Avoimen järjestelmän infrastruktuuri ja sovelluskehitysympäristö

Markus Kinni

Avoimien hajautettujen järjestelmien-seminaari
Helsingin yliopisto
Tietojenkäsittelytieteen laitos
2. helmikuuta 2000 Helsinki

Avoimen järjestelmän infrastruktuuri ja sovelluskehitysympäristö

Avoimien hajautettujen järjestelmien tutkimusseminaari
Helsingin yliopisto
Tietojenkäsittelytieteen laitos
Markus Kinni
2. helmikuuta 2000 Helsinki

Avoimen järjestelmän infrastruktuuri ja sovelluskehitysympäristö-artikkeli on kirjoitettu Helsingin yliopistolla pidettävään Avoimien hajautteujen järjestelmien tutkimusseminaariin. Artikkelissa kuvataan avoimen järjestelmän infrastruktuuria ja palveluita, jotka muodostavat alustan sovelluskehykselle.

ODP:n viitemallissa määritellään abstraktilla tasolla avoimen hajautetun järjestelmän toteutusympäristö ja sen ominaisuudet. Mallin pohjalta on kehitetty erilaisia väliohjelmistoja, joiden avulla on voidaan toteuttaa avoimuuden mahdollistavat rajapinnat ja samalla peittää järjestelmän tekninen toteutus. Erilaiset komponenttimallit sekä nopeasti kehittyvät väliohjelmistot tarjoavatkin nykyisin erilaisia ratkaisumalleja ja toteutusvaihtoehtoja avoimien järjestelmien kehittämiselle. Artikkelissa esitetään väliohjelmistoista erilaisiin toteutuksiin perustuva jaottelu. Väliohjelmistojen tarjoamia erilaisia ominaisuuksia ja palveluita kuvataan sekä lisäksi otetaan esille seikkoja, jotka vaikuttavat sovelluskehitykseen.

Avoimet hajautetut järjestelmät pohjautuvat monitasoarkkitehtuurin. Teknisen perusinfrastruktuurin päälle rakennetaan sovelluskehys, joka määrittelee rajapinnat eri palveluihin. Artikkelissa otetaan esimerkiksi J2EE:n tarjoama sovelluskehitysympäristö ja sen tarjoamat rajapinnat, mutta lisäksi arvioidaan mahdollisuuksia hyödyntää kieliriippumattoman ympäristön tarjoamia palveluita.

Aiheluokat: C.2.4, D.2.6
Avainsanat: avoin järjestelmä, monitasoarkkitehtuuri, väliohjelmistot, sovelluskehys

Sisältö

1 Johdanto

Avoin järjestelmä ymmärretään yleensä kattavaksi ja yhdenmukaiseksi joukoksi kansainvälisiä standardeja, jotka määrittelevät rajapinnat, palvelut ja tuettavat formaatit yhteentoimivuuden ja liittettävyyden aikaansaamiseksi järjestelmien, tiedon ja ihmisten välillä [ODP0198]. Ylemmän tason rajapinnoilla pyritään nostamaan abstraktiotasoa ja samalla peittämään allaoleva toteutus. Avoimen hajautetun järjestelmän (*engl. federated system*) toteutuksissa rajapinnoilla peitetään järjestelmien välisen kommunikaation toteutus sekä pyritään lisäämään yhteenliitettävyyttä avoimilla rajapinnoilla.

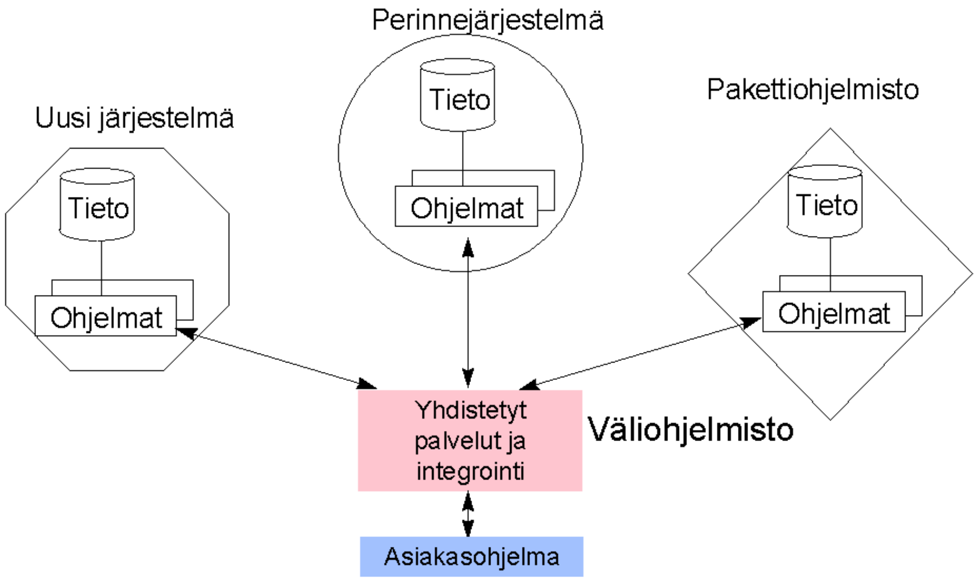
Avoimista rajapinnoista esimerkkinä voidaan mainita rinnakkaisohjelmoinnin rajapinnat erilaisiin prosessoriarkkitehtuurihin ja muistinhallintaan, etäproseduurikutsun rajapinta peittämään tiedon esitystapa järjestelmien välisessä kommunikoinnissa, väliohjelmistojen (*engl. middleware*) tarjoama rajapinta erilaisiin käyttöjärjestelmä palveluihin sekä tiedonhallintajärjestelmien rajapinnat peittämään erilaiset informaatiomallit tietokannoissa [ODP0198]. Avoimissa järjestelmissä rajapinnnat pyritään toteuttamaan väliohjelmistojen avulla, joten nykyisin löytyy erilaisia väliohjelmistoja edelläkuvattujen rajapintojen toteuttamiseen. Jotta järjestelmä olisi avoin ja sen tarjoamia palveluita pystyttäisiin hyödyntämään, pitää rajapinnat julkaista. Julkaistut rajapinnat sijoitetaan erillisiin kuvauskantoihin (*engl. repository*), joiden kautta palveluja voidaan käyttää.

