hyväksymispäivä arvosana

arvostelija

Palvelulaatutietoinen väliohjelmisto

Pasi Vettenranta

Helsinki 19.12.2003

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta/Osasto — Fakultet/Sektion — Faculty	Laitos — Institution — Department

Tekijä — Författare — Author
Pasi Vettenranta

Työn nimi — Arbetets titel — Title
Palvelulaatutietoinen väliohjelmisto

Oppiaine — Läroämne — Subject

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
	19.12.2003	

Tiivistelmä — Referat — Abstract

Viime vuosina väliohjelmistot ovat kehittyneet tukemaan hajautettujen ohjelmistojen toimintaa laajoissa, muuntuvissa ja epävarmoissa tietokoneverkoissa. Tämän kehityksen takia tulevaisuuden väliohjelmistojen odotetaan tarjoavan käyttäjilleen toimivia palvelulaadunhallintaominaisuuksia (Quality of Service, QoS). Tässä artikkelissa pohditaan millainen järjestelmä täyttää vaadittavat palvelulaadunhallintaominaisuudet täyttäviä. Lisäksi artikkelissa tutustutaan yhteen palvelulaatutietoiseen (Qos-aware) väliohjelmistoprojektiin, QuO-projektiin.

QuO-projektin katsauksessa on tarkoitus esitellä erään palvelulaatutietoisen väliohjelmiston toteutusta ja ajonaikaista käyttäytymistä. Tarkastelun kohteena on miten kyseinen järjestelmä hoitaa eri palvelupyyntöjen neuvottelutilanteet, sekä miten järjestelmä hoitaa eri osiensa välisen kommunikoinnin. Lisäksi tarkastellaan miten järjestelmä valvoo ja takaa käyttäjilleen että näiden tekemät palvelutilaukset pysyvät voimassa ulkoisten vaikutteiden muokatessa järjestelmän toimintaympäristöä.

Avainsanat — Nyckelord — Keywords
Middleware, QoS, QuO, Qosket

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Sisältö

1 Johdanto

Muuntuvissa ja epävarmoissa tietokoneverkoissa hyvin toimivat hajautetut järjestelmät ovat harvassa. Niiden toteuttaminen on vaativaa ja kallista. Hajautettuja järjestelmiä kehitetään ideaalisia oloja silmällä pitäen, koska muuntuvan ympäristön käsitteleminen on liian hankalaa ja kallista. Usein vain on, että hajautettujen järjestelmien toimintaympäristöt ovat kaukana näistä ideaaliympäristöistä. Vaikka järjestelmää suunniteltaessa yritettäisiin varautua tiettyihin tapahtumiin, niin uusia epätavallisia ja odottamattomia tapahtumia sattuu jatkuvasti. Onkin helpompi ja halvempi olettaa, että tätä meidän omaa järjestelmäämme ajetaan ainoastaan tietynlaisissa, järjestelmälle ideaaleissa olosuhteissa.

Kun näin suunniteltu hajautettu järjestelmä sitten lopulta siirretään tuotantokäyttöön, ajaudutaan suurella todennäköisyydellä loputtomalta vaikuttavaan paikkauskierteeseen. Odottamattomia tapahtumia käsittelevää koodia lisätään järjestelmään kerta toisensa jälkeen. On havaittu, että vanhemmissa jo vuosia tuotantokäytössä olleissa hajautetuissa järjestelmissä suurin osa, joskus jopa 90%, järjestelmän koodista on kirjoitettu käsittelemään järjestelmän epätavallisia poikkeustilanteita [ZBS95].

Miten siis toteuttaa helposti ja edullisesti järjestelmä, joka pystyisi omalla toiminnallaan sopeutumaan muuntuviin olosuhteisiin ja jonka muuntaminen olisi yksinkertaista ja sitä kautta myös edullista. ATM ja RSVP tarjoavat ohjelmille mahdollisuutta käyttää palvelulaatuominaisuuksia. Valitettavasti nämä ominaisuudet on suunniteltu enemmän verkon kuin ohjelman ominaisuuksia silmällä pitäen. Hajautetulle järjestelmälle on paljon mielekkäämpää saada tieto siitä kuinka

