QuO

Tuomas Nurmela
Väliohjelmistot
Helsingin Yliopisto
Tietojenkäsittelytieteen laitos
27.05.2003
Vedos 0.6

Tiivistelmä – DARPAn Quorum-hanke on tuottanut monipuolista adaptaatiota tukevan, ORBin toimintaa laajentavan Quality Objects väliohjelmiston. Tämän puitteissa ei-toimminnalliset aspektit kuten reaaliaikaisuus, luotettavuus, tietoturva sekä selviytymiskyky kyetään erottamaan itse sovelluksen (liike)toiminnallisesta logiikasta. Quality Objects rakentuu QuO ytimestä, QDL kuvauskielistä sekä koodigeneraattoreista, jotka generoivat tarvittavat adaptaatiomekanismit sekä sovellukseen että QuO ytimeen. Näiden lisäksi QuO ydin edellyttää joukkoa järjestelmän infrastruktuurin resurssien tilaa monitoroivia ja hallinnoinnista vastaavia objekteja, jotta järjestelmä kokonaisuudessaan kykenee tuottamaan adaptaation.

Seminaarialustus kuvaa yleisellä tasolla palvelun laadun lähtökohdat, QuO:n ajoympäristön, arkkitehtuurin ja toteutuksen päätökset, yhteydet liittyviin standardeihin, määrityksiin ja teknologioihin sekä kartoittaa QuO:n tulevaisuutta.

Avainsanat: CORBA, QuO, palvelun laatu, adaptaatio, reflektiivisyys, luotettavuus, reaaliaikaisuus, selviytyvät sovellukset

1 Johdanto

Palvelun laatu (Quality of Service) on yksi *ei-toiminnallisista* (non-functional) järjestelmän aspekteista. Palvelun laatuvaatimusten tarkastelu voidaan jakaa sovellusnäkökulmaan ja resurssinäkökulmaan sekä näitä yhdistävään, päästä-päähän mittauksesta vastaavaan systeeminäkökulmaan [1]. Näkökulmien mittarit sekä toimintaa kuvaavat sopimukset on kyettävä limittämään, jotta sopimukset ovat merkityksen omaavia.

Sovellusnäkökulman (application viewpoint) vaatimukset, mittaus- ja hallinnointimekanismit liittyvät tyypillisesti läheisesti *sovellusalueeseen* (application domain). Reaaliaikaisuuden toteutuminen on yksi tyypillinen sovellusaluekohtaisesti toteutuva piirre. Reaaliaikaisuus voidaan korkealla tasolla jakaa kahteen luokkaan sovelluksen palveluvaatimuksen ehdottomuuden mukaan:

- *Absoluuttiset reaaliaikavaatumukset* (hard-real-time, stringent real-time) omaava sovellus edellyttää laatuvaatimusten toteutumista kyetäkseen toteuttamaan tehtävänsä. Tyypillisesti toiminta perustuu *aikapohjaisiin* operaatioihin (time-triggered approach) [2:2-15]. Esimerkkejä tällaisista järjestelmistä ovat ohjaus- ja hallintajärjestelmät. Armeijan toiminta-ympäristössä nämä kattavat mm. miehittämättömät lentokoneet sekä ohjusten ohjausjärjestelmät. Siviilijärjestelmät puolestaan sisältävät mm. ydinreaktorien hallintajärjestelmät, metalli-sulattamoiden ohjausjärjestelmät sekä lääkkeiden valmistusjärjestelmät.
- *Lähes reaaliaikavaatimukset* (soft real-time, real-timeliness) omaava sovellus on herkkä palvelu reaaliaikasuuteen liittyviin rikkeisiin, mutta kykenee toipumaan väliaikaisista palvelun laatuvaatimusten rikkeistä esimerkiksi puskuroimalla dataa. Tyypillisesti toiminta perustuu *tapahtumapohjaiseen* adaptiivisuuteen (event-triggered approach) [2:2-15]. Esimerkkejä tällaisista järjestelmistä ovat mm. osa multimedia-sovelluksista.

Sovellusnäkökulman laatuvaatimus ei välttämättä ole loppukäyttäjän näkökulmasta määriteltävissä suoraviivaisesti sovellukselle tyypillisin kriteerein, vaan voi edellyttää muunnosta käyttäjälähtöiseen benchmark-mittariin. Tästä huolimatta subjektiivisten laatukäsitysten ja lopullisten laatumittareiden suhde on vähintäänkin vaikeasti määriteltävissä, erityisesti mikäli sovellus koostuu osista, joilla on eroavia tarpeita saman aspektin puitteissa: esimerkiksi yhtäaikaista audiokommunikointia ja videokuvan välitystä tukeva sovellus edellyttää toistaalta äänelle pientä kaistanleveyttä sietämättä viivettä siinä missä kuvaa voidaan tarvittaessa muuttaa vaikka yksittäisiksi otoksiksi, joskin optimaalinen toiminta edellyttää merkittävästi ääntä suurempaa kaistanleveyttä.

Resurssinäkökulma (resource viewpoint) kuvastaa infrastruktuurin resurssien omaamia tiloja: toiminnan kohteina ovat resurssin mittareiden monitorointi sekä hallinnointi sovelluksen vaatiman sopimuksen mukaisesti. Tyypillisiä mekanismeja ovat käyttöjärjestelmän vuorontaminen, lukitusprimitiivit tai tietoliikenteen hallinnointimekanismit.

Systeeminäkökulman (system viewpoint) toteuttaminen edellyttää kokonaiskäsitystä, jota on kuitenkin vaikea toteuttaa sovellusriippumattomoilla järjestelmillä, koska sovellusalueiden vaatimukset ovat kovin erilaiset.

Ei-toiminnallisen luonteensa vuoksi systeeminäkökulmaa on yhtälailla vaikea toteuttaa systemaattisesti sovelluksessa, jonka useat eri toiminnot käsittelevät samaa asiaa. Vaikka päällekkäisyyttä ei olisikaan, sovelluskohtainen toteutus rajoittaa mekanismin uudelleenkäyttöä.

Systeeminäkökulman määrittämiseksi tyypillisesti tukeudutaan sopimuksiin, joiden puitteissa kuvataan asiakassovelluksen vaatimustasoja sekä näiden täyttymisestä vastaavia resursseja kussakin paikalliskontekstissa. Lisäksi, joskin harvemmissa tapauksissa, kuvataan myös sovelluksen käytössä olevat adaptaatiomekanismeja tuottavia rajapinnat (esimerkiksi vaihtoehtoiset kuvan näyttöresoluutiot).

DARPAN Quarum-hankkeen ydinprojekti Quality Objects (QuO) on tuottanut jo kuusi vuotta CORBAa laajentavaa QuO väliohjelmistoa, jonka lähtökohtana on erottaa ei-toiminnalliset aspektit sovelluskoodin toteutuksesta erillisiksi kokonaisuuksiksi tukeutumalla erillisiin kuvauskieliin. QuO väliohjelmiston ensimmäiset kuvaukset julkaistiin kesällä 1997 [3]. QuO:n väliohjelmiston julkaistiin syyskuussa 1998, toinen versio 1999 toukokuussa. Nykyinen QuO ajoympäristön versio 3.0, julkaistiin 2001 [14].

Muut Quorum -hankkeen projektit tuottavat QuO:n liitettäviä lisäpalveluja sekä laajennuksia. Jaoittelu tuottaa luonnollisen järjestelmän modulaarisuuden. Keskeisin liittymäprojektien tuotos on objektiyhdyskäytävä, joka laajentaa resurssien hallinnan tietoliikenteen laatupalveluita tarjoviin ratkaisuihin.

Lähtökohtaisesti QuO keskittyi palvelun laatutekijöiden erottamiseen sovelluksesta sekä lisämekanismien toteuttamiseen laajentaen TAO Real-Time CORBA OBRin toimintaa, mahdollistaen reflektiivisyyden ja adaptaation. Reflektiivisyydellä tarkoitetaan tässä yhteydessä kykyä tulkita järjestelmän ja ympäristön toimintaa sekä ominaisuuksia. Adaptaatiolla puolestaan tarkoitetaan kyvykkyyttä mukauttaa järjestelmän toimintaa muutoksiin.

