoana parametrinä on `time_to_live`-parametri. Jos viestin lähettäjänä on terminaalisilta, viestin `time_to_live`-parametrilla ilmoitetaan aikatoivomus, joka halutaan käyttää yhdyssillassa terminaalisillan GIOP-viestien

välittämiseen (engl. *forwarding*). Vastaavasti sisääntulonsillan lähettämänä tämä parametri kertoo ajan, jonka se käyttää GIOP-viestien välittämiseen. Viestin lähettäjä ei enää lähetä GTP-viestejä tunneliin. Lähettäjä odottaa kuitenkin vastausta viestiin ennen kuin vapauttaa yhteyden. Tunnelinpurkupyyntöön vastataan `ReleaseTunnelReply`-viestillä, joka niin ikään sisältää vain `time_to_live`-parametrin. Sen arvo tulee olla yhtä suuri tai pienempi kuin siihen liittyvän `ReleaseTunnelRequest`-viestin. Lähettäjä ei enää lähetä tunneliin GTP-viestejä `ReleaseTunneReply`-viestin jälkeen.

`HandoffTunnelRequest`-viestin lähettää yhdyssilta terminaalisillalle. Tällä viestillä käynnistetään tunnelinvaihto ja siinä viedään parametrinä lista yhdyssiltojen osoitteita. Viestin lähettäjä ei tämän viestin jälkeen lähetä GTP-viestejä tunneliin ennen kuin on saanut vastauksena `HandoffTunnelReply`-viestin tai ennen kuin mahdollinen toteutuskohtainen aikaraja on saavutettu. Sen sijaan yhdyssilta vastaanottaa edelleen GTP-viestejä terminaalisillalta, jotka se hylkää tai käsittelee `HandoffTunnelReply`-viestin mukaan. Mikäli tietyn aikavälin kuluessa ei tätä viestiä saada, voi yhdyssilta lähettää `ReleaseTunnelRequest`-viestin. Terminaalisilta ilmoittaa `HandoffTunnelReply`-viestissä tunnelinvaihdon onnistumisesta. Jos vaihto onnistuu, terminaalisilta lähettää vastauksen jälkeen `ReleaseTunnelRequest`-viestin tunnelin purkamiseksi. Terminaalisilta ei kuitenkaan välttämättä tue menetelmää, jossa uusi tunneli luodaan vanhan rinnalle. Tästä ilmoitetaan tällöin `HandoffTunnelReply`-viestissä, jonka vastaanotettuaan yhdyssilta lähettää `ReleaseTunnelRequest`-viestin. Jos päätelaite ei pysty luomaan uutta tunnelia, ilmoitetaan handoff-pyynnön hylkäämisestä HandoffTunnelReply-viestissä. Tällöin tunnelia ei pureta ennen kuin vaihdetaan `ReleaseTunnelRequest`- ja `ReleaseTunnelResponse`-viestit.

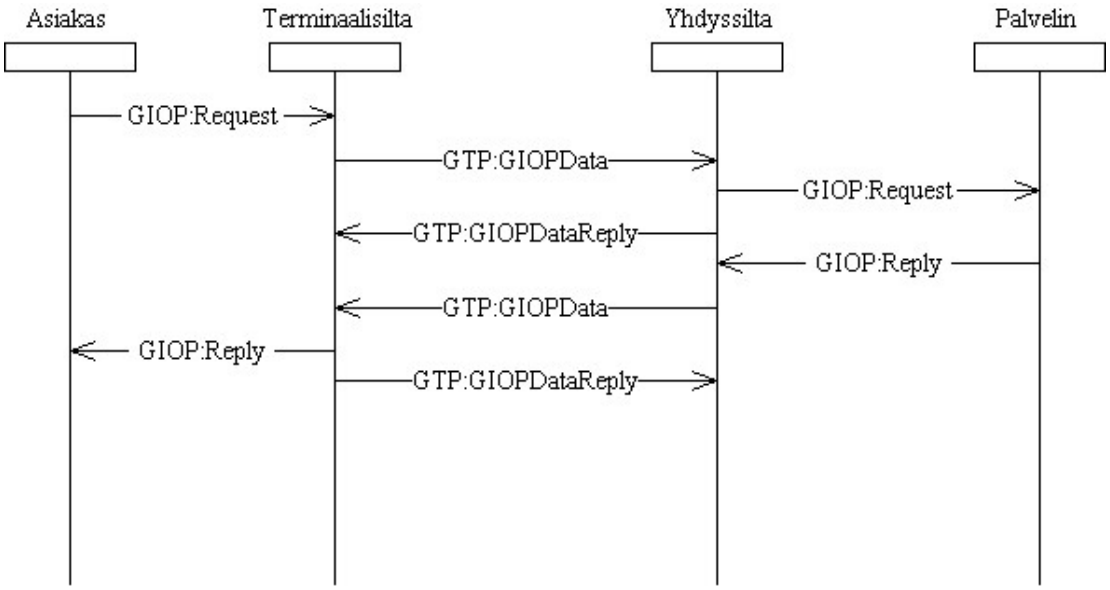
`OpenConnectionRequest`-viestillä terminaalisilta tai yhdyssilta yrittää varata yhteyden palvelinolioon toisiltaan. Pyyntöön liitetään palvelinolion viitteen lisäksi tunniste, joka palautetaan vastauksessa. Vastaukseen ei näin ollen tarvitse liittää palvelinolion viitettä. Lisäksi pyyntöön liitetään aikaraja, jonka aikana yhteyden tulisi muodostua. `OpenConnectionReply`-viestissä ilmoitetaan, onko yhteyden muodostus onnistunut. Mikäli yhteydenmuodostuksessa on tapahtunut virheitä, näistä ilmoitetaan

vastausviestissä. Lisäksi vastausviestissä palautetaan yhteystunniste sekä tunniste yhteydenmuodostuspyynnölle, jolla vastaus liitetään oikeaan pyyntöön.

Yhteyden purkamista varten lähetetään `CloseConnectionRequest`-viesti. Kuten yhteydenmuodostuspyynnön, myös yhteydenpurkupyynnön voi lähettää yhdyssilta tai terminaalisilta. Viestiin liitetään ainoastaan yhteystunniste. Yhteystunnisteella 0xFFFFFFFF voidaan pyytää kaikkien tunnelin yhteyksien purkamista (tunnelin multipleksaus mahdollistaa tunnelin jakamisen usean yhteyden kesken). `CloseConnectionReply`-viestillä vastataan yhteydenpurkupyyntöön. Vastaukseen liitetään yhteystunnisteen lisäksi informaatio yhteyden purkamisen onnistumisesta.

`CloseConnectionIndication`-viestin voi lähettää terminaalisilta sekä yhdyssilta. Tällä viestillä ilmoitetaan yhteyden katkeamisesta ilman `CloseConnectionRequest`-viestiä. Viestiin liitetään yhteystunniste ja informaatiota yhteyskatkon syystä. Yhteystunnisteella voidaan viitata kaikkiin tunnelin yhteyksiin samoin kuin `CloseConnectionRequest`-viestissä.

Varsinaisten GIOP-viestien kuljetukseen käytetään GTP:n `GIOPData`- ja `GIOPDataReply`-viestejä. `GIOPData`-viesti sisältää yhteystunnisteen, GIOP-viestin tunnisteen sekä kapseloidun GIOP-viestin. `GIOPData`-viestiin vastataan `GIOPDataReply`-viestillä. GIOP-viestin tunnistetta käytetään vastausviesteissä, jotta ne voidaan liittää lähetettyihin `GIOPData`-viesteihin. Yhteystunnisteella määritetään yhteys, jolle GIOP-viesti on suunnattu. `GIOPData`-viestin voi lähettää joko terminaalisilta tai yhdyssilta. `GIOPDataReply`-viestissä on GIOP-viestin tunnisteen lisäksi informaatio viestin kuljetuksen onnistumisesta. Kuljetuksen epäonnistuminen voi johtua esimerkiksi virheellisestä yhteystunnisteesta. Yhden pyynnön ja sen vasteen kuljettamiseen tarvitaan siis kahdeksan sanomaa, joista neljä ovat GTP-sanomia. Kuvan 10 sekvenssikaaviossa on esitetty kutsun eteneminen asiakkaalta palvelimelle GIOP-tunnelin läpi.



Kuva 10. GIOP-viestien välitys GIOP-tunnelin kautta.

Yhdyssilta tai terminaalisilta välittävät `GTPForwardMessage`-viestillä GTP-viestejä vanhalle yhdyssillalle tai vanhalta yhdyssillalta. Viesti koostuu yhdyssillan viitteestä, GTP-viestin tunnisteesta sekä kapseloidusta GTP-viestistä. Yhdyssillan viite on kohde-osoite viestin lähettäjän ollessa yhdyssilta ja lähde-osoite viestin lähettäjän ollessa terminaalisilta. `GTPForwardMessage`-viestiin vastataan `GTPForwardMessageReply`-viestillä, johon liitetään `GTPForwardMessage`-viestin GTP-viestin tunniste. Tällä voidaan liittää vastausviestit oikeisiin `GTPForwardMessage`-viesteihin. GTPForwardMessageReply-viesti sisältää GTP-viestin tunnisteen lisäksi informaatiota välityksen onnistumisesta.

GTP:hen liittyvistä virheistä ilmoitetaan `Error`-viestillä, jonka voi lähettää terminaalisilta ja yhdyssilta. Viesti sisältää virheeseen liittyvän GTP-viestin numeron sekä virhekoodin [OMG3].

3.3.3 TCP-tunnelointi

TCP tunneloinnissa GTP-viestit siirretään tavuvirtana (engl. *byte stream*) ilman täytebittejä (engl. *padding*). Kohdeosoite annetaan joko IP-osoitteena ja porttinumerona tai DNS-isäntänimenä ja porttiosoitteena [OMG3]. Yksityiskohtaisempaa tietoa TCP-protokollasta on lähteessä [S1].

3.3.4 UDP-tunnelointi

UDP on TCP/IP-protokollaperheeseen kuuluva yhteydetön kuljetuskerroksen protokolla. UDP-tunneloinnissa GTP-viestit siirretään UDP-tunnelointiprotokollalla eli UTP:llä (engl. *UDP Tunneling Protocol*), joka kulkee UDP-tietosähkeessä. UDP-tunnelointia käytettäessä kohdeosoite annetaan IP-osoitteella ja porttinumerolla. UDP on esitelty tarkemmin lähteessä [S1].

UTP tarjoaa GIOPin vaatiman kuljetuskerroksen luotettavuuden ja pakettien oikean järjestyksen. UTP puolestaan olettaa, ettei se saa korruptoitunutta dataa. UTP määrittelee GTP-viestien kapseloinnin. UTP sisältää tuen viestien segmentoinnille ja uudelleen kokoamiselle. Yhdessä UTP-viestissä voidaan myös kuljettaa useita GTP-viestejä. Kullakin GTP-viestillä on UTP-sanomassa oma lohkonsa (engl. *chunk*). UTP-viesti siirretään siis UDP-protokollan datakentässä ja se koostuu UTP-otsikosta ja yhdestä tai useammasta UTP-lohkosta. UTP:n otsikko on neljän tavun pituinen. Otsikko käsittää järjestysnumeron sekä sanoman sisältämien lohkojen lukumäärän. UTP:ssä merkkijonot ovat aina 8-bittisessä ANSI ASCII –formaatissa.

UTP-lohkon perusrakenne on TFLV (*type-flags-length-value*), eli UTP-viestin tyyppi, liput, viestin arvon pituus sekä viestin arvo. Tyyppi ilmoitetaan oktetilla. Kaikki viestit eivät sisällä lippuja. Tarvittaessa lipuille on varattu myös oktetti. Viestin arvokentän pituus ilmoitetaan 0-2:een oktetilla. Viestin arvo sisältää UTP lohkon sisältävän informaation.

Tyyppi	Tyyppiarvo
`InitialAccessRequest`	0x01
`InitialAccessReply`	0x02
`Pause`	0x03
`Resume`	0x04
`Acknowledgement`	0x05
`GTPData`	0x06

Taulukko 3. UTP-lohkotyypit.