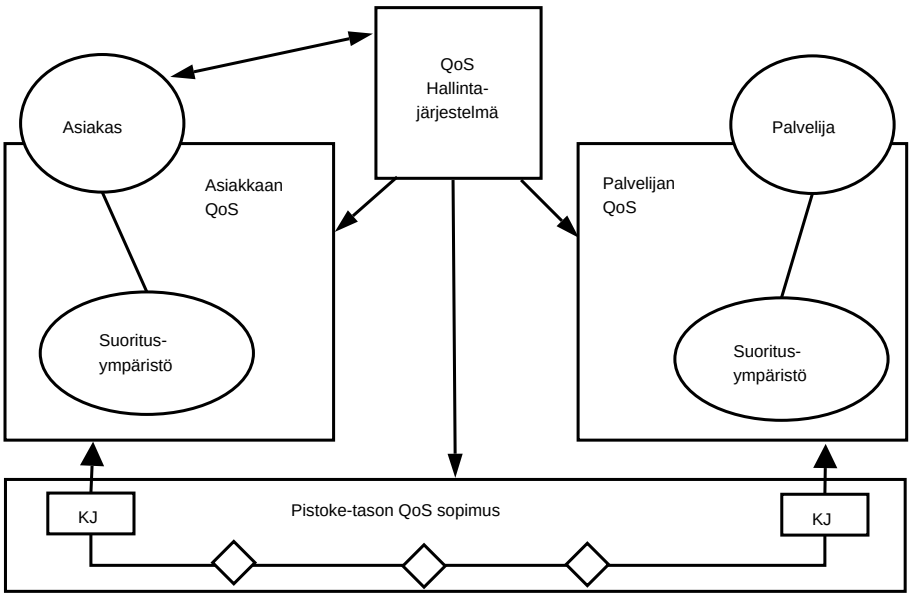
monta minkäkinlaista etämetodia sekunnissa järjestelmä ehtii suorittamaan, kuin se että kuinka monta bittiä sekunnissa verkko pystyy kuljettamaan asiakaspään ja olion haltijan välillä [ZBS95]. Tämän kaltainen uusi palvelulaadunmäärittely ohjelman kannalta auttaa sekä ohjelmoijaa, että järjestelmää itseään pyytämään / vaatimaan riittävää ja oikean kokoista palvelulaatusopimusta.

Seuraavassa luvussa mietitään millä tavoin pystytään parantamaan hajautettujen järjestelmien kykyä sopeutua muuntuviin oloihin. Kolmannessa luvussa esitellään QuO-projektin palvelutietoisen väliohjelmiston rakennetta ja toimintaa. Neljännessä luvussa tutustutaan menetelmään, jolla Quo-ominaisuuksia pystytään kapseloimaan omiksi uudelleen käytettäviksi kokonaisuuksikseen.

2 Palvelulaatutietoinen väliohjelmisto

Palvelulaatutietoisten (QoS-aware) ohjelmien kirjoittaminen on hyvin hankalaa. Vaadittava tieto on hajallaan ympäri järjestelmää ja usein vielä sen hankalan tyyppisessä muodossa. Tällaista ohjelmistoa kirjoittavan ohjelmoijan on pystyttävä koordinoimaan keskenään asiakas- ja palvelinpään ajonaikainen järjestelmä, käyttöjärjestelmien tarjoamien neuvottelujärjestelmän avulla hallittava pistoketason palvelulaatusopimukset (ks. Kuva 1) [ZBS95].