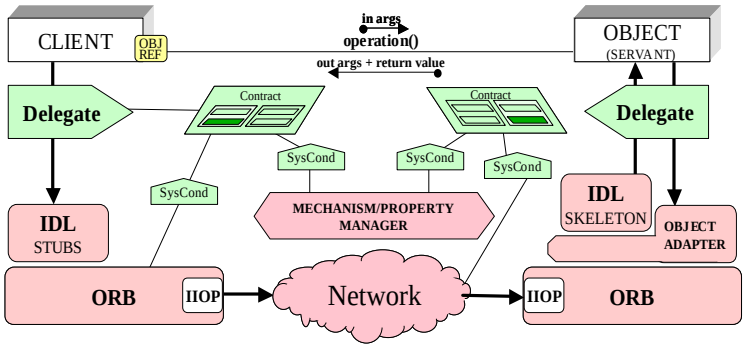
Seminaarialustus käsittelee tiivistetysti QuO:n osa-alueita, arkkitehtuuria ja toteutusta sekä tarkastelee lisäksi QuO:n myöhempiä laajennuksia palvelun laadusta poikkeaville, muille ei-toiminnallisten aspektien alueille, kuten luotettavuuteen sekä sovellustasoiseen selviytymiskykyisyyteen. Alustuksen seuraava kohta käsittelee yleisellä tasolla QuO ajoympäristön toimintaa, hajautuksen tuntumattomuudesta vastaavaa objektiyhdyskäytävää sekä QDL-kuvauskieliä. Kohta 3 kokoaa suunnittelupäätökset. Kohta 4 esittelee lyhyesti QuO:n uusimman sovellusalueen, selviytymiskykyiset sovellukset. Kohta 5 sisältää lyhyen pohdinnan QuO:n etenemisestä ja vaikutuksesta.

2 Quality Objects

QuO käyttöönotto edellyttää QuO ajoympäristöä [4], objektiyhdyskäytävää [5], QDL-kuvauskielten sekä räätälöityjenkoodigeneraattoreiden [6] hyödynnystä.

2.1 QuO ajoympäristö

QuO ajoympäristö (QuO Runtime) on kuvattuna kuvassa 1. Tavalliseen ORB:n toimintaan lisättävät osat on erotettuna vihreällä.



Kuva 1: QuO ajoympäristön arkkitehtuuri [4]

Ajoympäristö jakautuu sovelluskohtaisiin delegaatteihin sekä *QuO ytimeen* (QuO kernel), joka sisältää sekä järjestelmätilanneobjektit että sopimusobjektit.

2.1.1 Delegaatit

Delegaatit (delegates) vastaavat sovellusnäkökulman edustamisesta. Delegaatit ovat joukko lisäkerroksia, jotka liitetään CORBA asiakassovelluksen tynkään ja palvelinobjektien luurankoihin. Delegaatin kutsurajapinnat korvaavat IDL:n määrittämät kutsurajapinnat. Delegaatit määrittävät oletusarvoisen lisämekanismien käytön sekä tekevät päätöksen mahdollisista QuO ytimeltä saadun järjestelmän tilan perusteella adaptaatiota toteuttavien sovellustoimintojen kutsumisesta *ennen metodin kutsua* (premethod) ja *ennen sovellusmetodin paluuarvojen palauttamista* (postmethod). Delegaatit pyrkivät siis tuottamaan paikallisen adaptaation tuntumattomasti sovelluslogiikalle etäkutsujen yhteydessä (”in-band adaptaatio”. Delegaatit ovat eräällä tavalla analogisia erilliseen tietoliikenneprotollakerrokseen).

Delagaatit voivat määrittää useita yhteyksiä asiakkaasta etäobjektiin, kukin erillisillä palvelun laatuvaatimuksilla. Tällä tavalla asiakkaan saman aspektin eritasoiset palvelutarpeet voidaan erotella selkeyttäen QuO:lle sovelluksen todelliset resurssivaateet.

2.1.2 Sopimukset

Sopimukset (contracts) vastaavat systeeminäkö-kulman toteuttamisesta, koordinoiden keskitetysti delegaattien ja järjestelmäntilaobjektien adaptaatiota. Sopimukset ovat erillisiä CORBA-objekteja, joita voidaan kutsua delegaattien toimesta tai järjestelmätilaobjektin tuottaessa tapahtuman resurssin tilan muuttuessa. Sopimukset määrittävät odotukset sekä toimenpiteet mitattujen arvojen poiketessa odotusarvoista. Toimenpiteet voivat olla

- *Resurssien tilan muutos*: resurssien toiminnna muutostoimenpiteitä, joita sopimusobjektit toteuttavat järjestelmäntila-objektien välityksellä
- *Sovelluksen adaptaatio*: sovelluksen toimenpiteitä, joita kutsutaan *takaisinkutsua tarjoavan*

rajapinnan metodeilla (callback API method). Takaisinkutsu tarjoaa mahdollisuuden tiedottaa sovellukselle (ja tästä eteenpäin esimerkiksi käyttäjälle) ympäristön tilan muutoksista sekä esimerkiksi edelleenhyödyntää sovellukseen jo toteutettuja adaptaatiotekniikkoja QuO:n käyttöönotossa.

Sopimusobjektin tarjoamiin rajapintoihin kohdistuva kutsu johtaa tarvittessa useiden resurssien tilan muutokseen tai useisiin sovelluksen puolella tehtäviin toimenpiteisiin. Reaaliaikaivaateiden tapauksessa takaisinkutsuissa hyödynnetään TAO:n RT CORBAan mukautettua Event Service –palvelua [7].

Sopimusten hienojakoisuuden tarkastelemiseksi adaptaation päätösajankohdat voidaan jakaa *aikakausiin* (time epochs). Aikakausi määrittä milloin sovelluksen toiminnanmäärityksessä sitoudutaan tietynlaiseen käytökseen. Vaihtoehtoisia aikakausia ovat [3]:

- suunnittelu
- toteutus
- konfiguraatio
- käännös
- instanssin käynnistys
- ajoaika

QuO tukee sopimusten määritystä käännöksen yhteydessä. Sopimuksia ei voida määrittää instanssin käynnistyksessä tai ajonaikaisesti, eikä tällaiselle tuelle ole QuO 3.0:n dokumentaatiossa muutossuunnitelmia [14] [15]. Tämä johtunee siitä, ettei QuO:n absoluuttisista reaaliaikavaatimuksista lähtevissä sovellusalueissa ole tarvetta ajonaikaiselle sopimuksen sisällön ajonaikaiselle muuttamiselle.

2.1.3 Järjestelmäntilat

Järjestelmäntilat (system condition) vastaavat sopimus toiminnan liittämisestä infrastruktuuriin (resursseihin, objekteihin, mekanismeihin ja ORBiin), tarjoten kyseisen infrastruktuurin osan resurssinäkökulman. Järjestelmäntilat ovat CORBA-objekteja, joita kutsutaan vastaavat infrastuktuurin valvonta- ja hallintamekanimista, mitaten aktiivisesti infrastruktuurin toimintaa, tuottaen sopimusobjekteille herätteitä tilan muuttuessa systeeminäkökulman vaikutuksen arvioimiseksi sekä muuttaen sopimusten vaateiden mukaan infrasruktuurin toimintaa. Tämä mahdollistaa sovelluksen suoritusvaiheesta riippumattoman, resurssilähtöisen adaptaation (”out-of-band adaptaatio”. Järjestelmäntilat ovat analogisia tietoliikenneprotokollissa toiminnasta eristettyyn kontrollitasoon, joka käsittää esim. kaistan monitorointi- ja hallinnointimekanismit). Reaaliaikaisen tapahtumien käsittelyssä hyödynnetään hyödynnetään TAO:n RT CORBAan mukautettua Event Service –palvelua [7].

Järjestelmäntilaobjektit voidaan luokitella viiteen ryhmään [14]:

- *Yksinkertaiset muuttujaobjektit*: yksinkertaiset muuttujaobjektit ylläpitävät asiakkaan asettamaa muuttuja-arvoa, mahdollistaen asiakkaan vaikutuksen sopimuksen käsittelyyn
- *Mittausarvo-objektit*: mittausarvo-objektit kuvaavat jonkin resurssin toimintaa. Arvo saadaan järjestelmäobjektissa toteutetulla mittaus-mekanismilla (esim. sovelluksen läpimenovuo)
- *Tilaobjektit*: tilaobjektit hakevat suoraan resurssilta tai epäsuoraan resurssin hallinnoinnista vastaavalta palvelulta resurssin tilan
- *Koostearvo-objektit*: koostearvo-objektit tuottavat mittausarvon laskennallisesti useiden resurssien ja/tai järjestelmänobjektien tuottamien tila tai mittausarvon testauksen perusteella. Koostearvo peittää yksitäisten infrastruktuurin osien tilan politiikalta.
- *Hallinnointiobjektit*: hallinnointiobjektit tuottavat itse tai peittävät ulkopuolisen resurssienhallintamekanismin rajapinnan mahdollistaen infrastruktuurin adaptaation. Hallinnointiobjektit voivat muodostaa keskinäisiä riippuvuuksia, mikäli esimerkiksi yhden hallinnointiobjektin tehtävänä on peittää lopullisen hallintamekanismin monimutkaisuutta.