UTP-lohkoja on kuusi erilaista, joiden tyypit on kuvattu taulukossa 3. `InitialAccessRequest` lähetetään terminaalisillan toimesta. Viesti sisältää lippu- ja pituuskentät. Lipuilla ilmoitetaan viestin mahdollisesta pilkkomisesta. Arvokenttä sisältää evästyksen (engl. *cookie*) sekä terminaalisillan osoitteen. Evästys on terminaalisillan valitsema bittikuvio ja osoite koostuu IP-numerosta sekä UDP-portista (esim. ”192.168.0.50:9876”). Vastauksena tähän yhdyssilta lähettää `InitialAccessReply` -lohkon, joka sisältää myös lippu- ja pituuskentät. Arvokenttä sisältää evästyksen (engl. *cookie*) sekä terminaalisillan osoitteen.

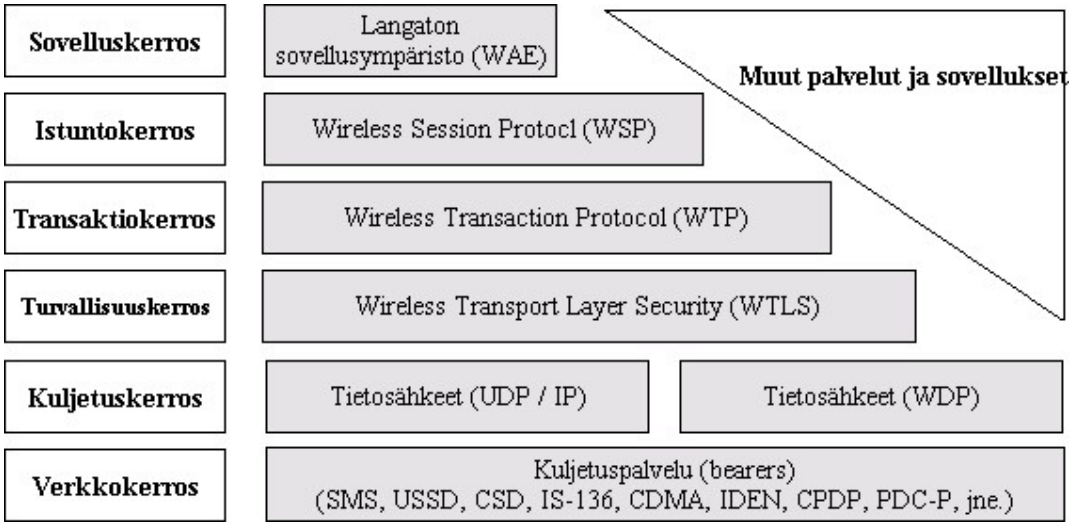
`Pause`-lohkon voi lähettää yhdyssilta ja terminaalisilta. Viestiin ei kuulu lippu-, pituus- eikä arvokenttää. Viestin vastaanottajan tulisi tulkita viesti siten, että viestin lähettäjä hylkää kaikki UTP-viestit ennen `Resume`-lohkoa, jolla lähettäjä ilmoittaa olevansa valmis vastaanottamaan jälleen viestejä. Resume-viestiin ei niin ikään liity lippu-, pituus- eikä arvokenttää.

`Acknowledge`-lohkossa ei ole lippukenttää. Lohkon voi lähettää yhdyssilta ja terminaalisilta. Lohkon pituuskentässä ilmoitetaan merkintöjen (engl. *entry*) lukumäärä arvokentässä. Arvokentän pituus kaksi kertaa pituuskentän sisältöä pidempi. Arvokentässä ilmoitetaan vastaanotetuista UTP-viesteistä korkein järjestysnumero sekä mitä muita numeroita on vastaanotettu.

`GTPData` lohkossa lipuilla ilmoitetaan viestin pilkkomisesta (engl. *fragmentation*). Lipuilla ilmoitetaan onko segmentti GTP-viestin ensimmäinen, viimeinen vai sijaitseeko segmentti GTP-viestin keskellä. Lisäksi ilmoitetaan jos viestiä ei ole pilkottu. Pituuskenttä ilmoittaa lohkon arvokentän pituuden [OMG3].

3.3.5 WAP-tunnelointi

WAP (*Wireless Application Protocol*) on standardi langattomille päätelaitteille, joka kattaa tiedon esittämisen sekä tiedon kuljetuksen. WAP-standardi on siis varsin laaja. Oleellista GIOP-tunneloinnin kannalta on WAPin määrittelemä protokolla-arkkitehtuuri, joka perustuu olemassaoleviin Internet-standardeihin, kuten XML sekä IP. WAPin protokollapinolla (engl. *protocol stack*) mahdollistetaan yhteensopivuus erilaisten siirtoverkkojen välillä sekä minimoidaan siirtoverkon kaistanleveyteen kohdistuvat vaatimukset. WAPin protokollapino on esitetty kuvassa 11.



Kuva 11. WAP-stadardin protokollapino.

WAP Forum tekee WAPiin liittyvää määrittelytyötä. Tarkempaa tietoa WAPista on lähteissä [W1] ja [W2] sekä WAP Forumin kotisivuilla (`http://www.wapforum.org`).

GTP-viestien WAP-tunneloinnissa käytetään WAPTP:tä (*WAP Tunneling Protocol*) GTP-viestien välittämiseen yhdyssillan ja terminaalisillan välillä. WAPTP perustuu WAP-standardiin. WAPTP:n suunnittelussa on painotettu yksinkertaista toteutusta. WAPTP:tä oletetaan käytettävän pienissä sulautetuissa järjestelmissä, joissa resurssit ovat rajalliset. WAPTP varmistaa, että GTP:n vaatimukset täytetään (eli säilytetään viestien järjestys eikä korruptoitunutta dataa välitetä).

WAPTP käyttää WAP-standardin määrittelemää yhteydetöntä WDP:tä (*Wireless Datagram Protocol*), joka soveltuu monenlaisiin verkkoihin [ks. kuva 11]. WDP tarjoaa samankaltaista palvelua kuin UDP. Myös WDP:ssä käytetään porttinumeroita sovellusten tunnistamiseen kuljetusosoitteessa. WDP sijaitsee eräänlaisen kuljettimen (engl. *bearer service*) päällä. Mikäli kuljetin ei tue viestien segmentointia ja uudelleenkokoamista tulee WDP toteutukseen huolehtia tästä. Kuljettimessa tietosähkeen maksimikoko on kuljetinkohtainen. Kuljetin olettaa, ettei GTP-toteutus yritä lähettää maksimikokoa suurempia tietosähkeitä kuljettimelle. Tästä seuraa, että myös oliovälittimen tulee myös tuntea tietosähkeelle asetetut rajoitukset ja tarvittaessa pilkkoa GIOP-viestit. WDP takaa tietosähkeen segmenttien oikean järjestyksen, mutta ei tietosähkeiden oikeaa järjestystä.

WAPTP sovittaa yhden GTP-viestin yhteen WDP-tietosähkeeseen. Tietosähkeiden oikean järjestyksen varmistamiseksi WAPTP:n tulee viivyttää GTP-viestien lähetystä, joiden järjestysnumero on odotettua korkeampi. WDP tukee useita osoiteformaatteja, kuten

- IP-osoitteita (IPv4 ja IPv6),
- puhelinnumeroiden (MSISDN) eri formaatteja (IS_637, ANSI_I36, GSM, CDMA, iDEN, FLEX ja TETRA),
- GSM-palvelukoodia (engl. *GSM_Service_Code*),
- TETRA_ISIa sekä
- Mobitex MANia.

Tavallisesti käytetään IP-osoitteita tai puhelinnumeroita. IP-osoitteen tulee olla desimaalimuodossa, ettei DNS:n (*Domain Name Service*) käyttöä tarvita [OMG3].

3.4 Kanavanvaihto ja yhteyden palautus

Kanavanvaihdolla (engl. *Handoff*) tarkoitetaan tässä tapahtumaa, jossa mobiili päätelaite vaihtaa yhdyssiltaa. Tapahtumassa yhdyssilta ikään kuin ”luovuttaa” päätelaitteen toiselle yhdyssillalle. Yleisesti kanavanvaihto koostuu kolmesta vaiheesta, jotka ovat tiedon keruu, päätöksenteko sekä suoritus. ORB-kerrokselle näkyvä yhdyssillan vaihto on osa suoritusvaihetta. GTP:ssä on kanavanvaihtotuki versiosta 2.0 alkaen.

Kanavanvaihto voi tapahtua joko etuperin (engl. *forward*) tai takaperin (engl. *backward*). Takaperin tapahtuva kanavanvaihto tarkoittaa tavallista tapausta, jossa päätelaite vaihtaa yhdyssiltaa ja etuperin tapahtuvalla kanavanvaihdolla tarkoitetaan tilannetta, jossa yhteyskatkon jälkeen muodostetaan yhteys yhdyssiltaan. Tässä käytetään takaperin tapahtuvasta kanavanvaihdosta termiä **kanavanvaihto** ja etuperin tapahtuvasta kanavanvaihdosta termiä **yhteyden palautus**.

Yhdyssiltarajapinta tarjoaa menetelmät kanavanvaihdon käynnistämiseen (`start_handoff`-operaatio). Myös päätelaitteen `MobileTerminal`-moduulin `HandoffCallback`-rajapinta tarjoaa operaation (`report_handoff_status`), jolla yhdyssilta puolestaan voi ilmoittaa kanavanvaihdon onnistumisesta.

Kanavanvaihto voidaan käynnistää päätelaitteen lisäksi myös verkosta. Tällöin jokin ulkoinen sovellus kutsuu päätelaitetta palvelevan yhdyssillan `start_handoff`-operaatiota. Kanavanvaihdon yhteydessä vanhalla yhdyssillalla tarkoitetaan yhdyssiltaa, joka luovuttaa sillan toiselle, ns. uudelle yhdyssillalle. Kanavanvaihdossa oletetaan, että päätelaite pystyy muodostamaan yhteyden uuteen yhdyssiltaan ennen kuin vapauttaa yhteyden vanhaan yhdyssiltaan [OMG3].

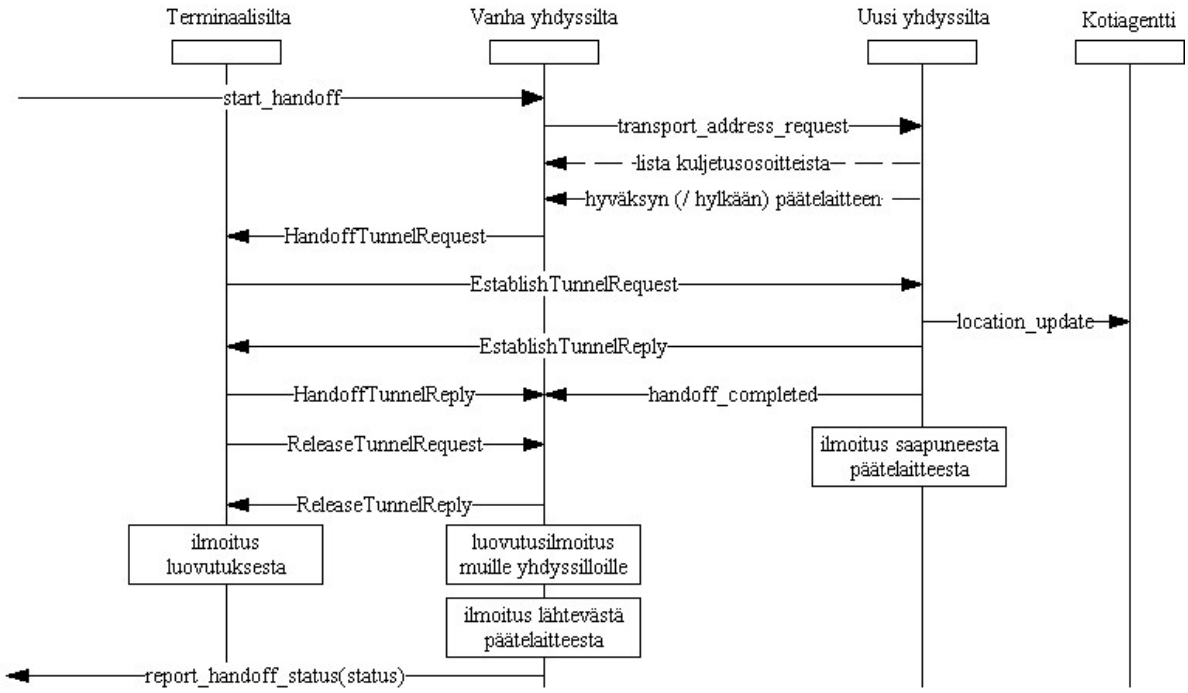
3.4.1 Verkosta käynnistetty kanavanvaihto

Kuvan 12 viestikaaviossa on kuvattu onnistunut verkosta käynnistetty kanavanvaihto. Kanavanvaihdon käynnistää vanhan yhdyssillan vastaanottama `start_handoff`-viesti. Tällöin vanha yhdyssilta lähettää uudelle yhdyssillalle `transport_address_request`-viestin, johon uusi yhdyssilta lähettää vastauksena listan uuden yhdyssillan kuljetusosoitteista sekä boolean-arvon. Tällä arvolla ilmoitetaan, hyväksyykö uusi yhdyssilta päätelaitteen vai ei. Mikäli päätelaite hylätään, keskeytetään kanavanvaihto ja vanha yhdyssilta jatkaa päätelaitteen yhdyssiltana. Jos uusi yhdyssilta hyväksyy päätelaitteen, lähettää vanha yhdyssilta terminaalisillalle `HandoffTunnelRequest`-viestin. Tämän saatuaan terminaalisilta yrittää muodostaa yhteyden uuteen yhdyssiltaan. Yhteyden muodostuksen onnistumisesta lähetetään vanhalle yhdyssillalle tieto `HandoffTunnelReply`-viestin `status`-parametrillä.