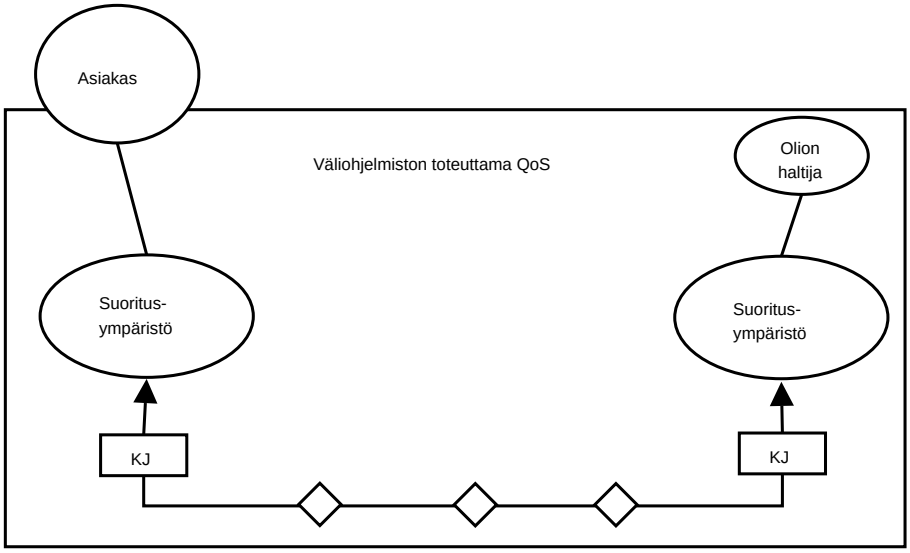
Oliosuuntautunut näkemys palvelulaatutietoisesta hajautetusta järjestelmästä yksinkertaistaa palvelulaadun hallintaan kykenevän järjestelmän toteuttamista. Palvelunlaatutietoinen väliohjelmistoarkkitehtuuri suosii yleispätevillä komponentti-



Kuva 1: Kuvaus asiakas/palvelin -mallin mukaisesta palvelulaadunhallintajärjestelmästä [ZBS95].

malleilla suunniteltuja ohjelmistoja [NXWL01]. Palvelulaadun hallinnasta vastaa sitä juuri tarkoitusta varten rakennettu hajautettu komponentti (ks. Kuva 2), jonka tehtäviin kuuluu hallita ja valvoa kuinka hyvin järjestelmän sisäinen kommunikointi tapahtuu sekä mahdollistaa järjestelmän mukautuva päätöksen teko ja toiminnallisuus äkisti muuntuvissa tilanteissa ja tapahtumissa [SLAP02]. Tällainen järjestelmä kykenee tarjoaakaan käyttäjilleen yksinkertaisemman ja ohjelmalähtöisemmän tavan käyttää verkkotason palvelulaadun hallintaa.

Väliohjelmistoarkkitehtuuriin perustuvan palvelunlaatutietoisen järjestelmän voidaan katsoa koostuvan komponenteista. Komponentti jakona voidaan käyttää järjestelmän resurssien- ja palvelunlaadunhallintaa. Palvelunlaatutietoinen resurssienhallintajärjestelmä käsittää resurssien välittäjä-, resurssien mukauttaja- (adaptor) ja resurssien tarkkailija -komponentit [NXWL01]. Näiden komponent-



Kuva 2: Kuvaus olio-mallin mukaisesta palvelulaadunhallintajärjestelmästä [ZBS95].

tien vastuulla on resurssien varaaminen, niiden toimeenpano, mukauttaminen sekä tarkkailu. Tämä komponentti rakennetaan käyttöjärjestelmien tarjoamien palveluiden päälle. Palvelunlaatutietoinen palvelulaadun hallintajärjestelmä koostuu useista väliohjelmistokomponenteista, joita yleisesti kutsutaan nimityksellä QoS-Proxy. Sen päätökset ja toiminnot perustuvat alla olevan resurssienhallintajärjestelmän sille antamiin raportteihin [NXWL01].

3 Quality Object, QuO

Quality Object (QuO) on palvelulaatutietoisten hajautettujen järjestelmien toteutusta varten kehitetty ohjelmointikehys (framework). Se mahdollistaa muuntuviin palvelulaatutilanteisiin sopeutuvien hajautettujen järjestelmien toteutta-

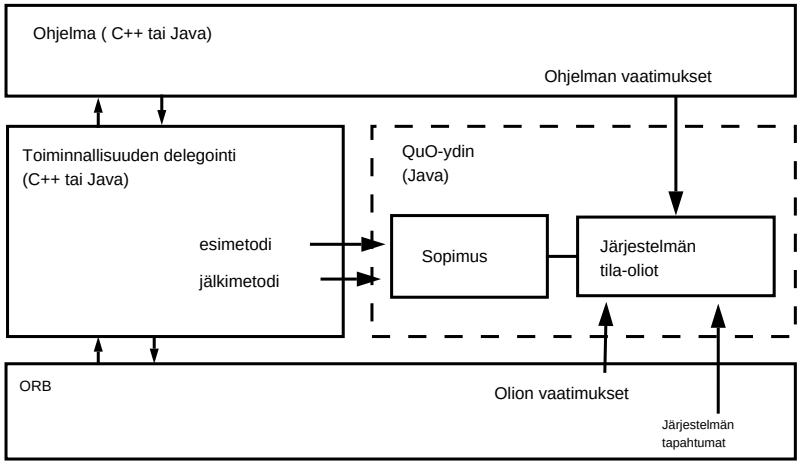
misen. Sen avulla pyritään toteuttamaan entistä vikasietoisempia hajautettuja järjestelmiä, saamaan aikaan parempi kontrolli siitä, kuinka ohjelmistot mukautuvat järjestelmän alla tapahtuviin muutoksiin, sekä tukea sopeutuvaisten mekanismien uudelleen käyttöä (ks. Kuva 3) [VZL+98].