Julkisten rajapintakuvauksien tarjoamista palveluista ja niiden toiminnallisuudesta on saatava informaatiota, jota tarjotaankin ODP:n (*engl. Open Distributed Processing*) viitemallissa RM-ODP (*engl. ODP reference model*). Viitemallin avulla pyritään helpottamaan järjestelmäintegraation toteuttamista heterogeenisessä ympäristössä ja erillisten organisaatioiden välillä. ODP:n standardeilla pyritään takaamaan, että järjestelmiä kehitettäessä ohjelmistot ovat liitettäviä ja yhteentoimivia, järjestelmäintegraation toteuttaminen mahdollista erilaisissa arkkitehtuureissa ja järjestelmäkehitys toteutettavissa dynaamisesti ajonaikana. Lisäksi järjestelmien palvelun laatu QoS (*engl. Quolity of Service*) ja riittävät tietoturvaominaisuudet on pystyttävä takaamaan sekä mahdollistamaan tuntumattomuus erilaisille kommunikointitoteutuksille [ODP0198].

ODP:n viitemallissa on määritelty abstraktilla tasolla toteutusalusta, jota pyritään konkretisoimaan luvussa 2 esiteltävillä hajautetun monitasoarkkitehtuuriin (*engl. multitier architecture*) pohjautuvan järjestelmän kehittämisessä huomioitavilla ominaisuuksilla. Luvussa 3 esitellään erilaiset väliohjelmistot ja niiden palveluita, joita hyödynnetään avoimien hajautettujen järjestelmien toteuttamisessa. Avoimien järjestelmien sovelluskehitysympäristöä käsitellään luvussa 4. Asiat tuodaan esille pääasiassa J2EE:n (*engl. Java 2 Platform, Enterprise Edition*) tarjoaman alustan ja rajapintojen päälle kehitetyn sovelluskehyksen (*engl. framework*) avulla. Luvussa 5 tehdään yhteenveto avoimien hajautettujen järjestelmien infrastruktuurin rakentamisessa ja sovelluskehityksessä huomioitavista seikoista.

2 Avoimen hajautetun järjestelmän infrastruktuuri

Sovellusarkkitehtuurien nykysuuntauksena on komponenttipohjaiset ja monitasoisuutta käyttävät ratkaisut, joissa käytetään ylemmäntason rajapintoja kuvaamaan järjestelmäinfrastruktuurin tarjoamat palvelut. Hajautetun järjestelmän taustalla on monesti käytössä tiukasti nykyiseen järjestelmäinfrastruktuuriin sidottuja perinnejärjestelmiä (*engl. legacy system*), jolloin monitasoisuuden käyttöönotolla halutaan hyödyntää perinnejärjestelmien tarjoamia palveluita luomalla liityntärajapinnat niihin. Ratkaisu mahdollistaa samalla uusien palveluiden käyttöönoton ja yhtenäisen rajapinnan julkaisemisen taustalla olevasta toteutuksesta riippumatta. Nykyisin järjestelmäkehitys toteutetaankin ympäristössä, jossa järjestelmäintegraatio on tärkeässä roolissa, sillä uudet järjestelmät tehdään yleensä olemassaolevien järjestelmien ja infrastruktuurin päälle. Kuvassa 1 näkyy väliohjelmiston sijoittuminen monitasoarkkitehtuurissa. Väliohjelmisto tarjoaa rajapinnat ja palvelut, jotka voidaan julkaista. Rajapinnan kautta järjestelmän palveluita käyttävänä asikasohjelmana voi toimia joko perinteinen käyttöliittymä tai avoimien järjestelmien tapauksessa jokin ulkopuolinen järjestelmä.