Mikäli yhteyden ja tunnelinmuodostus uuteen yhdyssiltaan epäonnistuu, keskeytetään kanavanvaihto ja vanha yhdyssilta jatkaa päätelaitteen yhdyssiltana. Jos uusi yhdyssilta hyväksyy tunnelinmuodostuksen, lähettää se päätelaitteen kotiagentille päivitysviestin. Onnistuneen päivitysviestin jälkeen lähetetään vanhalle yhdyssillalle `handoff_completed`-ilmoitus. Mikäli yhdyssillassa tuetaan mobiliteettitapahtumia, voidaan näillä ilmoittaa uuden päätelaitteen liittymisestä yhdyssiltaan.

Vanha yhdyssilta voi lähettämänsä HandoffTunnelRequest-viestin jälkeen saada `HandoffTunnelReply`-viestin ja `handoff_completed`-ilmoituksen missä tahansa järjestyksessä. `HandoffTunnelReply`-viestistä tarkistetaan `status`-parametri, joka ilmoittaa kanavanvaihdon onnistumisesta. Mikäli kanavanvaihto on epäonnistunut, jatkaa vanha yhdyssilta päätelaitteen yhdyssiltana. Muussa tapauksessa yhdyssilta jää odottamaan `ReleaseTunnelRequest`-viestiä terminaalisillalta. Tämän saatuaan vanha yhdyssilta lähettää vastauksena `ReleaseTunnelReply`-viestin, jonka jälkeen tunneli terminaalisillan ja vanhan yhdyssillan väliltä puretaan. Jos terminaalisilta tukee mobiliteettitapahtumia, voi se ilmoittaa tapahtumalla kanavanvaihdosta.

Vastaanotettuaan `handoff_completed`-viestin vanha yhdyssilta tietää, että uusi yhdyssilta on ottanut päätelaitteen vastuulleen. Vanha päätesilta ilmoittaa muille päätelaitteen liikkumisesta kiinnostuneille yhdyssilloille kanavanvaihdosta. Jos vanha yhdyssilta tukee mobiliteettitapahtumia, voi se ilmoittaa tapahtumalla poistuneesta päätelaitteesta [OMG3].



Kuva 12. Verkosta käynnistetty kanavanvaihto.

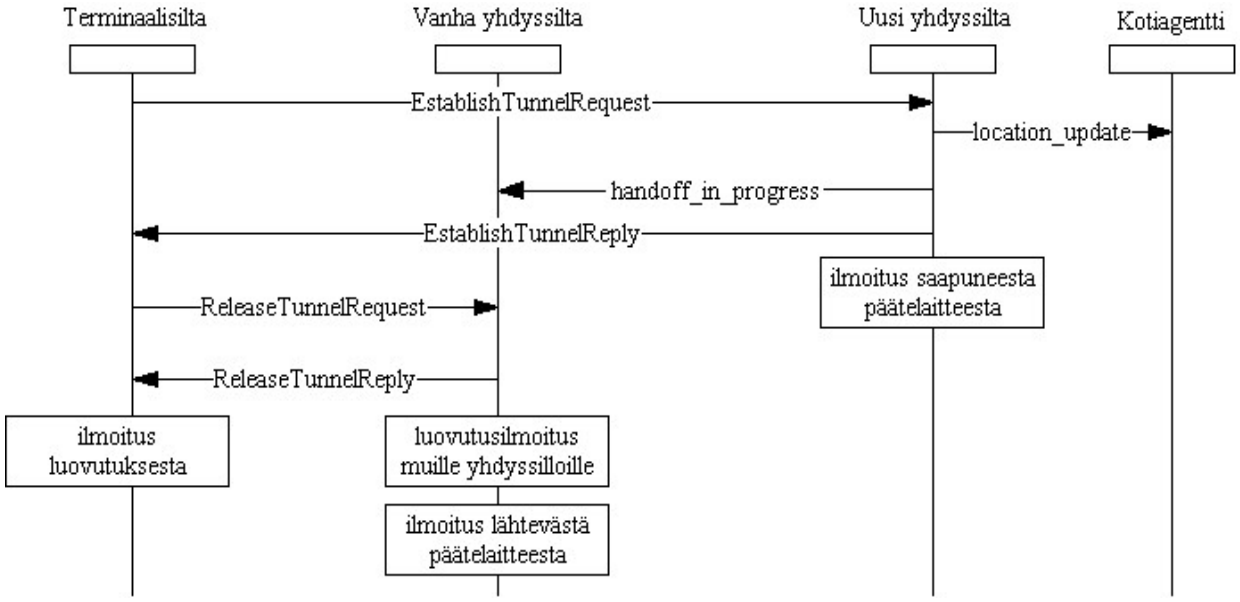
3.4.2 Päätelaitteen käynnistämä kanavanvaihto

Päätelaitteen käynnistämässä kanavanvaihdossa vaaditaan, että päätelaite kykenee muodostamaan yhteyden uuteen yhdyssiltaan ennen kuin yhteys vanhaan yhdyssiltaan puretaan. Mikäli näin ei voida toimia, joudutaan käyttämään yhteyden palautusta. Kuvan 13 viestikaaviossa on esitetty onnistunut päätelaitteen käynnistämä kanavanvaihto.

Päätelaitteen terminaalisilta käynnistää kanavanvaihdon muodostamalla yhteyden uuteen yhdyssiltaan ja lähettämällä sille tunnelinmuodostuspyynnön. Jos uusi yhdyssilta hyväksyy tunnelinmuodostuksen, lähettää se päivitysviestin päätelaitteen kotiagentille. Mikäli

päivitysviestin lähetys epäonnistuu, keskeytyy kanavanvaihto ja terminaalisillalle ilmoitetaan tästä `EstablishTunnelReply`-viestin parametrillä. Muussa tapauksessa vanhalle yhdyssillalle lähetetään `handover_in_progress`-ilmoitus, jolla ilmoitetaan kanavanvaihdon olevan käynnissä.

Tämän jälkeen terminaalisillalle lähetetään `EstablishTunnelReply`-viesti, jossa kerrotaan tunnelinmuodostuksen onnistumisesta. Jos tunnelinmuodostus on epäonnistunut purkaa terminaalisilta yhteyden uuteen yhdyssiltaan ja vanha yhdyssilta jatkaa terminaalisillan yhdyssiltana. Mikäli tunnelinmuodostus on onnistunut, lähettää terminaalisilta vanhalle yhdyssillalle tunnelinpurkupyynnön (`ReleaseTunnelRequest`). Vanha yhdyssilta vastaa tähän `ReleaseTunnelReply`-viestillä, jonka jälkeen tunneli terminaalisillan ja vanhan yhdyssillan välillä puretaan. Mikäli terminaalisillassa ja yhdyssilloissa tuetaan mobiliteettitapahtumia, voidaan kanavanvaihdosta ilmoittaa tapahtumilla, kuten verkon käynnistämässä kanavanvaihdossa [OMG3].



Kuva 13: Päätelaitteen käynnistämä kanavanvaihto.

3.4.3 Yhteyden palautus

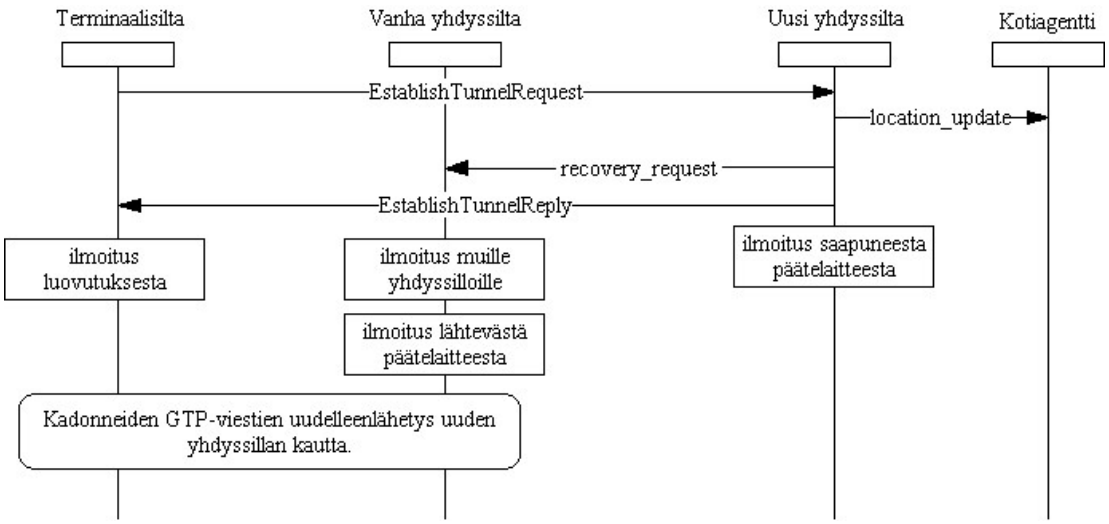
Kuten kanavanvaihdossa, myös yhteyden palautuksessa on kaksi toimintamallia. Yhteys joko palautetaan vanhaan yhteyssiltaan tai sitten yhteys muodostetaan uuteen yhdyssiltaan. Yhteyden palautus käynnistetään terminaalisillassa, kun havaitaan yhteyskatko yhdyssiltaan.

Yhteyden palauttamisessa vanhaan yhdyssiltaan terminaalisilta lähettää tunnelinmuodostuspyynnön (`EstablishTunnelRequest`). Yhdyssilta tulkitsee pyynnöstä, että kyseessä on yhteyden uudelleenmuodostus. Lisäksi pyynnössä ilmoitetaan terminaalisillan viimeksi vastaanottaman GTP-viestin järjestysnumero. Järjestysnumeron perusteella yhdyssilta lähettää kadonneet viestit uudelleen. Mikäli yhdyssilta tukee mobiliteettitapahtumia, ilmoittaa se yhteyden palauttamisesta tapahtumalla ainoastaan, jos se on ilmoittanut tapahtumalla myös yhteyskatkosta.

Yhdyssilta vastaa tunnelinmuodostuspyyntöön `EstablishTunnelReply`-viestillä, josta voidaan lukea, onko vastauksen lähettänyt yhdyssilta sama kuin vanha yhdyssilta. Mikäli näin on, voidaan vastauksesta lukea myös viimeisen yhdyssillan vastaanottaman GTP-viestin järjestysnumero ja terminaalisilta voi lähettää kadonneet viestit uudelleen. Jos terminaalisilta tukee mobiliteettitapahtumia, voi terminaalisilta ilmoittaa yhteyden palauttamisesta tapahtumalla.

Mikäli yhteyden palautuksessa otetaan yhteys uuteen yhdyssiltaan, havaitsee uusi yhdyssilta yhteydenpalautusyrityksen tunnelinmuodostuspyynnöstä. Mikäli tunnelinmuodostus hyväksytään, ilmoitetaan tästä `location_update`-viestillä päätelaitteen kotiagenttille. Mikäli päivitysilmoitus epäonnistuu, ilmoitetaan tästä `EstablishTunnelRequest`-viestissä ja myös yhteyden palautus epäonnistuu. Muussa tapauksessa uusi yhdyssilta lähettää vanhalle yhdyssillalle `recovery_request` – viestin, jonka saatuaan vanha yhdyssilta ilmoittaa tapahtuneesta muille päätelaitteen liikkeistä kiinnostuneille yhdyssilloille. Mikäli vanha yhdyssilta tukee mobiliteettitapahtumia, ilmoitetaan päätelaitteen poistumisesta tapahtumalla. Tämän lisäksi vanha yhdyssilta välittää yhteyskatkon vuoksi kadonneet GTP-viestit uuden yhdyssillan kautta päätelaitteelle.