QuO tarjoaa hajautetun järjestelmän hallintaan joukon aspektikieliä. Kyseistä joukkoa kutsutaan yleisesti laadunkuvaus kieliksi (Quality Description Languages, QDL) [LBS+98]. Niiden avulla järjestelmälle määritellään palvelulaatutiloja, järjestelmän resursseja sekä mekanismeja. Niiden avulla myös mitataan ja hallitaan palvelulaadun toimintaa, sekä luodaan järjestelmän käyttäytymismallit. Käyttäytymismallien avulla järjestelmä voi muuntaa toimintaansa ajonaikaisesti [PLS+00].

3.1 QuO rakenne

Tavallisessa CORBA-ohjelmassa, asiakas suorittaa etämetodikutsun seuraavalla tavalla. Asiakas suorittaa kutsun oman oliorajapintansa läpi. Rajapinnan tarjoaa asiakaspäässä sijaitseva oliokutsun välittäjä (Object Request Broker, ORB). Se ottaa kutsun vastaan ja välittää sen kohdeolion isännälle. Isäntä suorittaa kutsun ja palauttaa suorituksen omalle ORB:lleen. Isännän ORB välittää vastauksen takasin asiakaskoneen ORB:lle, joka välittää sen edelleen asiakkaalle. Asiakkaan kannalta näyttää että kyseessä on puhtaasti funktionaalinen metodikutsu [ZBS97].

QuO muokkaa hieman edellä esitettyä tapaa suorittaa etämetodikutsuja. Ennen jokaista suoritettavaa etämetodikutsua QuO tarkastaa alla olevan verkon palve-



Kuva 3: QuO -pohjaisen järjestelmän komponentit [VZL+98].

lutasotilan, tekee päätelmät sen tilan perusteella ja toteuttaa etämetodikutsut parhaimammaksi katsomallaan tavalla. Jos QuO:n tekemät päätelmät on syytä viestittää myös asiakaspään tietoon niin QuO tekee sen käyttämällä siihen takaisinkutsua (callback) tai nostamalla poikkeuksen [VZL+98].

QuO:n suunnittelijan tehtävä on kirjoittaa järjestelmän toiminnallisuus käyttäen QuO: tarjoamia aspektikieliä (QDL). Niiden avulla suunnittelija määrittelee palvelulaadun tilat, ohjelmiston muuntuvan käyttäytymisen mallit ja ne järjestelmän osat ja komponentit, joiden tilaa ja toimintaa QuO:n tulee valvoa. Lopullinen tuotos nidotaan yhteen koodigeneraattoreiden avulla. Ne generoivat QDL-kielistä koodia ja linkittävät sen yhteen asiakaspäänkoodin, QuO-ytimen sekä CORBA-tynkien ja rankojen kanssa [ZBS97].

QuO -pohjainen järjestelmä koostuu asiakaan osoiteavaruudessa sijaitsevasta delegointi osasta sekä QuO-ytimestä. Seuraavaksi esitellään QuO-pohjaisen järjes-

telmän komponentit ja niiden toimintaperiaatteet.

3.1.1 Olio tehdas

Olio tehdas sijaitsee QuO-ytimessä. Sen tehtävänä on luoda ja alustaa sopimukset sekä järjestelmän tila-oliot. Toistaiseksi kaikki nämä oliot luodaan QuO-järjestelmän käynnistyksen yhteydessä, mutta tulevaisuudessa järjestelmän tulisi pystyä luomaan dynaamisesti sopimuksia ja järjestelmän tila-olioita [VZL+98], [ZBS97].