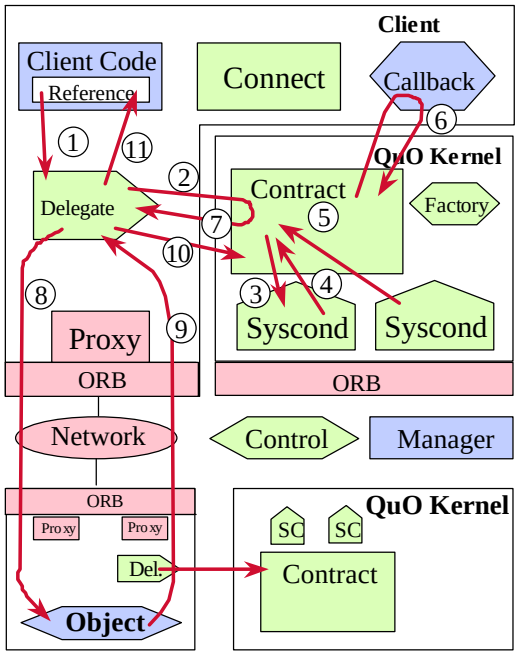
Järjestelmäntilaobjektit lasketaan osaksi infrastruktuuria, eivätkä täten ole toteutettuna osana QuO:ta.

2.1.4 Toimintaesimerkki

QuO:n ajoympäristön eri osien yhteistoiminta käy parhaiten ilmi yleisestä esimerkistä. Kuva 2 sisältää tyypillisen sovelluksen etäobjektin kutsun suorituspolun. Kuva sisältää seuraavat vaiheet [4]

1) Asiakassovellus tekee etämetodikutsun, joka on delegaatin toiminnan piirissä
2) Delegaatti kutsuu sopimusobjektilta tila-arviota
3) Sopimusobjekti kutsuu tarvittavia järjestelmäntilaobjekteja (tässä yhtä) tiedustellaan paikallisresurssien tilaa
4) Sopimus saa vedoksen järjestelmäntilaobjektin peittämien resurssien tilasta
5) Sopimus arvioidaan uudestaan toisen järjestelmäntilaobjektin saadessa resurssilta tilanmuutostapahtuman
6) Jälkimmäinen järjestelmäobjektin arvo aiheuttaa muutoksen mittausarvoalueelta toiselle. Sopimus reagoi tuottamalla takaisinkutsun asiakassovellukselle tiedottaakseen muutoksesta
7) Paikallisresurssien tilaa kuvaava mittausarvo palautetaan delegaatin metodille
8) Palvelun laadun vastatessa odotusarvoaluetta, siirretään palvelukutsu eteenpäin suorituspolulla ORBin perustoiminnallisuuden mukaisesti
9) Palvelinobjekti palauttaa paluuarvon. Paluuarvoa arvioidaan palvelimen paikallisessa ajoympäristössä paikallista sopimusta vasten ennen metodin toteutusta ja tämän jälkeen. Mikäli palvelun laatu vastaa odotusarvoja siirretään paluuarvo ORBin suorituspolulle.

10) Ennen paluuarvon palauttamista asiakassovellukselle sopimus arvioidaan uudestaan. Mikäli palvelun laatu vastaa odotusarvoja siirretään paluuarvo asiakassovellukselle käsiteltäväksi
11) Paluuarvo siirretään asiakassovelluksen suorituspolkuun



Kuva 2: Toimintaesimerkki [4] [13]

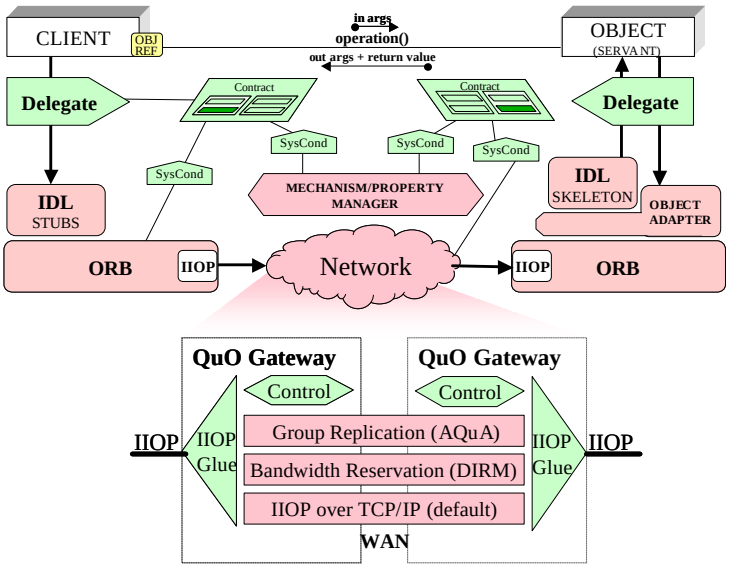
Kuten toiminnasta voidaan todeta, QuO ei tuota infrastruktuurin adaptaatiota vaan tuottaa ainoastaan instrumentoinnin ja tarvittavat mekanismit,
joilla liittää itse infrastruktuurin hallinnasta vastaavat mekanismit (CORBA-objektit tai kuorrutetut, CORBA-rajapinnat omaavat sovellukset), mahdollistaen näiden ja sovelluksen palvelun laatuun vaikuttavien tekijöiden käsittelyn systeeminäkökulmasta. Toisin sanoen QuO mahdollistaa systeeminäkökulman mukaisen adaptaation, mutta ei kykene tuottamaan sitä itsenäisesti.

2.2 Objektiyhdyskäytävä

QuO edellyttää lisäpalveluiden tai laajennuksien sisällyttämistä ajoympäristöön, jotta QuO ydin saa palautetta tietoliikenneresursseista sekä kykenee vaikuttamaan siirtotien tietoliikenneresurssien toimintaan adaptaation toteuttamiseksi. *Objektiyhdyskäytävä* (object gateway) vastaa ORBille tuntumattomasta aspektien toteutuksesta hajautuksessa. Lisäksi yhdyskäytävä integroi seuraavat ratkaisut osaksi lisäpalveluita, joilla tukea reaaliaikaisuutta sekä luotettavuutta.

- *QoSME (Quality of Service Management Environment)* [8]*:* QoSME on Columbian yliopiston laatutakeita tukevien tietoliikenneratkaisujen (pääasiassa ATM sekä IntServ-arkkitehtuurin RSVP-signalointi-protokolla) ohjelmointirajapintoja abstraktoiva korkean tason ohjelmointirajapinta. QoSME mahdollistaa lisäksi mm. laadun seurannan oman SNMP MIB-rakenteen avulla.
- *DIRM (Dynamic Integrated Resource Management)* [9]*:* DIRM integroi QoSME:n ohjelmointirajapinnan objektiyhdyskäytävään, peittäen tämän yksinkertaisemmalla rajapinnalla. Ylimääräinen abstraktiokerros mahdollistaa tarvittaessa toisen palvelun laatutakeita tukevan ohjelmointirajapinnan käyttöönoton mahdollisimman vähäisin muutoksin itse objektiyhdyskäytävään.
- *AQuA* [10]*:* AQuA integroi Ensemble-työkalupaketin luotettavan ryhmäkommunikointi-mekanismin objektiyhdyskäytävään, tuottaen vikasietoisten prosessiryhmien tuen.

Kuva 3 sisältää sekä yleiset peruselementit että projekteista integroidut lisätoiminnallisuudet.



Kuva 3: Objektikäytävän yleisarkkitehtuuri [5] [13]

2.2.1 Objektiyhdyskäytävän yleiskuvaus

Asiakaspuolella *IIOP-välikerros* (IIOP-Glue) terminoi IIOP-yhteyden asiakkassovellukseen, toimien asiakkaalle palvelimena. IIOP-välikerrros muodostaa DSI:n avulla uuden kutsun, DSI:n mahdollistaessa ajonaikaisen kutsun määrityksen. Sanomasta muodostetaan GIOP-sanoma, joka siirretään eteenpäin protokollakohtaiseen jonoon, josta *yhdyskäytäväkoordinaattori* (Gateway Coordinator) ottaa nämä edelleenkäsiteltäväksi. Koordinaattori huolehtii vuonhallinnasta, virhehallinasta sekä reitityspolun valinnasta, mikäli tietoliikenneprotokolla tämän mahdollistaa sekä mahdollisista protokollan erityisominaisuuksista. Koordinaattori sijoittaa kutsujen vastaukset asiakassovelluksen puolella vastausjonoon, josta IIOP-välikerroksen mekanismi välittää ne IIOP-sanomina asiakkaalle. Jonon ja

jonotunnisteiden avulla koordinaattori ja IIOP-välikerros huolehtivat siitä, että välitetyille viesteille löytyy vastinepari vastausjonosta.

Palvelinpuolella IIOP-välikerros poistaa kutsuista asiakassovelluksen puolen välikerroksen tuottaman yhdyskäytävien väliseen liikenteeseen liittyvät osiot, terminoiden yhdyskäytävien välisen IIOP-yhteyden. Palvelinpuolen IIOP-välikerros muodostaa uuden kutsun DII:n avulla alkuperäisen kutsun kohteena olleelle etäobjektille. IIOP-välikerros sijoittaa palvelimen paluusanomat protokollakohtaiseen vastausjonoon, peilaten asiakaspuolen toimintaa.