Vastaanottaessaan `EstablishTunnelReply`-viestin terminaalisilta tarkastaa, onko tunnelinmuodostus onnistunut. Onnistuneen tunnelinmuodostuksen jälkeen terminaalisilta lähettää yhteyskatkon vuoksi kadonneet viestit vanhalle yhteyssillalle uuden yhteyssillan kautta. Kuvassa 14 on kuvattu yhteyden palauttaminen uuden yhdyssillan avulla [OMG3].



Kuva 14. Yhteyden palauttaminen.

3.4.4 GTP-viestien välitys

GTP-viestien välityksellä (engl. *forwarding*) tarkoitetaan tässä toimenpidettä, jossa vanha yhdyssilta välittää päätelaitteelle suunnatut palvelupyynnöt uudelle yhdyssillalle. Yhdyssillan `AccessBridge`-rajapinta määrittelee operaatiot GTP-viestien välitykseen. Viestinvälityksen tarve johtuu GIOPista, jossa vaaditaan, että GIOP-vasteet lähetetään samalla yhteydellä, jolla pyynnöt saapuivat. Koska GIOP-yhteyden toinen päätepiste on yhdyssilta, on vastaukset välitettävä perille silloinkin, kun päätelaite on vaihtanut yhdyssiltaa.

Vastaanottaessaan yhdyssiltaa vaihtaneelle päätelaitteelle tarkoitetun viestin, voi vanha yhdyssilta selvittää päätelaitteen uuden yhdyssillan esimerkiksi kotiagentilta. Tämä viesti kuljetetaan uudelle yhdyssillalle `AccessBridge`-rajapinnan `gtp_from_terminal`-viestillä Uusi yhdyssilta välittää viestin päätelaitteelle `GTPForward`-viestillä. Vastaavasti, jos päätelaite haluaa lähettää viestin vanhan yhdyssillan kautta, lähettää se

`GTPForward`-viestin uudelle yhdyssillalle. Tämä välittää viestin vanhalle yhdyssillalle `AccessBridge`-rajapinnan `gtp_from_terminal`-viestillä [OMG3].

3.4.5 Päätelaitteiden jäljitys

Vanhan yhdyssillan on tunnettava päätelaitteen uusi yhdyssilta niin kauan kuin sillä on avoimia GIOP-yhteyksiä päätelaitteeseen. Tätä varten `AccessBridge`-rajapinta sisältää operaatiot päätelaitteiden jäljitykseen (engl. *terminal tracking*).

Oletetaan, että päätelaite on vaihtamassa yhdyssillasta A yhdyssiltaan B. Tällöin yhdyssilta A ilmoittaa tapahtuneesta myös yhdyssillalle C, josta päätelaite aiemmin vaihtoi yhdyssiltaan A. Lisäksi A ilmoittaa asiasta muille informaatiota tarvitseville yhdyssilloille D ja E. Jotta yhdyssillat C, D ja E saisivat ilmoituksen seuraavasta yhdyssillan vaihdoista, on niiden rekisteröitävä kiinnostuksensa `subscribe_handoff_notice`-viestillä. Viesti lähetetään päätelaitteen uudelle yhdyssillalle B. Seuraavan yhdyssillan vaihdon yhteydessä B ilmoittaa kiinnostuksensa rekisteröineille vaihdosta `handoff_notice`-viestillä [OMG3].

3.5 Palvelut langattomassa ympäristössä

OMG on julkaissut vuoden 2001 kesällä RFI-dokumentin (*Request For Information*) jossa haetaan tietoa langattomassa ympäristössä tarpeellisista CORBA-palveluista. RFI:n ja sen vastausten perusteella tuotetaan RFP-dokumentti (*Request For Proposal*), jossa esitetään vaatimukset spesifikaatiolle. Palveluiden määrittely on siis hyvin varhaisessa vaiheessa, eikä mitään erityisiä CORBAn mobiliteettipalveluita ole vielä määritelty.

RFI:ssä haetaan tietoa seuraaviin kysymyksiin [OMG5]:

- Sijaintipalvelut - Minkälaisia ja minkä tyyppisiä sijaintipalveluita tarvitaan esimerkiksi päätelaitteen liikkumisesta kiinnostuneille sovelluksille ?
- Ohjelmistonhallintapalvelut – Miten ohjelmiston lataaminen (engl. *download*) ja päivitys (engl. *upgrade*) tulisi hoitaa ?

- QoS-sovitukset – Kuinka QoS sovitetaan langattomaan tiedonsiirtoon ja kuinka se käsitellään ORB-tasolla ?
- Roskien keruu (engl. *Garbage Collection*) – Onko roskien keruu mobiliteettikohtaista ? Miten se tulisi toteuttaa tässä tapauksessa ?
- Autentikointi – Onko mobiililaitteille erityisiä autentikointivaatimuksia ?
- Liikkuvuudenhallinta (engl. *Mobility Management*) – Minkälaisia liikkuvuudenhallintapalveluita tarvitaan ORB-tasolla ?
- Henkilökohtainen liikkuvuus (engl. *personal mobility*) – Miten henkilökohtaisen liikkuvuuden tuki sisältyy CORBA-spesifikaatioon ja mitä laajennuksia siihen tarvitaan ?
- Muut palvelut – Mitä muita palveluita ORBin tulisi toteuttaa ?

Langattomien päätelaitteiden rajoitukset huomioon ottaen on toistaiseksi vielä mahdotonta tarjota täydellisesti CORBA-standardin täyttävää toteutusta palveluineen. Tämän vuoksi langattomille päätelaitteille toteutettavat CORBA-toteutukset tulevat olemaan toiminnoiltaan suppeampia kuin vastaavat tehokkaampiin suoritusympäristöihin suunnatut toteutukset. CORBAn soveltamista rajoitettuihin ympäristöihin on käsitelty luvussa 2.6.

4 Esimerkkisovellus

Tässä luvussa määritellään, suunnitellaan ja toteutetaan yksinkertainen esimerkkisovellus, jolla simuloidaan langatonta sovellusympäristöä. Ensin määritellään järjestelmälle vaatimukset, jonka jälkeen suunnitellaan järjestelmän arkkitehtuuri ja toiminnallisuus.

Esimerkkisovelluksen yhteydessä oli tarkoituksena testata langatonta CORBAa aidossa langattomassa ympäristössä langattomilla päätelaitteilla, mutta tämä ei ollut laitteiston puuttumisen vuoksi mahdollista. Tästä johtuen langatonta ympäristöä jouduttiin simuloimaan TCP/IP-verkossa.

Järjestelmän testauksessa keskityttiin simuloidun langattoman ympäristön suorituskyvyn selvittämiseen. Tässä luvussa tarkoitetaan **työasemasovelluksella** TCP/IP-verkkoon liitetyllä työasemalla suoritettavaa CORBA-sovellusta, jonka toiminta perustuu täysin IIOP:hen. Vastaavasti **langattomalla sovelluksella** tai **terminaalisovelluksella** tarkoitetaan sovellusta, joka kommunikoi työasemasovellusten kanssa GIOP-tunnelin läpi. Fyysisesti langatonta sovellusta suoritetaan esimerkissä tavallisella työasemalla.

4.1 Järjestelmän vaatimukset

Järjestelmän tarkoituksena on testata CORBAn toimintaa ja suorituskykyä langattomassa ympäristössä sekä tutkia simuloidun langattoman ympäristön aiheuttamia viiveitä vasteaikoihin. Vertailupohjana käytetään vastaavaa järjestelmää kiinteässä verkossa. Järjestelmässä kiinteässä verkossa sijaitsevaa palvelinta kutsutaan työasema-asiakkaalla sekä simuloidussa langattomassa ympäristössä sijaitsevalla asiakkaalla. Langatonta palvelinta ei esimerkkisovelluksessa toteuteta.

4.1.1 Palvelinsovellus

Järjestelmän palvelinsovellus vastaa asiakassovellusten pyyntöihin, joiden avulla selvitetään palvelupyynnön välittämiseen, suorittamiseen sekä vasteen palautukseen kulunut kokonaisaika. Palvelinsovelluksen tarvitsee ainoastaan toteuttaa määritelty rajapinta. Tämän tutkielman puitteissa palvelinsovelluksen toiminta testataan ainoastaan työasemalla.

4.1.2 Asiakassovellus

Järjestelmän asiakassovellus esittää palvelinsovellukselle palvelupyyntöjä ja laskee palvelupyyntöjen välittämiseen, suorittamiseen sekä vasteen palautukseen kuluneen kokonaisajan.

Testituloksista tuotetaan raportti tiedostoon. Raportista tulee ilmetä palvelupyyntöjen määrä, palvelupyyntöön kulunut keskimääräinen aika sekä palvelupyyntöihin käytetty kokonaisaika. Asiakassovellusta voidaan suorittaa työasemalla ja langattomalla päätelaitteella.

4.1.3 Toteutustyökalut

ORB-toteutuksena käytetään MIWCOa, joka on OMG:n langattoman spesifikaation toteuttava laajennus MICOon (*Mico is CORBA*). MIWCOn ohjelmakirjastot perustuvat langattoman CORBAn määrittelytyöhön. MICO on ns. *open-source* –toteutus CORBA-standardista. MICO-projektin tarkoituksena on toteuttaa vapaasti saatavilla oleva toteutus CORBA-standardista. MICO tukee ainoastaan C++ -ohjelmointikieltä [M1].

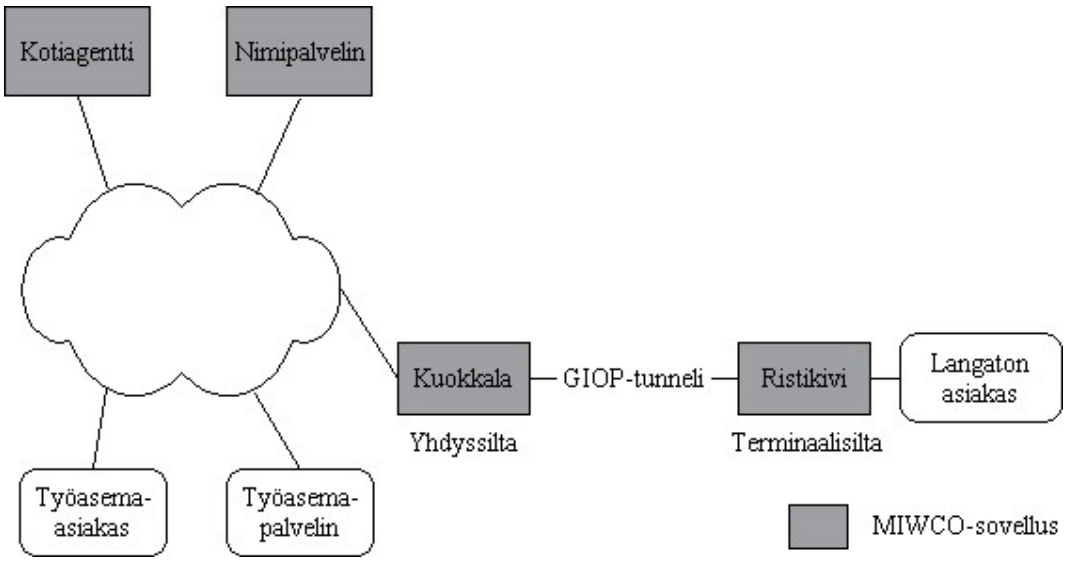
MIWCO-laajennus käsittää toteutukset kotiagentista, yhdyssillasta ja terminaalisillasta [ks. luku 3.2] sekä GIOP-viestien tunneloinnista GTP:n avulla [ks. luku 3.3]. Laajennus on toteutettu Helsingin yliopistossa **Jaakko Kangasharjun** toimesta. MIWCOsta on saatavilla lähdekoodit, joten se on siis ensin käännettävä toteutusympäristössä.

Kääntämistä (engl. *compile*) varten MICOn mukana on make-tiedostoja UNIX-ympäristössä ja Windows-ympäristössä (Microsoftin VisualC++ -kääntäjällä) tapahtuvalle kääntämiselle. Langattoman laajennuksen kääntäminen Windows-ympäristössä nmake-työkalulla vaati make-tiedostojen sekä toteutuksen tarkastamista ja vähäistä muokkaamista.