3.1.2 Sopimus -olio

Sopimus olion tehtävänä on pitää tieto järjestelmän palvelulaaduntilasta ja käynnistää järjestelmän toimintojen mukauttaminen, palvelulaadun tilan muuttuessa. Jokainen sopimus käsittää joukon sisäkkäisiä alueita, jotka määrittelevät järjestelmän tilan suhteessa sen hetkiseen palvelulaatutasoon. Sopimus-olio sisältää asiakkaan pyytämän, etäolion vaatiman ja järjestelmän havaitseman palvelulaaduntason. Siirtyminen alueesta toiseen voi käynnistää järjestelmän muutosvaiheen käyttäytymisen [VZL+98],[ZBS97].

3.1.3 Delegoija

Delegoija tarjoaa asiakaspäälle etäolion kanssa identtisen metodirajapinnan. Metodien toiminnallisuuden sijasta, se sisältää toiminnallisuuden voimassa olevien sopimusalueiden tilan tarkasteluun ja tämän pohjalta tehtävän toiminnallisuuden

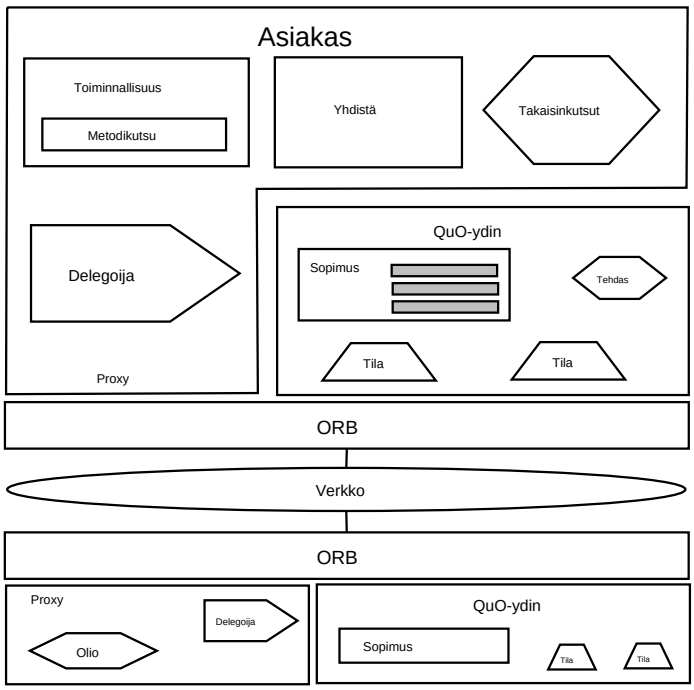
päättelyyn. Kun asiakas kutsuu etämetodia, välitetään kutsu oikeasti asiakaspään delegoijan käsiteltäväksi. Se tarkastaa voimassa olevien sopimusten tilan. Saamiensa tietojen perusteella se päättelee, miten toimia etämetodikutsun kanssa. Delegoija voi esimerkiksi valita jonkin vaihtoehtoisista välitystavoista. Jos palvelulaatu romahtaa voi delegoija jättää paketin odottamaan, estää sen lähetyksen tai ilmoittaa siitä asiakasohjelmalle nostamalla poikkeuksen. Delegoija suorittaa samanlaisen tarkistuksen kun etämetodikutsu palaa [VZL+98],[ZBS97].

3.1.4 Järjestelmän tilaolio

Järjestelmän tilaolio toimii rajapintana sopimusten ja resurssien, mekanismien, muiden olioiden ja ORB:n välillä. Niitä käytetään mittaamaan ja kontrolloimaan palvelulaatua (QoS). Jotkut järjestelmän tila-oliot voivat käynnistävät sopimuksen tarkastusrutiineja havaitessaan muutoksia joissain valvomissaan järjestelmän komponenteissa. Toiset järjestelmän tilaoliot, joiden valvomat järjestelmän komponenttien tilat muuttuvat usein, tyytyvät informoimaan sopimusoliota järjestelmässä tapahtuneista muutoksista vain pyydettäessä [VZL+98],[ZBS97].