IIOP-välikerroksen GIOP-sanoman protokolla-kohtaiseen jonoon sijoitus on toteutettu Reactor-suunnittelumallilla [11], joka mahdollistaa protokollakohtaisen koordinaattorin valinnan tapahtumapohjaisesti objektiyhdyskäytävän toimesta sekä sanomien lähetysten ja vastausten demultipleksoinnin. Ratkaisu edellyttää IIOP 1.2:sta muiden versioiden rajoitusten vuoksi:

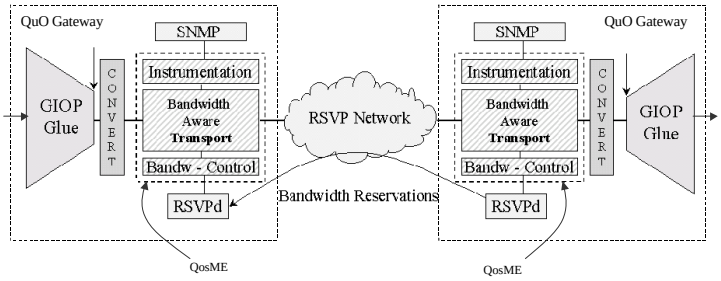
- IIOP 1.0 edellyttää sovellustason sanoman peräkkäiskäsittelyä, joka tekee useiden useiden objektiyhdyskäytävien välisten yhteyksien rinnakkaiskäytön mahdottomaksi ja voi muodostaa yhdyskäytävästä pullonkaulan, koska GIOP:n sanomanpituus on mielivaltaiinen
- IIOP 1.1 rajoittuu GIOP-sanoman fragmentointiin, eikä täten puolestaan mahdollista siirtokerrosmuunnosta (esim. integrated services – pohjaisen ATM:n käyttöä ja osittaista viestin kokoamista kohteessa käyttäen IIOP 1.2:n IIOP-fragmentointia).

Mikäli objektiyhdyskäytävien välisessä liikenteessä tapahtuu virhe, muodostetaan tästä sovellukselle erillinen järjestelmävirhe sovellusvirheen sijaan, sillä yhdyskäytävä on selkeä ORBin laajennus.

Mikäli objektiyhdyskäytävä tarjoaa lisäpalveluita, ottavat yhdyskäytävät vastaan QuO ajoympäristön hallinnointisanomat, joiden kautta ajoympäristön sopimusten ohjaamat järjestelmäntilaobjektit tekevät muutoksia toimintaan.

2.2.2 Reaaliaikaisuuden tuki

DIRM-projektin QoSME:n integraation myötä objektiyhdyskäytävä sisältää Integrated Services ja ATM-tuen. Kuva 4 sisältää laajennuksen peruselementit.



Kuva 4: DIRM-objektyhdysikäytävä [5]

QoSME:n liittäminen on toteutettu osaan:

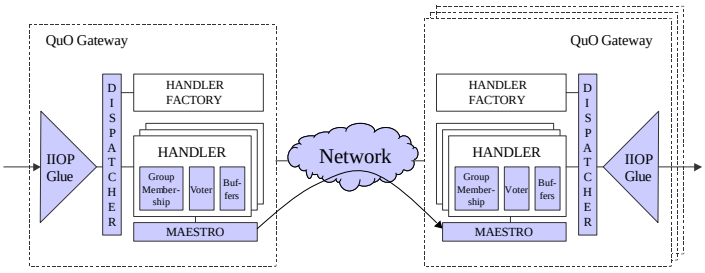
- *Objektiyhdyskäytävän RSVP-laajennus* (Bandwidth Aware Transport): RSVP-laajennus integroi QoSME:n ja laajentaa sen palvelun laadun monitoroinnin ORBin sanomatasolle RSVP-yhteyskohtaisesta tasosta.
- *RSVP-hallintarajapinta* (Bandw-Control): RSVP-hallintarajapinta tarjoaa CORBA-objektille rajapinnat resurssivarausten tekemiseen. Rajapinta on käytettävissä yhteyden muodostamiseksi sekä adaptaation toteuttamiseksi järjestelmäntila-objektin toimesta. RSVP-hallintarajapinta tukee sekä lähettäjä- että vastaanottajalähtöisen resurssivarauksen tekoa.
- *Monitorointi ja instrumentointi* (Instrumentation): Monitorointi- ja instrumentointimekanismi ylläpitää objektiyhdyskäytävän tilatietoja yhteyden käyttäytymisestä hyödyntämällä QoSME:n pohjalta toteutettua laajennettua MIB-rakennetta, josta SNMP-pohjainen järjestelmäntilaobjekti voi käydä hakemassa tilatiedot. Laajennettu MIB-rakenne mahdollistaa RSVP-laajennuksen sanomatasoisen toiminnan.

Yhdyskäytävä kykenee tekemään useita eritasoisia resurssivarauksia kahden yhdyskäytävän välille. IIOP-välikerroksessa *konversiokerros* (convert) ylläpitää GIOP-sanomien ja resurssivarausten välistä yhteyttä, tehden tarvittaessa uusia resurssivarauksia RSVP-hallintarajapinnan kautta.

2.2.3 Luotettavuuden tuki

AquA-projektin taustalla on tavoite adaptaativisesta luotettavuudesta, jossa mekanismia voidaan säädellä suhteessa suorituskyky, tietoturva ja sovelluksen selviytymiskyvyn vaateisiin. Lisäksi prosessiryhmän toteutuksen on kyettävä kerrostamaan luotettavuuspalvelut erilaisten piirteiden perusteella. Lisäksi adaptaatiomekanismin on oltava vaihdettavissa.

Kuva 5 kuvastaa AquAn rakennetta yhdyskäytävään integroituna.



Kuva 5: AQuA-objektiyhdyskäytävä prosessiryhmätuki [5] [9]

Välittäjä (Dispatcher) vastaa uuden kutsun luonnista. Välittäjä muodostaa yhteyden *käsittelijätehtaan (handler factory)* avulla ja ylläpitää tietoja asiakkaan käyttämistä käsittelijöistä, välittäen IIOP-sanoman oikealle käsittelijälle jo muodostettujen yhteyksien osalta.

Käsittelijätehdas on abstract factory suunnittelumallin [12:87-96] mukainen, käsittelijän toteutuksen peittävä, käsittelijöiden elinkaaresta vastaava rajapinta. Käsittelijätehdas sisältää erillisen *käsittelijöiden varaston* (handler repository), joka omaa kuvaukset vaihtoehtoisten käsittelijöiden toiminnasta. Käsittelijät jakautuvat kahteen ylätason luokkaan:

- *staattiset käsittelijät:* staattiset käsittelijät vastaavat prosessiryhmään liittämisestä ja ylläpitämisestä ryhmässä, mikäli kutsuvat objekti on pysyvä prosessiryhmän jäseniä. Staattiset käsittelijät edelleen jakautuvat aktiivisen replikoinnin toteuttaviin, joissa ryhmästä useammat osapuolet osallistuvat kutsun prosessointiin sekä passiivisen replikoinnin toteuttaviin, joissa prosessin vikasietoisuus toteutetaan sanomavarastoinnilla ja varaprosessilla tai –prosessiryhmällä. Passiivisessa replikoinnissa käsittelijät vastaavat sanomavarastoinnin puskuroinnin toteutuksesta (buffers), aktiivisessa replikoinnissa käsittelijät sisältävät tiedon ryhmän jäsenistä, joilla on valtaa vastaussanoman oikeellisuudesta päätettäessä sekä vallan määrästä, mikäli tämä ei jakaudu tasaisesti (voters).
- *dynaamiset käsittelijät:* dynaamiset käsittelijät vastaavat itsenäisen prosessin väliaikaisesta tarpeesta kommunikoida prosessiryhmälle. Dynaamisen käsittelijään tukeutuvat objektit liittyvät prosessiryhmään aina sanomanlähetyksen yhteydessä, jonka jälkeen objektit poistuvat ryhmästä.

2.3 QDL-kuvauskielet ja koodigeneraattorit

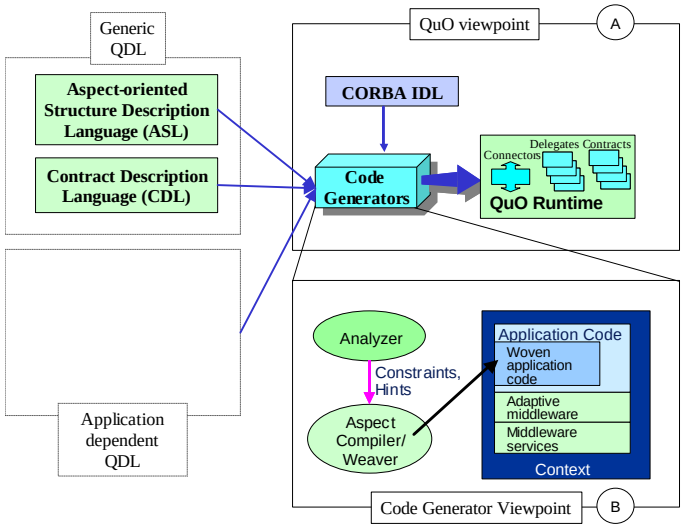
QuO:n ei-funktionaalisten toiminnallisuuksien erottaminen sovelluskoodista perustuu piirteitä määrittävien kuvauskieliin (Quality Descriptive Languages, QDL). Näillä tuotetaan sekä delegaatit että sopimusobjektit QuO ajoympäristöä varten.