4.2 Järjestelmän arkkitehtuuri

Koejärjestelmän arkkitehtuuri on esitetty kuvassa 15. Suorituskyvyn testaamiseksi langattomassa ympäristössä järjestelmässä suoritetaan asiakkaita niin langattomina kuin työasemasovelluksina. Palvelinta suoritetaan ainoastaan työasemasovelluksena. Palvelin ja

asiakassovellukset ovat toteutukseltaan identtisiä työasemasovelluksena ja langattomana sovelluksena. Terminaalisillat, yhdyssillat sekä kotiagentti käyttävät nimipalvelinta.



Kuva 15. Koejärjestelmän arkkitehtuuri.

Järjestelmä toteutetaan lähiverkossa MIWCOn tarjoamilla sovelluksilla. Näiden itsenäisten sovellusten (engl. *stand-alone software*) avulla voidaan lähiverkossa simuloida langattoman verkon toimintaa. Itsenäisillä sovelluksilla tarkoitetaan tässä kotiagenttia, yhdyssiltaa (*Kuokkala*) ja terminaalisiltaa (*Ristikivi*).

4.2.1 IDL-rajapinta

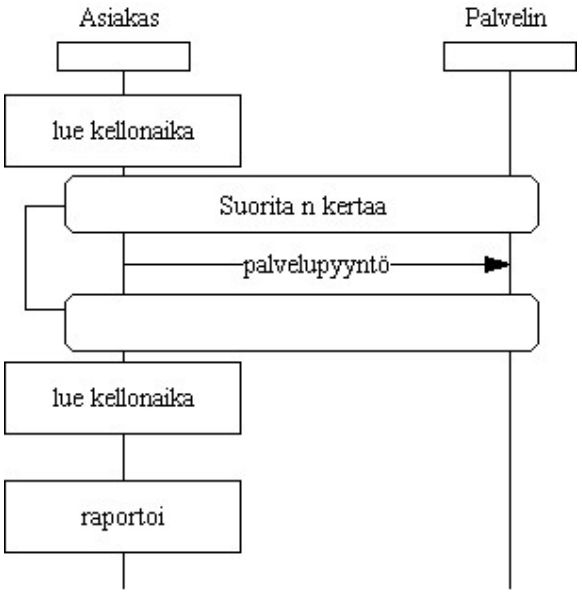
Palvelinsovellusten toteuttamassa IDL-rajapinnassa määritellään yksi palvelupyyntö, `ping`. Asiakassovellus kutsuu palvelinsovellusta tällä metodilla, joka palauttaa kokonaisluvun. Rajapinta on kuvattu liitteessä 7.1.1.

4.3 Toiminnallinen kuvaus

Tässä luvussa kuvataan koejärjestelmän toiminnallisuus, johon toteutus perustuu. Toiminnallisuuden lisäksi järjestelmälle määritellään toteutusympäristö.

4.3.1 Asiakkaan ja palvelimen välinen kommunikaatio

Kuvan 16 viestikaaviossa on kuvattu asiakkaan ja palvelimen välinen kommunikaatio sekä siihen liittyvät toiminnot asiakas- ja palvelinoliossa. Asiakas paikallistaa palvelimen tiedostoon merkkijonomuotoon kirjoitetun olioviitteen, IORin, avulla. Ennen palvelupyyntöjen suoritusta asiakasolio tarkastaa kellonajan. Myös palvelupyyntöjen suorituksen jälkeen tarkistetaan kellonaika ja lasketaan palvelupyyntöihin kulunut aika. Tämän jälkeen palvelupyynnöistä raportoidaan lokitiedostoon. Palvelinoliossa toteutetussa `ping`-operaatiossa voidaan palauttaa mikä tahansa kokonaisluku.



Kuva 16. Kommunikointi koejärjestelmässä.

Asiakkaan ja palvelimen välisessä kommunikaatiossa lähetettyjen GIOP- ja GTP-viestien lukumäärä on huomattavasti suurempi, kun kutsutaan palvelinta terminaalilta. Mikäli asiakas sijaitsee kiinteässä verkossa, riittää pyynnön ja vasteen kuljettamiseen kaksi GIOP/IIOP-viestiä (`Request` ja `Reply`).

GIOP-tunnelia käytettäessä on yhteys syytä jakaa kolmeen osaan [ks. kuva 19]: palvelimen ja yhdyssillan välinen yhteys, GIOP-tunneli sekä terminaalisillan ja asiakkaan välinen yhteys. Tällöin jo pelkkiä GIOP-viestejä tulee yhtä pyyntöä kohden kaksi ja vastaavasti myös vastetta kohden kaksi (yhdyssillan ja palvelimen välinen yhteys sekä terminaalisillan

ja asiakkaan välinen yhteys). Tämän lisäksi tarvitaan vielä GTP-sanomia GIOP-sanomien kuljettamiseen tunnelissa. GTP-sanomia tarvitaan kaksi pyyntöä ja kaksi vastetta kohti (`GIOPData` ja `GIOPDataReply`). Näin yhden palvelupyynnön ja siihen liittyvän vasteen kuljetukseen tarvitaan kaksi viestiä ilman tunnelointia ja kahdeksan (eli nelinkertainen määrä) viestiä käytettäessä GIOP-tunnelia [ks. kuva 10]. Tämän perusteella voidaan arvioida sanomien lähetykseen, kuljetukseen ja vastaanottamiseen kuluvan viiveen olevan langattomassa ympäristössä nelinkertainen verrattuna kiinteässä verkossa tapahtuviin palvelupyyntöihin. Kokonaisviivettä kasvattaa jonkin verran myös yhdyssillassa ja terminaalisillassa tapahtuva prosessointi. Lisäksi tunneloidut GIOP-sanomat ovat kooltaan GIOP/IIOP-sanomia suurempia, joka saattaa näkyä verkosta aiheutuvana viiveenä. Esimerkkisovellus testataan ruuhkattomassa verkossa.

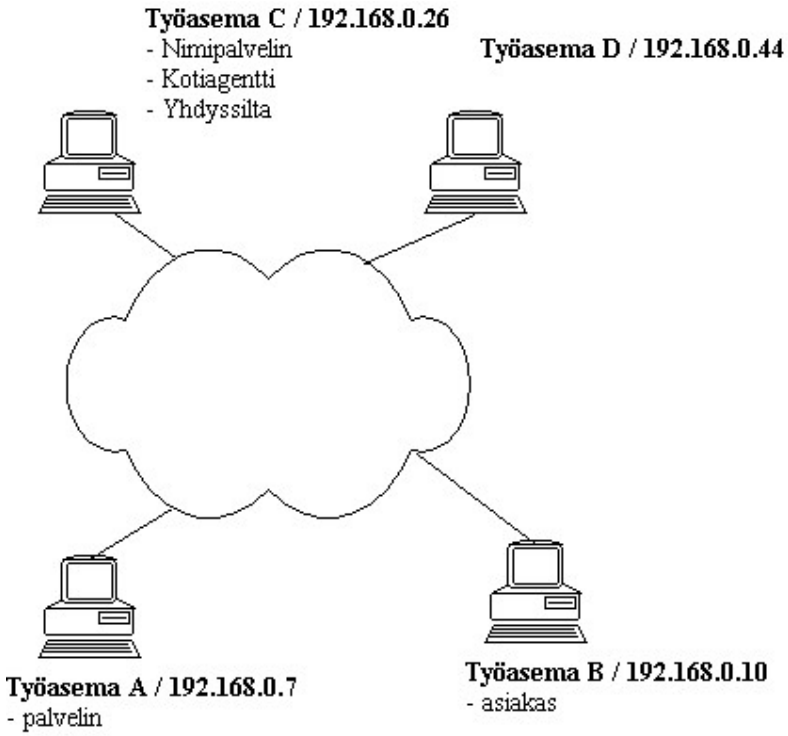
Mikäli tunnelia ei ole muodostettu, tarvitaan tähän kaksi GTP-sanomaa lisää (`EstablishTunnelRequest` ja `EstablishTunnelReply`) [ks. taulukko 2]. Esimerkkisovelluksessa GIOP-viestien tunnelointi perustuu TCP:hen. Tunnelissa ei kuitenkaan tule muodostaa TCP-yhteyttä TCP:n kättelymenettelyn [S1] mukaisesti.

4.3.2 Toteutus- ja testausympäristö

Järjestelmä toteutetaan ja testataan PC-työasemilla, joissa on Microsoft Windows NT tai Microsoft Windows 2000 käyttöjärjestelmä. Toteutusalustan valinnasta johtuen MIWCO käännetään Microsoft Visual C++ -tuotteen *nmake*–työkalulla. MIWCOn käytöstä johtuen asiakas ja palvelin toteutetaan C++ -ohjelmointikielellä. Myös järjestelmän asiakas ja palvelin käännetään nmake-työkalulla. Järjestelmän lähdetiedostot, IDL-tiedostot sekä suoritusskriptit löytyvät liitteistä.

Toteutuksessa ja testauksessa ei käytetä langatonta tiedonsiirtoa eikä siten myöskään langattomia päätelaitteita. Langatonta päätelaitetta simuloidaan tavallisella työasemalla. Langattoman verkon toimintaa simuloidaan TCP/IP-verkossa MIWCOn kotiagentti-, yhdyssilta- ja terminaalisiltasovelluksilla. Näin järjestelmä käyttää langattoman standardin mukaista GIOP-tunnelia [ks. kuva 15] kommunikoinnissa langatonta päätelaitetta simuloivan työaseman kanssa. Tunneloinnissa käytetään TCP-tunnelointia [ks. luku 3.3].

Sanomien kokoa ja kulkua verkossa tutkitaan *Anasil 2.2* -verkkoanalysointiohjelmalla. Testausympäristö on esitetty kuvassa 17.



Kuva 17. Testausympäristö.

4.4 Testaus

Testaus suoritetaan luvussa 4.3.2 määritellyssä ympäristössä. Testauksen tarkoituksena on osoittaa langattoman CORBAn toimivuus toteutusympäristössä sekä tarkastella langattoman ympäristön aiheuttamaa viivettä vasteajoissa. Suorituskykyyn liittyvät tulokset ovat vain suuntaa antavia, sillä vasteaikaan vaikuttavat todellisessa ympäristössä testiympäristöä enemmän verkon kuormitus, CORBA-toteutuksen valinta ja ennen kaikkea langattoman päätelaitteen suorituskyky. Tässä halutaan kuitenkin keskittyä CORBA-toteutuksen ja ennen kaikkea GIOP-tunnelin testaamiseen, joten verkolle ei testauksen aikana aiheuteta suurta ylimääräistä kuormaa.

4.4.1 Testitapaukset

Tässä luvussa määritellään testitapaukset suorituskyvyn selvittämiseksi. Taulukossa 4 on esitetty testikonfiguraatiot. Konfiguraatioissa viitataan kuvan 17 testausympäristöön.

Nro	Työasema A	Työasema B	Työasema C	Työasema D
1	Palvelin	Asiakas	-	-
2	Palvelin	Asiakas, terminaalisilta	Nimipalvelu, kotiagentti, yhdyssilta	-
3	Palvelin	Asiakas	Nimipalvelu, kotiagentti, yhdyssilta	Terminaalisilta

Taulukko 4. Testikonfiguraatiot.

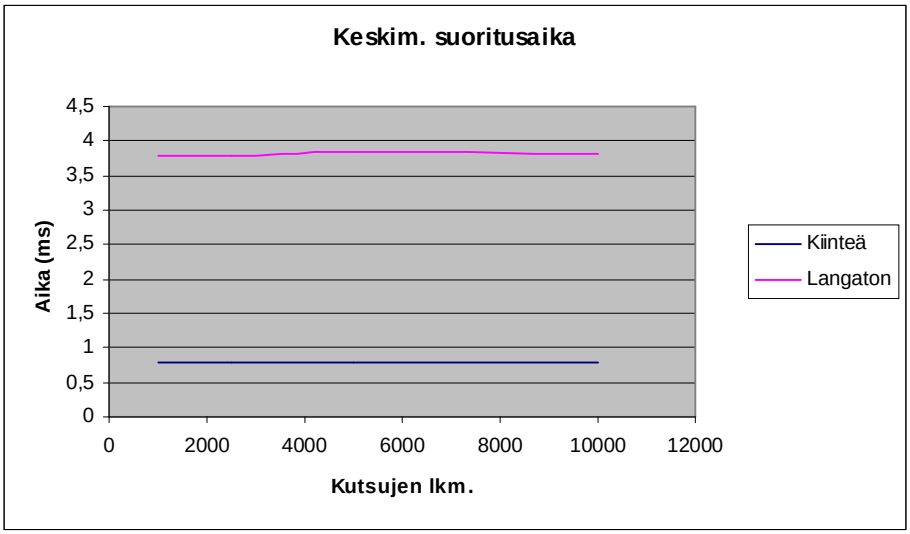
Testitapaukset on kuvattu taulukossa 5.