3.2 QuO-pohjaisen järjestelmän toimintaperiaate

QuO-pohjainen järjestelmä asiakkaan halutessa suorittaa etämetodikutsua toimii seuraavalla tavalla (ks. Kuva 3). Aluksi asiakas suorittaa yhdistä-metodin (connect). Se vastaa toiminnallisuudeltaan CORBA:n liitä-metodia (bind), mutta lisää mukaan asiakaspään palvelulaatu vaatimuksen [VZL+98]. Nyt QuO-järjestelmä on tietoinen asiakkaan haluamasta palvelulaadusta. Tämän jälkeen asiakas suo-



Kuva 4: Tarkempi kuvaus QuO-pohjaisen järjestelmän rakenteesta [LSZB98].

rittaa haluamansa etämetodikutsun. Etämetodin suorittamisen sijaan, suorittaa järjestelmä itseasiassa paikallisen delegoijaolion vastaavan metodin. Paikallisella delegoijalla on etäolion kanssa identtinen metodirajapinta. Saatuaan metodikutsun delegoija käynnistää sopimuksenarviointirutiinin. Tällöin suoritusvuoro siirtyy sopimusoliolle. Se lähettää kyselyt kaikille järjestelmän tilaolioille, jotka palauttavat tiedot valvomistaan järjestelmän komponenteista ja arviot niiden toiminnoista tulevaisuudessa [ZBS97].

Sopimus käsittää joukon sisäkkäisiä sopimusalueita, jotka kuvaavat järjestelmän kaikkia merkityksellisiä palvelunlaadun tiloja. Jokainen näistä alueista on määritelty riippumaan jostain järjestelmän tilaolion tarjoamasta tiedosta. Sopimusolio tarkistaa mitkä riippuvuussuhteet ovat voimassa ja lähettää niistä listan dele-

goijalle. Tämä lista kuvaa järjestelmän sen hetkistä palvelulaadun tilaa voimassa olevalla sopimuksella. Jos sopimuksen arviointi aiheutti sopimusalueen vaihdon eli aktiivisten alueiden lista ei ole samanlainen, kuin se edellisellä kerralla oli kun sopimusta arvioitiin, saattaa järjestelmä käynnistää muutosvaiheen. Muutosvaihe käsittää siirtymisen toisista sopimusalueista toisiin, tästä aiheutuvan mahdollisen asiakkaan informoimisen takaisinkutsulla tai palvelulaaduntilan muokkaamisen järjestelmän tilaolioiden avulla. [ZBS97].

Normaalisti kun palvelunlaatu on hyväksyttävää, siirtää delegoija etämetodikutsun etäolion suoritettavaksi. Jos palvelunlaatu ei vastaa sovittua delegoija voi puskuroida tai blokata metodinkutsun tai jopa nostaa poikkeuksen informoidakseen asiakasta tapahtuneesta.

4 Qosket

Yksi QuO:n päämääristä on erottaa järjestelmän toteutus ohjelman toteutuksesta ja aikaan saada näin erillisiä, helpommin ylläpidettäviä ja uudelleen käytettäviä komponentteja. Lopullinen päämäärä tässä erottelussa on se, että järjestelmän käyttäytymismallit voidaan kapseloida omaksi uudelleen käytettäväksi toteutukseen, jopa täysin irralleen niitä käyttävistä ohjelmista. Qosket on tällainen kapseloitu yksikkö. Niitä käytetään kokoamaan yhteenpaikkaan kaikki ohjelmasta riippumaton järjestelmän toiminnallisuutta varten toteutetut määrittelyt ja oliot [SLAP02].

Qosket on komponentti, joka kapseloi järjestelmän rajapinnat, sopimukset, tilaoliot, takaisinkutsuoliot, määrittelemättömän mukautuvan käytöksen ja uudelleen käytettävään järjestelmän toiminnanosaan liittyvän koodin omaksi uudelleen käytettäväksi komponentiksi. Qosket:ia voi käyttää kahdella tavalla. Joko ohjelmalle yleisesti käytössä näkyvänä komponenttina tai yhdessä QuO:n delegoijaolion kanssa. Myös kummankin vaihtoehdon samanaikainen käyttö onnistuu. Tämän takia Qosket tarjoaa ulos kaksi erilaista rajapintaa, muokkaaja- ja delegoijarajapinnan, vastaamaan komponentin käyttötarkoituksia [SLAP02].