Kuvauskieliä ei ole laadittu periaatteella ”kuvauskieli per aspekti” vaan kriteereinä uuden kuvauskielen määritykselle ovat esimerkiksi seuraavat: [4]

- *erilainen abstraktiotaso:* mikäli on tarpeellista kyetä tarjoamaan nykyisen kuvauskielen abstraktion lisäksi tarkemman tason kuvauksia, on tämä toteuttavissa erilliseksi kieleksi
- *aspektin erityispiirteiden edellyttämät erityispiirteet*: mikäli kuvauskielet eivät kykene kuvaamaan aspektille ominaisia piirteitä, voi olla tarpeellista luoda näille oma kuvauskieli huolimatta siitä, että osa toiminnallisuudesta voidaan määrittää muussa yhteydessä. Erityisesti esimerkiksi aspektit kuten puolustuskykyisyys, joka yhdistää luotettavuuden, tietoturvan ja palvelun laadun, eivät ole erotettavissa kokonaan oman kuvauskielen piiriin.

Kielet ovat ilmaisuvoimaltaan ja käyttötavaltaan käyttöjärjestelmien komentotulkkikielten kaltaisia. Verrattuna DCOM ja EJB komponenttimallien sovelluspalvelinten deklaratiiviseen ohjelmointimalliin ominaisuusarvojen käyttö on siis monimutkaisempaa, mutta avoimen toteutuksen mukaisesti myös tarjoaa määrittelyn vapauden.

Kuva 6 esittää kuvauskielenkäytön vaikutuksen sovelluksen käännökseen.



Kuva 6: QDL kuvauskielet ja aspektin kutominen [4] [13]

Kuvan vasemmalla puolella kuvauskielet jakautuvat *yleisiin kuvauskieliin* (Generic QDL), joita käytetään riippumatta aspekteista, joihin QuO:ta sovelletaan sekä näitä mahdollisesti täydentäviin *sovellusaluekohtaisiin kuvauskieliin* (application dependent QDL).

QuO:n näkökulmasta (A) kuvauskielet integroidaan koodigeneraattorilla sekä QuO ajoympäristöön että asiakassovelluksiin että palvelinobjekteihin.

Koodigeneraattorin näkökulmasta (B) kuvaukset jäsennetään ja sisällön perusteella *kudotaan* tarvittava lisäkoodi sovelluksiin ja IDL-kuvauksiin jollaisina jäsennin sekä QuO:n että asiakassovellukset ja palvelinobjektit näkee. Vaihe on analoginen esikääntäjän käyttämiseen, joskin hyödyntää sovelluksesta irrallisia kuvauksia, määrittäen IDL-rajapintojen perusteella asiakassovelluksen koodin muokkaamisen. Oleellista on huomioida, ettei ORBiin olla tekemässä mitään muutoksia, eikä QuO:n käyttö periaatteessa edellytä sovelluskoodissa huomiointia.

Seuraavaksi käsitellään yleisiä QuO:n kuvauskieliä. Käsittely keskittyy QuO 2.1:n kuvauskieliin, määrittäen lisäksi päivitykset, joita QuO 3.0 sisältää.

2.3.1 SDL ja ASL

SDL (Structure Definition Language) kuvauskielen määritys sisältää

- delegaatit ja näiden rajapinnat (interfaces)
- delegaatin hyödyntämät sopimukset (contracts)
- mittausarvoalueet (region)
- sopimuksen toimenpiteet mittausarvoalueittain
- oletusarvoiset toimenpiteet

Tarkastellaan esimerkinomaisesti asetähtäys-järjestelmän kutsurajapintaa, joka omaa metodit kohteen etäisyyden laskentaan ja kohteen tunnistamiseen:

```
interface Targeting {
 long calculate_distance_to_target(in long xcoord,
                                   in long ycoord);
 long identify_target(in long xcoord,
                      in long ycoord);
};
```

Kuvaus 1: Targeting IDL [6]

IDL-kuvauksen tueksi määritetään adaptaatio kuvauksen 2 SDL-kuvauksella, joka määrittää delegaatin toiminnan:

```
delegate behavior for
       interfaces Targeting and
       contracts Replication is
  obj : bind Targeting with name
    SingleTargetingObject;
  group : bind Targeting with characteristics
    { Replicated = True };
  call calculate_distance_to_target :
    region Available.Normal :
      pass to group;
    region Low_Cost.Normal :
      pass to obj;
    region Available.TooLow :
      throw AvailabilityDegraded;
  return calculate_distance_to_target :
    pass_through;
  default : pass_to object
end delegate behavior;
```

Kuvaus 2: SDL-kuvaus esimerkki [6]

Kuvaus määrittää targeting-rajapinnan metodille calculate_distance_to_target sopimuksen Replication puitteissa kriteerit, jotka kuvaavat toimintaa ennen metodikutsua:

- Available odotusarvoalueella normal mittausarvoalueella toimittaessa kutsutaan prosessiryhmää
- Low Cost odotusarvoalueella normal mittausarvoalueella toimittaessa kutsutaan palvelinobjektiaAvailable odotusarvolla TooLow mittausarvoalueella toimittaessa toteutetaan poikkeuskäsittely Metodikutsulle ei määritellä tarkistuksia ennen paluuarvon palautusta asiakassovelluksen suorituspolkuun. QuO 3.0:assa ASL (Aspect-Oriented Structure Definition Language) on SDL:n korvaava kuvauskieli, joka määrittää uusia järjestelmän loogisia osia, jotka helpottavat SDL-kuvauksien paketointia delegaattien keskinäisten ja sopimuksien välisen riippuvuuksien osoittamiseksi. Näiden muutosten myötä vältetään QuO 2.1:ssä käytössä ollut erillisen aspektien konfiguraatiokuvauskielen, CSL:n (Configuration Setup Language), käyttö. Kuitenkin määrittelyn ilmaisuvoima muilta osin pysyy samana.CDL

CDL (Contract Description Language) kuvaa sopimusobjektit, määrittäen arvoalueiden sisällön sekä menettelyn resurssilähtöiselle, out-of-band -adaptaatiolle. CDL määritys sisältää

- sopimukset (contract)
- sopimuskohtaiset odotusalueet (negotiated regions)
- monitoreiden mittausalueet (reality regions)
- tilat (states), jotka ovat vaihtoehtoinen tapa määrittää alueita
- toimenpiteet odotusarvoalueiden ja mittausalueiden erotessa (transitions) kohdistuen joko asiakkaan takaisinkutsuihin (client callback) tai järjestelmätilaobjektien metodeihin (system condition methods)

Kuvaus 3 täydentää aiempaa SDL-kuvausta, sisältäen määritelmät sopimuksen kahdelle odotusarvoalueelle, Low Cost ja High Avail.

```
contract Replication
 ( object client, object server ) is
 var measured  : measured_replication init();
 negotiated regions are
  region Low_Cost : when
   client.expectations.requested == 1 =>
   reality regions are
    region Too_Low  : when measured == 0 =>
    region Normal   : when measured == 1 =>
    region Too_High : when measured > 1 =>
    transitions are
     transition any->Too_Low :
      client.callbacks.availability_degraded();
     transition any->Normal :
      client.callbacks.availability_back_to_normal();
    end transitions;
   end reality regions;
  region High_Avail : when
   client.expectations.requested > 1 =>
   reality regions are
    region Too_Low  : when measured <= 1 =>
    region Normal   : when measured > 1  =>
    transitions are
     transition any->Too_Low :
      client.callbacks.availability_degraded();
     transition any->Normal :
      client.callbacks.availability_back_to_normal();
     end transitions;
   end reality regions;
   transitions are
     transition Low_Cost->High_Avail :
       adjust_degree_of_replication
        (client.expectations.requested);
     transition High_Avail->Low_Cost :
       adjust_degree_of_replication
        (client.expectations.requested);
   end transitions;
 end negotiated regions;
end Replication;
```

Kuvaus 3: CDL-kuvaus esimerkki [6]

Kuvauksen perusteella hyväksyttävät mittausarvoalueet ovat:

- LowCost odotusarvoalueelle soveltuvat replikointimittauksen (measured replication) arvo x, mikäli x>=1.
- HighAvail odotusarvoalueella soveltuu vain replikointimittauksen arvo x, mikäli x>1.

Poikkeuksien yhteydessä toteutetaan toimenpiteiksi määritetyt metodit.