Nro	Konfiguraatio	Kutsuja	Kuvaus
1	1	1000	Selvitetään kutsuihin käytetty suoritusaika.
2	1	2500	Selvitetään kutsuihin käytetty suoritusaika.
3	1	5000	Selvitetään kutsuihin käytetty suoritusaika.
4	1	10000	Selvitetään kutsuihin käytetty suoritusaika.
5	2	1000	Selvitetään kutsuihin käytetty suoritusaika.
6	2	2500	Selvitetään kutsuihin käytetty suoritusaika.
7	2	5000	Selvitetään kutsuihin käytetty suoritusaika.
8	2	10000	Selvitetään kutsuihin käytetty suoritusaika.
9	3	1	Tutkitaan verkkoanalysaattorilla lähetettyjä sanomia ja tunnelinmuodostusta.

Taulukko 5. Testitapaukset.

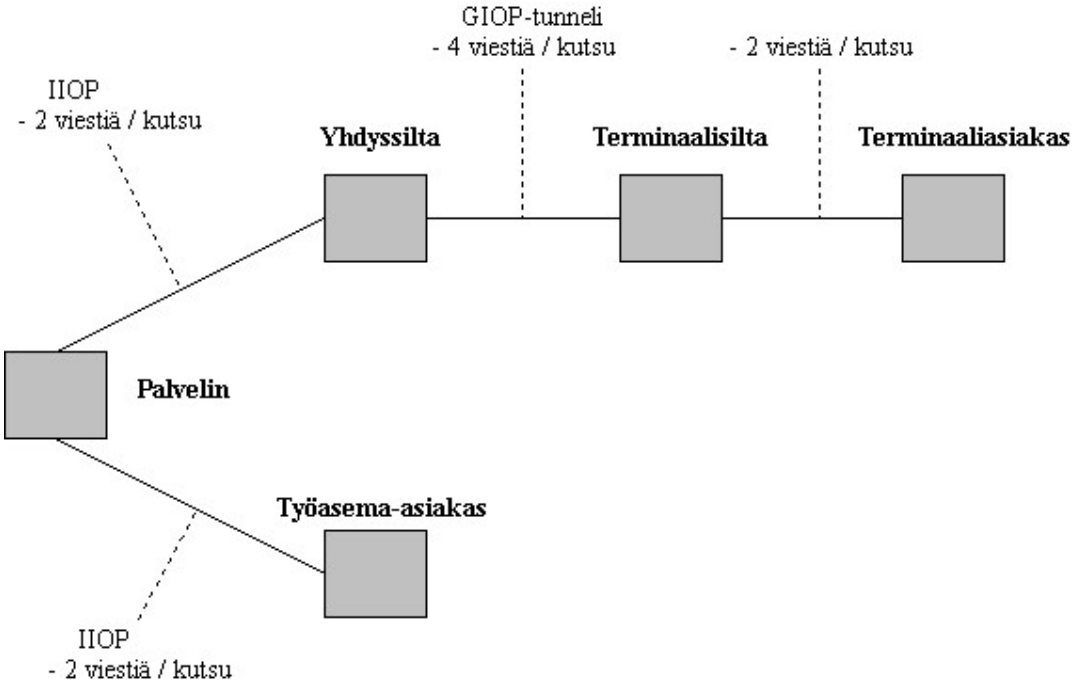
4.4.2 Testitulokset

Testitulokset osoittivat GIOP-tunneloinnin kasvattavan jonkin verran vasteaikaa palvelupyynnöissä. Kuvan 18 kaaviossa on esitetty testitapauksista 1-8 selvitetyt kutsuihin keskimäärin käytetyt ajat. Muutos keskimääräisissä suoritusajoissa oli erittäin vähäistä johtuen verkon vähäisestä käyttöasteesta. Käytetyllä CORBA-toteutuksella (MIWCO) viive kasvaa yli nelinkertaiseksi käytettäessä GIOP-tunnelia. Missään nimessä tunnelointi ei tule lyhentämään vasteaikoja, joten langattomia järjestelmiä suunniteltaessa on entistä enemmän kiinnitettävä huomiota järjestelmän arkkitehtuurin suunnitteluun.



Kuva 18: Kutsujen keskimääräinen suoritusaika.

Terminaaliasiakkaan viiveelle on helppo löytää selitys tarkastelemalla GIOP-tunnelointia [ks. luku 3.3] ja viive vastaakin luvussa 4.3.1 esitettyä arviota. GIOP-tunneloinnista aiheutuva nelinkertainen sanomien määrä näkyy suoraan vasteajassa. Verkkoanalysaattorilla suoritetut mittaukset vahvistivat sanomien lukumäärän nelinkertaisen kasvun palvelupyyntöä kohden GIOP-tunnelointia käytettäessä. Kuvassa 19 on esitetty esimerkkisovelluksen arkkitehtuuri viiveen muodostuksen kannalta. Kuvassa on esitetty GIOP-viestien lukumäärä yhtä pyyntöä ja siihen liittyvä vastetta kohden.



Kuva 19. Viiveen muodostuminen esimerkkisovelluksessa.

Voidaan siis ajatella, että langattoman päätelaitteen ja TCP/IP-verkossa sijaitsevan palvelimen kommunikoinnissa viive yhdyssillan ja palvelimen välillä on samaa luokkaa kuin TCP/IP-verkossa sijaitsevan asiakkaan ja palvelimen välisessä kommunikaatiossa. Testitulosten perusteella noin kolme neljäsosaa langattoman päätelaitteen ja palvelimen välisen kommunikaation viiveestä syntyy yhdyssillan ja päätelaitteen asiakassovelluksen välillä.

Verkkoanalysointiohjelmalla suoritettujen mittausten perusteella yhdyssillassa ja terminaalisillassa suoritettu sanomien prosessointiin (eli tunnelointi ja sen purku) käytetty aika oli merkityksetön (alle 0,001 ms). Myöskään sanomien koon kasvu GIOP-tunneloinnissa ei vaikuttanut merkittävästi vasteaikoihin. Näin ollen viiveen kasvu aiheutuu pääasiassa verkossa kuljetettavien sanomien lukumäärän kasvusta.

Tutkielman esimerkkisovelluksen GIOP-tunnelia testattaessa tunneli oli luotu jo ennen asiakkaan suorittamia palvelupyyntöjä. Käytännössä testauksessa todettuun viiveeseen tulisi lisätä myös mahdollisten tunnelien muodostusten sekä yhdyssiltojen vaihtojen eli kanavanvaihtojen aiheuttamat viiveet, käytettävän tunnelointi- ja kuljetusprotokollan viiveet sekä CORBA-toteutuksen aiheuttama viive. Käytettävä tunnelointi- ja kuljetusprotokolla ovat myös oleellisessa asemassa viivettä arvioitaessa. Esimerkissä käytettiin TCP-tunnelointia. Oma lukunsa ovat lisäksi langattomien laitteiden suorituskyky sekä siirtotie.

Verkkoanalysaattorilla tutkittaessa tunnelinmuodostukseen käytettiin MIWCOssa 23 TCP-sanomaa. Tähän sisältyi TCP-yhteyksien muodostukset sekä kommunikointi nimipalvelimen ja kotiagentin kanssa. Aikaa tunnelinmuodostukseen kului keskimäärin 0,18 ms.

GIOP-tunnelointi vaikuttaa aluksi raskaalta varsinkin TCP/IP-verkossa käytettynä. Käytännössä protokollan raskaus on hinta, joka on maksettava erilaisten verkkojen yhdistämisestä. Protokollan raskaus johtuu myös osaksi GIOPista, joka alun perin suunniteltiin luotettavan kuljetuskerroksen päälle. Siirrettäessä GIOP-viestejä verkossa, jossa ei taata luotettavaa kuljetusta, täytyy luotettavuus rakentaa GIOP-tunneloinnin yhteydessä. Ilman GIOPiin liitettyjä vahvoja kuljetuskerrokseen liittyviä oletuksia olisi siirto langattomaan ympäristöön ollut huomattavasti kätevämpää.

Vertailun vuoksi toteutettuun esimerkkisovellukseen mitattiin myös ICMP:hen (*Internet Control Message Protocol*) perustuvan ping-sovelluksen suoritusaika. Pingattaessa työasemalta B työasemaa A oli vasteaika alle millisekunnin. ICMP-sanomien koko oli vain puolet työasema-asiakkaan palvelupyynnön sisältämän sanoman koosta.

5 Yhteenveto

Tulevaisuudessa langattomien päätelaitteiden yleistyessä ja kehittyessä myös niiden sovellukset tulevat monipuolistumaan. CORBA-standardilla tullee olemaan merkittävä rooli langattomien sovellusten kehityksessä. Tällöin voidaan tuottaa yleisen standardin mukaisia tehokkaita monikerrosarkkitehtuurin perustuvia järjestelmiä, joissa liiketoimintalogiikka voidaan sijoittaa sinne, missä suoritustehoa on riittävästi käytössä. Standardin lopullisen hyvyyden määrittelee kuitenkin sen toteutukset.

CORBAssa oliovälittimien välinen kommunikointi tapahtuu GIOP:lla, joka suunniteltiin toimimaan minkä tahansa luotettavan kuljetuskerroksen päälle. 90-luvun alussa tätä pidettiin riittävän joustavana. Tänä päivänä nämä kuljetusprotokollaan kohdistuvat oletukset eivät päde siirrettäessä GIOP-viestejä verkossa, jossa ei luotettavaa kuljetusta tarjota. Tämän vuoksi CORBAn soveltaminen langattomaan ympäristöön vaati uusien protokollien määrittelyn. Näitä protokollia käytetään GIOP-viestien tunneloimiseen ja kuljetukseen epäluotettavan kuljetusyhteyden yli. Mikäli GIOPin suunnittelu ei olisi perustunut luotettavaan kuljetukseen, olisi CORBAn käyttö langattomassa ympäristössä ollut huomattavasti yksinkertaisempaa.

Tämän tutkielman yhteydessä toteutetulla koejärjestelmällä sekä sen testauksella osoitettiin langattoman CORBA-laajennuksen toimivuus. Testituloksia luettaessa ja tulkittaessa on syytä huomioida, että OMG:n langattoman CORBAn spesifikaatio ei tutkielman tekohetkellä ollut vielä julkinen eikä siitä siten ole vielä toteutuksia kaupallisissa CORBA-tuotteissa. Lisäksi on syytä huomioida langattoman järjestelmän suorituskykyyn liittyvän useita muita, tässä tutkielmassa vähemmälle huomiolle jääneitä tekijöitä. Näitä ovat mm. verkon infrastruktuuri sekä verkon ja yhdyssiltojen käyttöaste.

Testitulokset osoittavat tunneloinnin lisäävän vasteaikaa. Vasteajan kasvuun vaikutti eniten GIOP-tunnelin käytöstä johtuva verkossa kuljetettavien sanomien lukumäärän nelinkertaistuminen. Vasteajan merkitys korostuu sovelluksissa, joissa verkon yli suoritettavia kutsuja tehdään useita. Tällöin on kuitenkin syytä tarkastella sovelluksen arkkitehtuuria tarkemmin ja sitä kehittämällä pyrkiä minimoimaan verkon yli suoritettavien kutsujen lukumäärä. Toisaalta liian suurien viestien lähettämistä tulisi myös

välttää, joten kommunikaatioon on löydettävä kultainen keskitie. Vasteaikojen kasvaessa lisääntyy myös suunnittelun merkitys. Tärkeä seikka on pitää asiakas mahdollisimman pienenä ja yksinkertaisena. Palvelimen toteuttaminen liikkuvalle päätelaitteelle lienee vielä harvinaisempaa. Palvelinsovelluksen suorittaminen langattomalla päätelaitteella voisi olla perusteltua esimerkiksi kulkuneuvoihin liitettävissä ohjelmistoissa. Tämän vuoksi standardin on toimittava hyvin myös tilanteissa, joissa asiakas sijaitsee kiinteässä verkossa (tai langattomalla päätelaitteella) ja palvelin langattomalla päätelaitteella.