Qosketin muokkaajarajapinta on toteutettu ohjelmien toteuttajia varten. Se tarjoaa välineet palvelulaadun mittaukseen ja kontrollointiin, sekä pääsyn kiinni qosketin sisältämiin mukautumis ominaisuuksiin, siten että niitä on mahdollisuus käyttää missä tahansa ohjelmassa. Delekointirajapinta tarjoaa yksinkertaistetun tavan määritellä funktioita, joiden avulla delegoijan mukautumisen toteutus on yksinkertaisempaa [SLAP02].

5 Yhteenveto

Hajautettujen järjestelmien on pystyttävä mukautumaan muuntuviin olosuhteisiin, jotta niiden käyttäminen muuntuvissa ympäristöissä olisi mahdollista. Palvelulaadunhallintaan kykenevän hajautetun järjestelmän toteutus on hankalaa. ATM ja RSVP tarjoavat ohjelmoijille rajapinnan käyttää palvelulaatu ominaisuuksia. Valitettavasti niiden käyttö on hankalaa ja halutun kaltaisten järjestelmien toteuttamiseen vaadittava tieto on jakautunut useaan eri paikkaan, jotta niiden toteuttaminen olisi yksinkertaista ja helppoa. Oliosuuntainen ajattelu voi-

si olla ratkaisu tähän ongelmaan. Väliohjelmiston hoitama palvelunlaadunhallinta yksinkertaistaisi abstrahoisi ympäristöönsä mukautuvan hajautetun järjestelmän toteuttamista. Se eristäisi järjestelmän komponentteja siten, että niiden uudelleen käyttö olisi mahdollista. Tämä nopeuttasi ja helpottaisi vastaavien järjestelmien rakentamista. QuO-projekti osoittaa, että on mahdollista luoda palvelulaatutietoinen väliohjelmisto ja Qosket osoittaa, että QuO:n ominaisuudet pystytään eristämään omiksi uudelleen käytettäviksi kokonaisuuksiksi. Vastaavanlaisten järjestelmien avulla toimintaympäristöönsä mukautuvien hajautettujen järjestelmien toteuttaminen on entistä yksinkertaisempaa ja edullisempaa.

Lähteet

LBS+98 Loyall, J. P., Bakken, D. E., Schantz, R. E., Zinky, J. A., Karr, D. A., Vanegas, R. ja Anderson, K. R., Qos aspect languages and their runtime integration. *Selected Papers from the 4th International Workshop on Languages, Compilers, and Run-Time Systems for Scalable Computers*. Springer-Verlag, 1998, sivut 303–318.

LSZB98 Loyall, J. P., Schantz, R. E., Zinky, J. A. ja Bakken, D. E., Specifying and measuring quality of service in distributed object systems. *Proceedings of the 1st International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*, 1998, URL `citeseer.nj.nec.com/loyall98specifying.html`.

NXWL01 Nahrstedt, K., Xu, D., Wichadakul, D. ja Li, B., Qos-aware middleware for ubiquitous and heterogeneous environments, ieee communications magazine, 2001. URL `citeseer.nj.nec.com/article/nahrstedt01qosaware.html`.

PLS+00 Pal, P., Loyall, J., Schantz, R., Shapiro, R. ja Megquier, J., Using qdl to specifyt qos aware distributed (qoo) application configuration.

Proceedings of The 3rd IEEE International Symposium on Object-orient Real-time distributed Computing (ISCOR 00), March 2000.

SLAP02 Schantz, R. E., Loyall, J. P., Atighetchi, M. ja Pal, P. P., Packaging quality of service control behaviors for reuse. *Symposium on Object-Oriented Real-Time Distributed Computing*, 2002, sivut 375–385.

VZL+98 Vanegas, R., Zinky, J. A., Loyall, J. P., Karr, D. A., Schantz, R. E. ja Bakken, D. E., Quo's runtime support for quality of service in distributed objects. *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98)*, 1998.

ZBS95 Zhinky, J., Bakken, D. ja Schantz, R., Overview of quality of service for distributed objects, 1995. URL `citeseer.nj.nec.com/zhinky95overview.html`.

ZBS97 Zinky, J. A., Bakken, D. E. ja Schantz, R. E., Architectural support for quality of service for CORBA objects. *Theory and Practice of Object Systems*, 3,1(1997). URL `citeseer.nj.nec.com/zinky97architectural.html`.