´ Mikäli siis esimerkissä replikointimittauksen arvo vastaa prosessiryhmän prosessin kopioiden lukumäärää, on helposti nähtävissä odotusarvoalueiden ajatus. Molemmissa odotusarvokuvauksissa sovellukselle annetaan lisäksi tieto prosessiryhmän tilan palautumisesta odotusarvon mukaiseksi.

3 QuO:n tavoitteet ja arkkitehtuuri

Adaptaation toteutus edellyttää tietoisia päätöksiä sekä tavoitteiden että arkkitehtuurin ja toteutustekniikoiden osalta, joita tarkastellaan seuraavaksi.

3.1 Tavoitteet

QuO:n neljää keskeisintä tavoitetta voidaan arvioida kohdassa 2 esitettyjen mekanismien puitteissa. Tavoitteet ja näitä tukevat ratkaisut ovat [3]:

- *järjestelmän tilan määritys resurssien, ajankohdan ja sovellustoiminnan osalta:* delegaatit sitoutuvat asiakassovelluksen metodeihin sekä palvelinobjektin metodeihin määrittäen mahdolliset käyttöajankohdat resurssien tilapoikkemien rinnalla. Toimenpiteet määrittävät erilaiset resurssilta odotettavat adaptaatiotoimenpiteet.
- *ei-toiminnallisten piirteiden systematisointi:* delegaattien tuottaminen useissa kerroksissa yhden asiakassovelluksen osalta mahdollistaa kaikkien tarvittaen aspektien yhdistämisen. Järjestelmäntilaobjektit ovat erillisiä objekteja. Järjetelmäntilat johtavat samanlaiseen käyttäytymiseen läpi sovelluksen, edellyttäen että delegaattien ja sopimuksen kuvaukset eivät ole ristiriitaisia ja paikallisen adaptaation tuottavat sopimukset ovat kullakin järjestelmän osapuolella yhtenäisen QDL-kuvauksen mukaisesti määritetty.
- *sovelluskehittäjien ei-toiminnallisten tavoitteiden eksplisiittinen esitys:* tukeutumalla QDL-kuvauskieliin, sovelluskehittäjiä vaaditaan käymään läpi vaihtoehdot ja määrittämään millaisia tarpeita sovelluksella on. Lisäksi heitä tuetaan abstraktoimalla esim. WAN-ympäristöjen palvelun laatuun sekä luotettavaan ryhmäkommunikointiin liittyvät piirteet, joihin sovelluskehittäjät usein eivät omaa tarvittavaa osaamista.
- *uudelleenkäytön tuki:* järjestelmätilaobjektit on tarkoituksenmukaista koota kirjastoksi, jota QoS-piirteisiin erikoistuneet sovelluskehittäjät voivat edelleenkehittää. Lisäksi QuO:n ulkopuoliset, ei-toiminnallisiin piirteisiin liittyvät sovellukset ja varusohjelmat voidaan integroida järjestelmäntilaobjektien välityksellä.

3.2 Arkkitehtuuri ja toteutus

QuO:n arkkitehtuuri ja sen toteutus sisältää selkeitä valintoja, joilla pyritään tuottamaan tuntumattomuus, suorituskyky, modulaarisuus ja adaptaatiotason mahdollisimman korkea granulariteettitaso.
Valinnat näkyvät selkeästi ajoympäristön kapseloinnissa [4] [14], adaptaation toteutuksessa sekä QDL-kielien sovelluslähtöisyydessä [3] [6].

3.2.1 Ajoympäristön kapselointi

Alunperin ajoympäristö olisi ollut mahdollista toteuttaa kokonaisuudessaan asiakassovelluksessa ja palvelinobjekteissa, jolloin vältyttäisiin delegaatin ja QuO ytimen välisiltä, ORBin kautta kulkevilta metodikutsuilta. Tällöin kuitenkin menetettäisiin seuraavat edut:

- *Sopimusobjektien joustavuus:* sopimusobjektien osalta koodigeneraattoreilta edellytetään ainoastaan tuki QuO-ytimen toteutuskielelle. Täten koodigeneraattorien on kyettävä tuottamaan ainoastaan delegaattiobjekteja ohjelmointikieli-riippumattomasti.
- *Luonnollinen rinnakkainen käsittely järjestelmän osien välillä:* Sijoittamalla delegaatit ja sopimusobjektit erillisiin ympäristöihin saadaan toteutettua vaivattomasti rinnakkaisuuden käsittely asiakassovelluksen metodikäsittelyn ja ympäristön resurssien tapahtumiin perustuvan adaptaation välillä
- *Hajautuksen tuki:* tarvittaessa QuO ydin voidaan hajauttaa erikseen itse delegaateista. Tämä voi olla hyödyllinen piirre rajatun muistikapasiteetin omaavissa alustoissa (esim. sensorijärjestelmät tai muut sulautetut järjestelmät)

QuO 3.0:n myötä ajoympäristö on vaihtoehtoisesti mahdollista integroida delegaatteihin, siirtäen kaiken adaptaatiotoiminnan asiakassovelluksiin ja palvelinobjekteihin.

3.2.2 Adaptaation toteutus

Sen sijaan, että in-band adaptaation toteutus perustuisi delegaatteihin, vaihtoehtona tuntumattomalle yleiselle adaptaatiomekanismin toteutukselle olisi ollut toteutus

- *CORBA:n kaappausmekanismilla*: kaappaus Portable Interceptor määrityksen [16:Chapter21] mukaisia kaappausrajapintoja
- *muulla kaappausmekanismilla:* hyödyntämällä esimerkiksi käyttöjärjestelmätason kaappaus-rajapintoja [17]

Delegaattien puitteissa toteutus kuitenkin omaa seuraavat edut muihin nähden [3] [4]:

- *Sidos asiakkaaseen:* asiakassovellus ja delegaatti ovat sidoksissa toisiinsa, joten asiakassovellus ei voi kaatua delegaatin kaatumatta. Lisäksi asiakkaan ja delegaatin välillä ei ole ruuhkan tai viiveen vaaraa
- *Resurssien käyttö vain tarvittaessa:* delegaatti vaatii resursseja ainoastaan sovelluksen käyttäessä metodia, johon on määritelty delegaatille toiminnallisuutta päinvastoin kuin kaappaus. Lisäksi delegaatti voi tarvittaessa välivarastoida arvoja, ja hyödyntää niitä tiettyjen arvoaluerajojen puittteissa. Esimerkiksi asiakassovelluksen käyttäjän tunnistus voitaisiin tehdä istunnon alussa ja välivarastoimalla tunnistustiedot, voidaan näitä käyttää sovelluksen edetessä istunnon säilyessä,

ilman että sovelluksessa on huomioitu käyttäjän tunnistusta.

- *Läpinäkyvyys ORBille:* delegaattimekanismin toteutus ei itsessään edellytä mitään viivettä tuottavaa mekanismia päinvastoin kuin kaappaus.

Vaikkakaan toteutustekniikkaa ei ole määritetty CORBA-standardissa, ovat vastaavat toteutukset olleet käytössä mm. kuormanjaon toteuttamiseen esim. Visigenic Visibroker ORBissa [18].

Delegaattien heikkous, metodikutsun edellytys, on korjattu out-of-band –adaptaatiolla tuottamalla järjestelmäntilaobjekteilta tapahtumia resurssi-muutoksen ilmoittamiseksi sopimusobjekteille, edelleenkin välttäen kaappauksen kaltaisen, suorituskyvyltään tehottomamman tekniikan käyttö.

3.2.3 QDL-kuvauskielten sovelluslähtöisyys

Sopimukset määrittävä CDL kuvaa sopimuksen peruselementit aspektiriippumattomasti. CDL pyrkii tuottamaan systeeminäkökulman SDL:n/ASL:n edustaessa aspektiriippumattomasti sovellusta.

Huomioiden että QoS:stä on kirjoitettu huomattavia määriä taksonomioita ja erilaisia QoS:n piirteitä kaappavia arkkitehtuurikuvauksia, tämä ei ole mitenkään suoraviivainen valinta. Esimerkiksi Frølund ja Koistinen kuvaavat QoS Modelling Language (QML) –kuvauskielen[19], joka keskittyy laajentamaan UML:n kuvausvoimaa erottelemalla QoS:n erillisiä ominaisuuksia kuten viivettä, saatavuutta ja läpimenovuota UML:a mukaillen. Tällaisella kielellä toteutettuna tarvittaisiin luotettavuutta varten oma kuvauskielensä, eivätkä asiakkaan vaikutus-mahdollisuudet olisi kuvattavissa.

CDL:n mahdollisuus määrittää sopimuksia useisiin eri sovellustarpeisiin, huomioiden sovellusnäkökulman omat adaptaatiomahdollisuudet, tarjoaa mahdollisuudet perehtyä yhteen kuvaustapaan. Täten työtä pyritään eriyttämään erilliseksi sovelluskehityksen osaksi erilliselle sovelluskehitysprosessin roolille.