Koska MIWCO oli toteutettu Linux-käyttöjärjestelmässä, tuotti toteutusympäristön rakentaminen Microsoft Windows –käyttöjärjestelmän ja nmaken pohjalle joitain ongelmia. Tämän tutkielman yleishyödyllisenä tuotoksena voidaan pitää MIWCOn koodin muokkaamista nmake-yhteensopivaksi.

Kun langattoman CORBAn spesifikaatio dtc/01-06-02 saadaan viimeisteltyä, voidaan siitä odottaa toteutuksia myös kaupallisiin CORBA-ohjelmistoihin. Palveluiden tarpeen ollessa vasta kartoituksen alla, ensimmäisiin langattoman CORBAn toteutuksiin saatetaan liittää toteutuskohtaisia palveluita ja ratkaisuja. Nämä on kuitenkin tehtävä siten, että eri valmistajien oliovälittimien välinen kommunikointi on mahdollista. Tätä edellytetään myös CORBA-standardissa.

Langattoman CORBAn määritelmä tulee laajentumaan palvelumäärittelyjen osalta lähitulevaisuudessa. Tämä määrittelytyö on vasta alussa ja sen etenemistä voi seurata OMG:n kotisivuilta [`http://www.omg.org`]. Uusia haasteita ovat myös reaaliaikajärjestelmien [ks. luku 2.8] sekä vikasietoisille järjestelmille suunnitellun CORBAn [ks. luku 2.9] liittäminen langattoman CORBAan.

6 Lähteet

[IETF1] Yeong, Howes, Kille, Lightweight Directory Access Protocol, IETF, 1995.

[K1] Kangasharju, MIWCO – Wireless CORBA Extension to MICO, University of Helsinki, 2001.

[M1] MICO, MICO is CORBA, version 2.3.6, saatavilla ps-muodossa osoitteesta [`http://www.mico.org`].

[N1] Novell, NDS Technical Overview, saatavissa HTML-muodossa osoitteesta [`http://developer.novell.com/ndk/doc/ndslib/`].

[O1] Orfali Robert, Harkey Dan, Client Server Programming with Java and CORBA, John Wiley & Sons, Inc., 1998.

[OMG1] OMG, The Common Object Request Broker: Architecture abd Specification (Revision 2.5), OMG, 2001.

[OMG2] OMG, CORBAservices: Common Object Services Specification, OMG, 1999.

[OMG3] OMG, dtc/01-06-02 - Wireless Access and Terminal Mobility in CORBA, OMG, 2001.

[OMG4] OMG, Request for Proposal – Wireless Access and Terminal Mobility, OMG, 1999.

[OMG5] OMG, telecom/2001-07-03 – Request For Information CORBA Mobility Services. OMG, 2001.

[OSF1] Opengroup, OSF DCE, [`http://www.opengroup.org/dce/`].

[S1] Stallings, Data and Computer Communications (6th edition), Prentice-Hall, 2000.

[SA1] Salminen, Tietoturva hajautetuissa järjestelmissä, Jyväskylän yliopisto, 2001.

[SU1] Surfnet, X.500 Directory Service, Surfnet 1997, saatavissa HTML-muodossa osoitteessa [http://www.surfnet.nl/innovatie/afgesloten/x500/].

[TC] Telecommunications International, ”CORBA: The Route to Interoperability”, Volume 35, No. 4. 2001.

[W1] WAP Forum, Wireless Application Protocol White Paper, WAP Forum 2000.

[W2] WAP Forum, WAP 2.0 Technical White Paper, WAP Forum 2001.

[CORBA] www.corba.com

7 Liitteet

7.1 Esimerkkisovelluksen lähdekoodit

7.1.1 WirelessTest-rajapinta

```
/**
* Pro Gradu-tutkielma
* "CORBA langattomassa ympäristössä"
* Timo Salminen
*
*
* Koejärjestelmän IDL-rajapinta
* wirelessTest.idl
*
*/
interface WirelessTest
{
    long ping();
};
```

7.1.2 Palvelinolio

```
/**
* Pro Gradu-tutkielma
* "CORBA langattomassa ympäristössä"
* Timo Salminen
* 29.11.2001
*
* Koejärjestelmän palvelinluokka
* server.cc
*/

#include <fstream.h>
#include <mico/CosNaming.h>
#include "wirelessTest.h"

class WirelessTest_impl : virtual public POA_WirelessTest
{
public:
  long ping ();
};

long WirelessTest_impl::ping ()
{
  return 0;
}

int main (int argc, char *argv[])
{
  /*
   * Alustetaan ORB.
   */
```

```
  CORBA::ORB_var orb = CORBA::ORB_init (argc, argv);

  /*
   * Hankitaan viite oliosovittimeen RootPOA ja sen Manager-olioon.
   */
  CORBA::Object_var poaobj = orb->resolve_initial_references ("RootPOA");
  PortableServer::POA_var poa = PortableServer::POA::_narrow (poaobj);
  PortableServer::POAManager_var mgr = poa->the_POAManager();

  /*
   * Luodaan palvelinolio
   */
  WirelessTest_impl * tServer = new WirelessTest_impl;

  /*
   * Aktivoidaan servant-olio.
   */
  PortableServer::ObjectId_var oid = poa->activate_object (tServer);
  CORBA::Object_var ref = poa->id_to_reference (oid.in());

  /*
   * Kirjoitetaan viite tiedostoon.
   */
  cout << "Talletetaan IOR tiedostoon...";
  ofstream of ("wTest.ref");
  CORBA::Object_var ref1 = poa->id_to_reference (oid.in());
  CORBA::String_var str1 = orb->object_to_string (ref1.in());
  of << str1.in() << endl;
  of.close ();
  cout << "Ok\n";


  /*
   * Aktivoidaan POA-oliosovitin ja käynnistetään palvelin.
   */
  printf ("Running.\n");
  mgr->activate ();
  orb->run();

  /*
   * Palvelimen alasajo (tänne ei koskaan päästä).
   */
  poa->destroy (TRUE, TRUE);
  delete tServer;
  return 0;
}
```

7.1.3 Asiakasolio

```
/**
* Pro Gradu-tutkielma
* "CORBA langattomassa ympäristössä"
* Timo Salminen
* 29.11.2001
*
* Koejärjestelmän asiakasluokka
```

```
* client.cc
*/

#include <fstream.h>
#include <time.h>
#include <mico/CosNaming.h>
#include "WirelessTest.h"
#define N_OF_CALLS 1000


int main (int argc, char *argv[])
{
  char uri[1100];
  double totalTime;
  clock_t begin, end;
  CORBA::ORB_var orb = CORBA::ORB_init (argc, argv);

  /*
   * IOR on tallennettu wTest.ref -tiedostoon paikallisessa hakemistossa
   */
  cout << "Avataan IOR-tiedostoa...";
  ifstream fi("wTest.ref");
  if (!fi)
  {
    cout << "IOR-tiedosto ei aukea.\n";
           return 1;
  }
  cout << "luku...";

  fi.getline(uri, 1100);
  fi.close();
  cout << "Ok!" << endl;
  cout << uri << endl;
  //Sidotaan IOR palvelimeen

  CORBA::Object_var obj = orb->string_to_object (uri);
  WirelessTest_var server = WirelessTest::_narrow (obj);

  if (CORBA::is_nil (server)) {
    printf ("Palvelinta ei paikallistettu.\n");
    return 1;
  }

  cout << "Kutsutaan palvelinta...";
  begin = clock();
  for (int i=0; i<N_OF_CALLS; i++)
  {
     server->ping ();
  }
  end = clock();
  totalTime = double(end-begin)/CLOCKS_PER_SEC;

  cout << "Lasketaan silmukan suoritukseen kulunut aika..." << endl;
  begin = clock();
  for (i=0; i<N_OF_CALLS; i++)
  {
```

```
  }
  end = clock();
  totalTime = totalTime - (double(end-begin)/CLOCKS_PER_SEC);
  cout << "Ok.\n";

  /*
   * Kirjoitetaan tulokset results.txt -tiedostoon.
   */
  ofstream of ("results.txt");
  of << "Kutsuja\t" << "Aika yht.\t" << "Keskim. kutsuaika" <<endl;
  of << N_OF_CALLS << "\t";
  of << totalTime << " s\t";
  of << totalTime/N_OF_CALLS << " s" <<endl;
  of.close ();

  return 0;
}
```

7.1.4 IDL:stä generoitu WirelessTest.h

```
/*
 *  MICO --- an Open Source CORBA implementation
 *  Copyright (c) 1997-2001 by The Mico Team
 *
 *  This file was automatically generated. DO NOT EDIT!
 */
#include <CORBA.h>
#include <mico/throw.h>

#ifndef __WIRELESSTEST_H__
#define __WIRELESSTEST_H__

class WirelessTest;
typedef WirelessTest *WirelessTest_ptr;
typedef WirelessTest_ptr WirelessTestRef;
typedef ObjVar<WirelessTest> WirelessTest_var;
typedef ObjOut<WirelessTest> WirelessTest_out;


/*
 * Base class and common definitions for interface WirelessTest
 */

class WirelessTest :
  virtual public CORBA::Object
{
  public:
    virtual ~WirelessTest();

    #ifdef HAVE_TYPEDEF_OVERLOAD
    typedef WirelessTest_ptr _ptr_type;
    typedef WirelessTest_var _var_type;
    #endif

    static WirelessTest_ptr _narrow( CORBA::Object_ptr obj );
    static WirelessTest_ptr _narrow( CORBA::AbstractBase_ptr obj );
```

```
    static WirelessTest_ptr _duplicate( WirelessTest_ptr _obj )
    {
      CORBA::Object::_duplicate (_obj);
      return _obj;
    }

    static WirelessTest_ptr _nil()
    {
      return 0;
    }

    virtual void *_narrow_helper( const char *repoid );

    virtual CORBA::Long ping() = 0;

  protected:
    WirelessTest() {};
  private:
    WirelessTest( const WirelessTest& );
    void operator=( const WirelessTest& );
};

// Stub for interface WirelessTest
class WirelessTest_stub:
  virtual public WirelessTest
{
  public:
    virtual ~WirelessTest_stub();
    CORBA::Long ping();

  private:
    void operator=( const WirelessTest_stub& );
};

#ifndef MICO_CONF_NO_POA

class WirelessTest_stub_clp :
  virtual public WirelessTest_stub,
  virtual public PortableServer::StubBase
{
  public:
    WirelessTest_stub_clp (PortableServer::POA_ptr, CORBA::Object_ptr);
    virtual ~WirelessTest_stub_clp ();
    CORBA::Long ping();

  protected:
    WirelessTest_stub_clp ();
  private:
    void operator=( const WirelessTest_stub_clp & );
};

#endif // MICO_CONF_NO_POA

#ifndef MICO_CONF_NO_POA
```

```
class         POA_WirelessTest         :         virtual         public
PortableServer::StaticImplementation
{
  public:
    virtual ~POA_WirelessTest ();
    WirelessTest_ptr _this ();
    bool dispatch (CORBA::StaticServerRequest_ptr);
    virtual void invoke (CORBA::StaticServerRequest_ptr);
    virtual CORBA::Boolean _is_a (const char *);
    virtual CORBA::InterfaceDef_ptr _get_interface ();
    virtual      CORBA::RepositoryId      _primary_interface      (const
PortableServer::ObjectId &, PortableServer::POA_ptr);

    virtual void * _narrow_helper (const char *);
    static POA_WirelessTest * _narrow (PortableServer::Servant);
    virtual   CORBA::Object_ptr   _make_stub   (PortableServer::POA_ptr,
CORBA::Object_ptr);

    virtual CORBA::Long ping() = 0;

  protected:
    POA_WirelessTest () {};

  private:
    POA_WirelessTest (const POA_WirelessTest &);
    void operator= (const POA_WirelessTest &);
};
#endif // MICO_CONF_NO_POA
extern CORBA::StaticTypeInfo *_marshaller_WirelessTest;
#endif
```

7.2 Esimerkkisovelluksen käynnistystiedostot

Käynnistystiedostoissa annetaan CORBA-sovellukselle tarvittavat parametrit. Kaikille soivelluksille yhteiset parametrit on määritelty tiedostossa micorc.cfg. Kotiagentin käynnistysparametrit on määritelty tiedostossa hlarc.cfg.

7.2.1 micorc.cfg

```
-ORBInitRef NameService=corbaloc::1.2@tyoasemac:16000/NameService
-ORBInitRef HomeLocationAgent =
IOR:010000003100000049444c3a6f6d672e6f72672f4d6f62696c655465726d696e616c2
f486f6d654c6f636174696f6e4167656e743a312e300000000002000000000000000400000
00010102000a0000003132372e302e302e3100e06a210000004d6f62696c6974795375707
06f72742f486f6d654c6f636174696f6e4167656e7400000000000000010000002400000000
0100000001000000010000001400000001000000010001000000000009010100000000000
-ORBGIOPVersion 1.2
-ORBIIOPVersion 1.2
```

7.2.2 hlarc.cfg

Tässä tiedostossa määritellään kotiagentin terminaalien nimet. Terminaalin tunniste muodostetaan nimen ja tunnisteosan yhdistelmästä. Tiedoston sisältö on koejärjestelmässä seuraavanlainen:

```
ristikivi
```

7.2.3 Nimipalvelin

```
nsd -ORBConfFile c:\miwcocfg\micorc.cfg -ORBIIOPAddr inet:tyoasemac:16000
```

7.2.4 Kotiagentti

```
hla -ORBConfFile c:\miwcocfg\micorc.cfg -ORBIIOPAddr inet:tyoasemac:27360
-POAImplName MobilitySupport -WATMTerminalPrefix %80%d6%0a%a6
```

7.2.5 Kuokkala (yhdyssilta)

```
ab -ORBNoIIOPServer -ORBConfFile c:\miwcocfg\micorc.cfg -WATMGIOPAddr
inet:tyoasemad:41514 -WATMGTPAddr inet:tyoasemad:21101 -WATMBridgeName
KuokkalaBridge -POAImplName MobilitySupport
```

7.2.6 Ristikivi (terminaalisilta)

```
tb -ORBNoIIOPServer -ORBConfFile c:\miwcocfg\micorc.cfg -WATMGIOPAddr
inet:tyoasemab:12022 -WATMGTPAddr inet:tyoasemab:21101 -WATMBridgeName
RistikiviBridge -POAImplName MobilitySupport -ORBTerminalId
%%04%%80%%d6%%0a%%a6%%08ristikivi
```

7.2.7 Työasemapalvelin

```
server -ORBConfFile c:\miwcocfg\micorc.cfg -POAImplName WirelessTest
```

7.2.8 Työasema-asiakas

```
client -ORBConfFile c:\miwcocfg\micorc.cfg
```

7.2.9 Langaton asiakas

```
client -ORBNoIIOPServer -ORBConfFile c:\miwcocfg\micorc.cfg -
ORBTerminalId %%04%%80%%d6%%0a%%a6%%08ristikivi -ORBMTBAddr
inet:tyoasemab:12022
```

7.3 Testitulokset

Tässä luvussa esitellään tulostiedostot sekä testitulokset hieman yksityiskohtaisemmin.

7.3.1 Tulostiedosto – results.txt

```
Kutsuja    Aika yht.  Keskim. kutsuaika

1000       2.984 s    0.002984 s
```

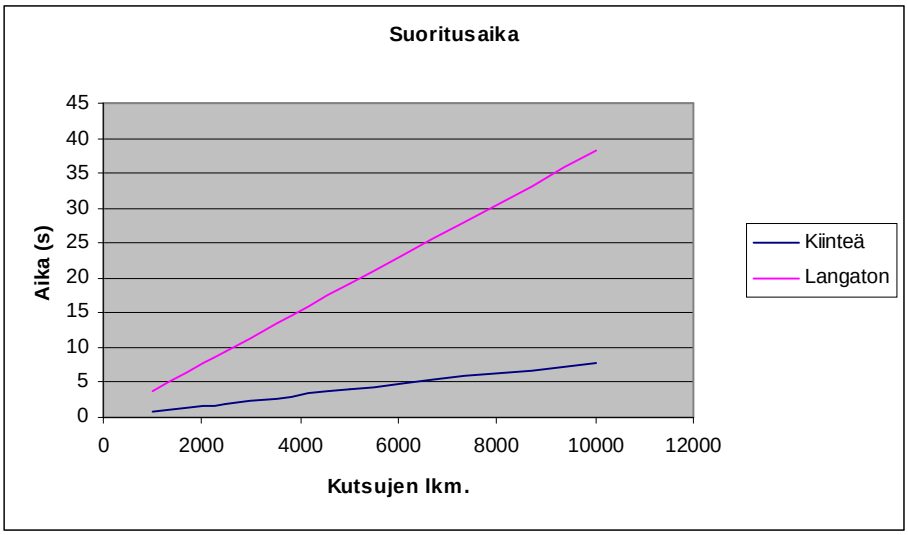
7.3.2 Tulokset

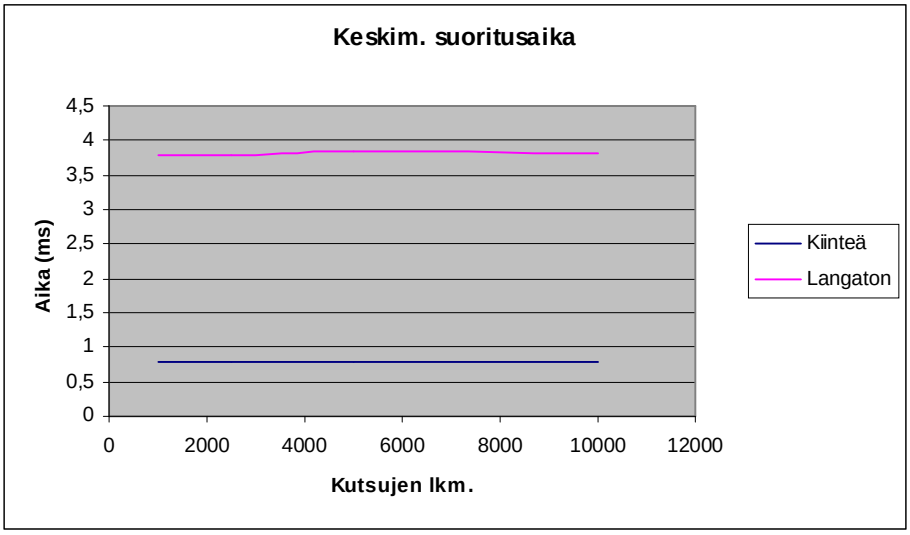
Työasema-asiakkaan tulokset:

Kutsuja	Keskim. kutsuaika(ms)	Aika yht. (s)
1000	0,791	0,791
2500	0,7808	1,952
5000	0,783	3,915
10000	0,7781	7,781

Terminaali-asiakkaan tulokset:

Kutsuja	Keskim. kutsuaika(ms)	Aika yht. (s)
1000	3,785	3,785
2500	3,8012	9,503
5000	3,8334	19,167
10000	3,8135	38,135





7.4 Otteita verkkoanalysaattorimittauksista

7.4.1 Työasema-asiakkaan palvelupyyntö

```
-------------------------------------------------------------------------------
Source: 192.168.0.10       Destination: 192.168.0.7
Source MAC: 00-50-04-02-FE-5C           Destination MAC: 00-10-5A-E6-1A-4A
Source IP: 192.168.0.10    Destination IP: 192.168.0.7

Frame received at:  0.03 s
Frame size: 142 bytes


--- Layer: Ethernet type II

               Destination address: 00-10-5A-E6-1A-4A
               Source address:      00-50-04-02-FE-5C

               Type  IP

--- Layer: IP header

               Version            4  Header length   5
               TOS               10  Total length  128
               Ident.         21360  Flags           2
               Fragment offset    0  TTL           128
               Protocol  TCP - Transmission Control
               Checksum        2596

               Source IP Address   192.168.0.10
               Dest. IP Address     192.168.0.7

--- Layer: TCP header

               Source port  UNKNOWN (0x447)
               Dest.  port  UNKNOWN (0x5BA0)
               Seq. number  0000E529  Ack. number  0000C841

               Header len.      5  Reserved          0
```

```
               Urgent           0  ACK               1
               PSH              1  RST               0
               SYN              0  FIN               0

               Window size   8760
               TCP Checksum  1614  Urgent pointer     0

--- UNDECODED

               47 49 4F 50 01 02 01 00 4C 00 00 00 01 00 00 00   GIOP....L.......
               03 00 00 00 00 00 00 00 12 00 00 00 2F 32 39 34   ............/294
               2F 31 30 30 39 34 35 39 37 37 32 2F 5F 30 00 00   /1009459772/_0..
               05 00 00 00 70 69 6E 67 00 00 00 00 01 00 00 00   ....ping........
               01 00 00 00 0C 00 00 00 01 00 00 00 01 00 01 00   ................
               09 01 01 00 00 00 00 00                           ........

-------------------------------------------------------------------------------
Source: 192.168.0.7        Destination: 192.168.0.10
Source MAC: 00-10-5A-E6-1A-4A           Destination MAC: 00-50-04-02-FE-5C
Source IP: 192.168.0.7     Destination IP: 192.168.0.10

Frame received at:  0.03 s
Frame size: 82 bytes


--- Layer: Ethernet type II

               Destination address: 00-50-04-02-FE-5C
               Source address:      00-10-5A-E6-1A-4A

               Type  IP

--- Layer: IP header

               Version            4  Header length   5
               TOS               00  Total length   68
               Ident.         41463  Flags           2
               Fragment offset    0  TTL           128
               Protocol  TCP - Transmission Control
               Checksum        D75A

               Source IP Address    192.168.0.7
               Dest. IP Address    192.168.0.10

--- Layer: TCP header

               Source port  UNKNOWN (0x5BA0)
               Dest.  port  UNKNOWN (0x447)
               Seq. number  0000C841  Ack. number  0000E581

               Header len.      5  Reserved          0
               Urgent           0  ACK               1
               PSH              1  RST               0
               SYN              0  FIN               0

               Window size   8672
               TCP Checksum 21799  Urgent pointer     0

--- UNDECODED

               47 49 4F 50 01 02 01 01 10 00 00 00 01 00 00 00   GIOP............
               00 00 00 00 00 00 00 00 00 00 00 00               ............
```

7.4.2 Terminaaliasiakkaan palvelupyyntö

```
------------------------------------------------------------------------------
Source: 192.168.0.10       Destination: 192.168.0.26
Source MAC: 00-50-04-02-FE-5C           Destination MAC: 00-01-03-87-E8-F1
Source IP: 192.168.0.10    Destination IP: 192.168.0.26

Frame received at:  0.02 s
Frame size: 274 bytes


--- Layer: Ethernet type II

               Destination address: 00-01-03-87-E8-F1
               Source address:      00-50-04-02-FE-5C

               Type  IP

--- Layer: IP header

               Version            4  Header length   5
               TOS               10  Total length  260
               Ident.         33280  Flags           2
               Fragment offset    0  TTL           128
               Protocol  TCP - Transmission Control
               Checksum        F66E

               Source IP Address   192.168.0.10
               Dest. IP Address    192.168.0.26

--- Layer: TCP header

               Source port  UNKNOWN (0x468)
               Dest.  port  UNKNOWN (0x526D)
               Seq. number  0000E716  Ack. number  D7FE6FD4

               Header len.      5  Reserved          0
               Urgent           0  ACK               1
               PSH              1  RST               0
               SYN              0  FIN               0

               Window size   8548
               TCP Checksum 32994  Urgent pointer     0

--- UNDECODED

               0C 80 03 00 02 00 D4 00 02 00 00 00 05 00 00 00   .€....Ô.........
               C8 00 00 00 47 49 4F 50 01 02 01 00 BC 00 00 00   È...GIOP....¼...
               01 00 00 00 03 00 00 00 02 00 00 00 00 00 00 00   ................
               15 00 00 00 49 44 4C 3A 57 69 72 65 6C 65 73 73   ....IDL:Wireless
               54 65 73 74 3A 31 2E 30 00 00 00 00 02 00 00 00   Test:1.0........
               00 00 00 00 34 00 00 00 01 01 02 00 0C 00 00 00   ....4...........
               31 39 32 2E 31 36 38 2E 30 2E 37 00 A0 5B 00 00   192.168.0.7. [..
               12 00 00 00 2F 32 39 34 2F 31 30 30 39 34 35 39   ..../294/1009459
               37 37 32 2F 5F 30 00 00 00 00 00 00 01 00 00 00   772/_0..........
               24 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00   $...............
               14 00 00 00 01 00 00 00 01 00 01 00 00 00 00 00   ................
               09 01 01 00 00 00 00 00 05 00 00 00 70 69 6E 67   ............ping
               00 00 00 00 01 00 00 00 01 00 00 00 0C 00 00 00   ................
               01 00 00 00 01 00 01 00 09 01 01 00               ............
```