3.3 Arkkitehtuurin ja toteutuksen tulevaisuus

Verrattuna esimerkiksi OpenORBin [20] rakenteeseen, QuO sisältää vain toiminnalliseen adaptaatioon liittyviä järjestelmän kuvauksia ja mekanismeja. Lisäksi infrastruktuurin kuvaus on täysin sivuttu erillisillä järjestelmäntilaobjekteilla.

QuO:n tulevat kehityslinjat ottavat kantaa pääasiassa jälkimmäiseen kysymykseen. Lähtökohtana on tuottaa toiminnallinen adaptaatio kuvaamalla ympäristö kolmella kuvauskerroksella, joskin kuvauksissa on kyettävä määrittämään myös sovellusrakenteellisia tekijöitä [15]:

- *QoS malli*: QoS-malli kuvaa korkean tason, aspektiin soveltuvan kuvausmallin (esim. jonoverkko reaaliaikaisuutta varten, tietoturva-politiikan mukaiset periaatteet tietoturvaa määrittäviä sopimuksia varten).
- *Sovellusrakenne*: Sovellusrakenne kuvaa järjestelmän prosessit sekä näiden keskinäiset suhteet ja prosessien väliset riippuvuudet aspektin näkökulmasta.
- *Resurssirakenne*: Resurssirakenne kuvaa järjestelmän fyysisten resurssit sekä näiden väliset suhteet toisiinsa sekä sijoittaa sovellusprosessit resursseihin.

QuO 3.0 sisältää QoS-mallin ja resurssirakenteen prototyyppitoteutuksen DIRM-projektin Resource Status Services –palvelun (RSS) puitteissa. RSS sisältää W3C:n RDL:llä kuvatun kuvausrakenteen ja elementit ontologian prototyypin. RSS kuvaa resursseja yksinkertaisesti kolmesta näkökulmasta [15]:

- *Sovellusten väliset kapasiteetit*: TCP/IP-vuokapasiteetin kuvaus maksimi- ja käyttämättömän kapasiteetina
- *Verkkosegmenttien linkkien kapasiteetit*: verkkoavaruuksien välisten linkkien kapasiteetit maksimi ja käyttämättömän kapaseetina
- *Palvelinresurssit*: palvelinten kapasiteetti prosessori, verkkoliittymä ja muistikapasiteettien kuvauksina

Vaihtoehtoisesti voidaan soveltaa erillistä tilahallinnasta vastaava järjestelmän osaa, joka ei tarvitse tietoa monitorointidatan kuluttajista vaan kapseloi monitoroinnista vastaavat mekanismit.

4 QuO ja selviytyvät sovellukset

QuO tunnetuin sovellusalue lienee AIRES (Aspects in Real-Time Systems), jossa, tukeutuen TAO:n CORBA A/V:n sekä DIRM-projektin verkonhallinnointia mahdollistaviin tuotoksiin, kyettiin tuottamaan reaaliaikaisen multimedia-liittymän omaava miehittämättömän lentokoneen (UAV, unmanned aerial vehicle) hallinnointisovellus. [21] [22]

AIRESin myötä tietoturvaan ja luotettavuuteen keskittyneet projektit alkoivat integroida tuotoksiaan, jonka myötä useat projektit limittyivät selviytymiskykyä tuottavan välikerros-ohjelmiston piiriin, tukeakseen tätä varten perustettua APOD (Applications Providing their Own Defense) -projektia. Sovelluksen selvitiytymiskykyisyyttä korostettiin erityisesti QuO:n missiokriittisten sovellusalueiden vuoksi.

Selviytymiskykyiset sovellukset ovat kolmas vaihe tietoturvallisten järjestelmien tutkimuksessa. Ensimmäinen vaihe 1980-luvun puolivälin vaiheilla pohjasi paljolti käyttäjähallintaan ja standardeihin pohjautuvaa määritystoimintaa, joiden pohjalta tuotettiin joukko järjestelmien arviointikriteerejä ja toimintaohjeita (mm. USAn puolustusministeriön ”sateenkaaristandardit” sisältäen DOD:n oranssin kirjan, ”Trusted Computer System Evaluation Criteria”). Toinen vaihe puolestaan tukeutui oletukseen täydellisen tietoturvan tuottamiseen salauksella, erilaisilla teknisillä toiminnan rajoitinmekanismeilla kuten palomuureilla sekä hyökkäykseen havainnointiin tilannetietoisuuteen IDS (Intrusion Detection System)

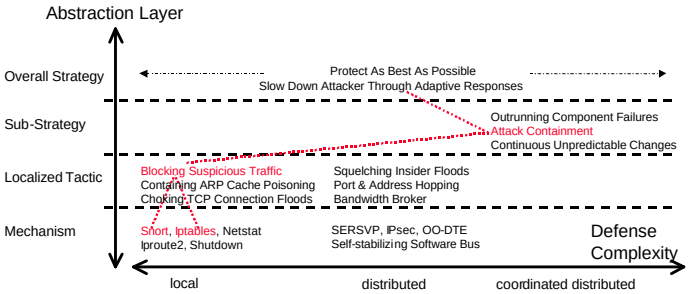
-järjestelmillä. Ero kumpaankin näistä vaiheista tulee selville jo lähtöasetelmissa: selviytymiskykyiset sovellukset olettavat tunkeutujan omaavan pääsyn järjestelmään sekä kykenevän vaikuttamaan järjestelmän toimintaan.

Selviytymiskykyiset sovellukset sisältävät erilaisia puolustusmekanismeja, jotka voidaan jaoitella seuraavan neljän selviytymiskykyä kuvaavaan strategialuokan puitteissa [23]:

1) *strategiat hyökkäysten estämiseksi:* mm. käyttäjänhallinta, salaus, palomuurit, konfiguroinnin tarkistusmekanismit, järjestelmän päivitys tunnettujen heikkouksien osalta
2) *strategiat hyökkäyksen ja tunkeutumisen havaitsemiseksi sekä järjestelmän tilan ymmärtämiseksi:* IDS-mekanismit kuten järjestelmän käytön profilointi, eheystarkistus, auditointi ja järjestelmän ja verkon valvonta
3) *strategiat toiminnallisuuden elvyttämiseksi onnistuneen tunkeutumisen jälkeen, vaurion rajoittamiseksi, ydinpalveluiden elvyttämiseksi palvelun laatuvaateiden puittiessa ja täydellinen elvyttämisen toteuttamiseksi:* mm. tiedoston-palautusjärjestelmät, vaihtoehtoiset palvelut, vikasietoiset prosessiryhmät, redundantit osajärjestelmät, vaurion eristys, kapasiteetin mukainen adaptaatio sekä kapasiteetin siirtäminen
4) *strategiat järjestelmän selviytymiskyvyn kehittämiseksi*: mm. tunkeutumisjälkien toistuvuus-piirteiden tunnistus, adaptiviiset paketti-suodattimet, lokiinkirjoitus

APOD laajentaa luotettavuus- ja tietoturvanäkökulmaa lisäämällä selviytymiskykyä tuottavia ominaisuuksia tai integroimalla näitä tuottavia sovelluksia QuO väliohjelmistoon.

Kuva 7 sisältää APODin mekanismien yhteenvedon, määrittäen kuinka sekä paikalliset että hajautetut mekanismit toimivat keskitetyn prosessiryhmän koordinaation alaisuudessa.



Kuva 7: APOD strategiat ja mekanismit [24]

APODin strategiat kattavat pääasiassa ensimmäisen, toisen ja kolmannen kohdan aiemmasta selvitytyvien sovellusten määrityksestä. APODin strategioiden tavoitteet ovat [24]:

- *replikointi (outrunning component failures):* vikasietoisuuden tukemiseksi sovellus on kyettävä hajauttamaan replikoimalla useisiin sijainteihin. Tämän lisäksi sovelluksen on taattava bysanttilainen vikasietoisuustaso eli kyettävä sietämään prosessiryhmän jäsenten virheellinen tai muun ryhmän toiminnan vastainen toiminta
- *hyökkääjän ja hyökkäyksen rajoittaminen (attack containment):* diagnostiikan jälkeen sovelluksen on omattava mekanismeja joilla estää tai eristää tunkeutujan toiminta
- *vaihteleva toiminta* (continuous unpredictable changes)*:* sovelluksen on kyettävä hämäämään tunkeutujan järjestelmän analyysiä tekemällä muutoksia omaan toimintaansa

Näiden lisäksi voisi määrittää APODille myös neljännen strategian, joka näkyy selkeästi sen mekanismeissa:

- *omatoiminen valvonta:* sovelluksen on kyettävä monitoroimaan sekä uhkia palvelun laadulle että itse sovelluksen toiminnalle omatoimisesti ja ilmoittamaan mittaustuloksista. Monitoroinnin on tuettava sekä suorien että epäsuorien hyökkäysten havaitsemista

APODin mekanismien rakennus toteutui kahdessa vaiheessa, jotka esitellään seuraavaksi vain sivuten, antaakseen esimerkkiä sovellusten integroinnista osaksi QuO:ta. Ensimmäiseksi tuotettiin tietoturvan ja luotettavuden lisätukea seuraavilla mekanismeilla [25]:

- Tripwire IDS integraatio, jonka puitteissa kaupallinen tiedostojen eheyden tarkistuksesta vastaava sovellus tuottaa tietoa, mikäli alustan tai sovelluksen tiedostoja on muutettu
- FileCounter, räätälöity tiedostolaskuri, joka ylläpitää sovelluksen hakemistojen tiedostolukumääriä
- Erillisten toiminnallisten politiikkojen määrittely, joiden puitteissa voidaan – hyödyntämällä viiveen vaihtelua sekä RTT:a – päätellä järjestelmän joutuneen hyökkäyksen kohteeksi

APOD:n toisessa vaiheessa laajennettiin tukea seuraaville mekanismeille [24]:

- Snort integraatio, jonka myötä toteutukseen saadaan mukaan hyökkäyksen tunnumerkkejä etsivä verkon toimintaa tutkiva IDS -sovellus
- Iptables/Netfilter integraatio palomuuri-filtteröinnin mahdollistamiseksi
- Netstat tcp/ip-pinon tilan seuraamiseksi
- laajennettu määritysrajapintatuki tietoturvaprotokollille (IPSec) sekä VPN-tietoturvatuotteille (Zebedee, Free S/WAN, openSSH)
- RSVP:n tietoturvavahvistetun version (RSVP-SE)

Työn alla oleva laajennus on käyttöönottaa kolmas IDS-järjestelmä, JAM [26], joka hyödyntää oppivia algoritmeja poikkeavuuksien havainnoinnin toteuttamiseksi sekä näistä hyökkäyksen tunnusmerkkien tuottamiseksi. JAMin eräajolähtöinen toiminta tuottaa kuitenkin tämän osalta varmasti ongelmia.

5 Pohdintaa

Poiketen monista reflektiivisten väliohjelmistojen toteutuksista, QuO ei ole tiettyyn sovellusalueeseen sidottu yksittäinen ratkaisu vaan pikemminkin prototyyppejä rakentava suurhanke, jonka lähtökohtana on tuottaa DARPAn järjestelmiin soveltuvia adaptaatiota hyödyntäviä järjestelmiä. Hankkeen projektit pyrkivät toteuttamaan tarpeita vastaavan järjestelmän ja siirtyvät eteenpäin. Tyypillisesti samaan asiaan ei välttämättä palata parantamaan sitä, mikäli kyseessä oli esimerkiksi yksittäisen järjestelmän toteutus. Kuitenkin prototyyppien kehityksen myötä itse QuO väliohjelmisto integroi merkittävimpiä tekniikoita ja mekanismeja liittymäprojekteista, kehittyen piirteissään.

Vuodesta 1997 eteenpäin Quorum –hanke on tuottanut toteutuksia niin reaaliaikaisien sulautettujen järjestelmien, vikasietoisten prosessiryhmien, kaistanhallintamekanismien kuin selviytyvien sovellusten puitteissa. Vaikkakin on epätodennäköistä että QuO:n mekanismeja standardoitaisiin CORBA-standardiin, ovat sen käytännöntoteutukset ja mittaustulokset tuoneet esille suunnitteluratkaisujen heikkoudet ja vahvuudet, lisäten merkittävästi välikerrossovelluksia tutkivan tutkijayhteisön tietoutta ja ehkä omalta osaltaan lisänneet kiinnostusta aihealueeseen.

VIITTEET

[1] B. Chatterjee, M. Sydir, T. Lawrence: “Taxonomy for Quality of Service”, Specifications”, Words 97, IEEE Computer Society, 1997

[2] H. Kopetz (ed): *Real-Time Systems: Design Principles for Distributed Embedded Applications*, 4th printing, Kluwer academic press, 1997

[3] John Zinky, David Bakken, and Richard Schantz: “Architectural Support for Quality of Service for CORBA Objects”, Theory and Practice of Object Systems, vol. 3, no. 1, 1997.11th European Conference on Object-Oriented Programming, June 1997.

[4] Rodrigo Vangeras , John Zinky , Joseph Loyall, David Karr, Richard Schantz ,David Backen: “QuO’s Runtime Support for Quality of Service in Distributed Objects”, BBN Technologies, 1998

[5] Richard Schantz, John Zinky, David Karr, David Bakken, James Megquier, Joseph Loyoll,: “An object-level Gateway Supporting Integrated-Property Quality of Service”, Proceedings of ISORC ’99, May 2-5 1999

[6] Joseph Loyall, David Backen, Richard Schantz. Douglas C. Schmidt, John Zinky, David Karr, Rodrigo Vangeras, Kenneth Anderson: “QoS Aspect Languages and their runtime integration”, Vol. 1511, Springer-Verlag. Proceedings of the Fourth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR98), 28-30 May 1998, Pittsburgh, Pennsylvania

[7] Timothy Harrison, David Levine, Douglas C. Schmidt: “Design and Performance of Real-Time Event Service”, OOPSLA ’97, October 1997

[8] Phil Wang, Yechiam Yemini, Danilo Florissi, Patricia Florissi: “QoSME: Towards QoS management and quarantees”, University of Columbia, 1998

[9] DIRM project home pages, http://www.dist-systems.bbn.com/projects/DIRM/ [27.05.2003]

[10] Yansong Ren, David Bakken, Tod Courtney, Michel Cukier, David Karr, Paul Ruben, Chetan Sabis, William Sanders, Richard Schantz, Mouna Seri: “AquA: an adaptive architecture that provides dependable distributed objects”, BBN Technologies, 1997

[11] Douglas C. Schmidt: “Reactor: an object behavioral pattern for concurrent event demultiplexing and dispatching”, 1st PLoP Conference, Illinois, 1994

[12] Erich Gamma, Richard Helm, Ralph Johnsson, John Vlissides, *Design Patterns: Elements of reusable software*, Addison-Wessley, 1996

[13] Joe Loyall: “Aspects In Real-time Embedded Systems (AIRES)”, BBN technologies, PCES PI Meeting, 24-26 October 2001, Mesa, AZ

[14] BBN, *QuO Toolkit Reference Guide*, QuO release 3.0.11, BBN Technologies, October 2002

[15] BBN, *QuO Toolkit Users Guide*, QuO release 3.0.11, BBN technologies, October 2002

[16] OMG, *Common Architecture Request Broker Architecture*, Version 3.0.2, December 2002

[17] Priya Narasimhan, Louise E. Moser, P.M. Melliar-Smith: “Using Interceptors to enhance CORBA”, IEEE Computer July 1999

[18] “Using the object wrappers” lähteessä *Visibroker for Java Programmers Guide*, Version 4.0, Visigenic Software, 1996

[19] Svend Frølund, Jari Koistinen: ”Quality of Service Specification in Distributed Object Systems Design”, "{IOP}/{BCS} Distributed Systems Engineering Journal, December 1998

[20] Gordon Blair et. al. “The Design and Implementation of OpenORB 2”, 2001 http://dsonline.computer.org/0106/features/bla0106_print.htm [15.05.2003]

[21] D. Karr C. Rodrigues, J. Loyall JP, R Schantz, Y. Krishnamurthy, I Pyarali, D Schmidt: “Application of the QuO Quality-of-Service Framework to a Distributed Video Application”, BBN, Proceedings of the International Symposium on Distributed Objects and Applications, September 18-20, 2001, Rome, Italy

[22] Y. Krishnamurthy, V.Kachroo, DA Karr, C. Rodrigues , J P Loyall, R. E. Schantz, D. C. Schmidt: “Integration of QoS-Enabled Distributed Object Computing Middleware for Developing Next-Generation Distributed Applications”, Proceedings of the ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems June 18, 2001, Snowbird, Utah

[23] Robert Ellison et al: “Survivability: Protecting your critical systems”, IEEE Internet Computing, November/December 1999

[24] Michael Atighetchi, Partha P. Pal, Chris Jones, Paul Rubel, Richard E. Schantz, Joseph P. Loyall, and John A. Zinky: “Building Auto-Adaptive Distributed Applications: The QuO-APOD Experience “, The 3rd International Workshop on Distributed Auto-adaptive and Reconfigurable Systems, in conjunction with the 23rd International Conference on Distributed Computing Systems, May 19-22, 2003, Providence, Rhode Island, USA

[25] Joseph Loyall, Partha Pal, Richard Schantz, Franklin Webber: “Building Adaptive and Agile Applications Using Intrusion Detection and Response”, Proceedings of NDSS 2000, the Network and Distributed System Security Symposium, February 2-4 2000, San Diego, CA

[26] University of Columbia, JAM Project: http://www1.cs.columbia.edu/~sal/JAM/ PROJECT/ [27.05.2003]