Kuva 1: *Väliohjelmiston rooli järjestelmäinfrastruktuurissa*

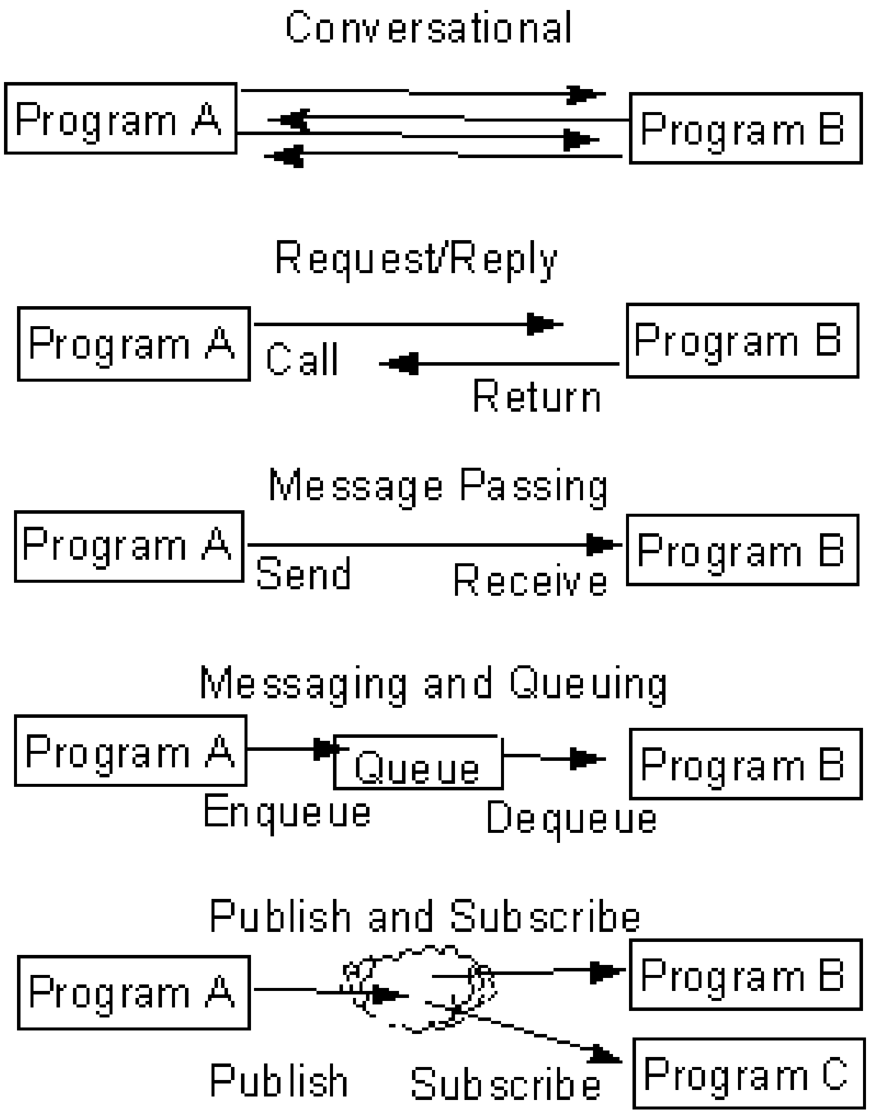
2.1 Arkkitehtuurille asetettavia vaatimuksia

Monitasoarkkitehtuurin päälle kehitetään jatkuvasti uusia järjestelmiä, jolloin suunnittelussa on huomioitava skaalautuvuus lisääntyvään määrään tapahtumia. Tapahtumakäsittely toteutetaan hajautetussa ympäristössä, koska uusissa toteutuksissa tarvittavat toiminnot ovat hajallaan eri järjestelmissä. Tapahtumakäsittelyn hallinta on merkittävässä roolissa, kun halutaan varmistua tiedon eheydestä ja oikeellisuudesta. Heterogeenisessä ympäristössä toimittaessa myös eri järjestelmien tiedon esitystavat poikkeavat, joten tiedon transformaation toteuttamiseen eri järjestelmien välillä on kehitettävä yhtenevät ratkaisut.

Tietoturvan (*engl. security*) toteuttaminen on yksi tärkeimmistä osista avoimia järjestelmiä toteutettaessa. Monitasoarkkitehtuuriin perustuvien avoimien järjestelmien toteutusympäristönä on Internet, jolloin tietoturvaan liittyvät haasteet ovat suurempia. Järjestelmistä saatava tieto on yleensä kriittistä, joten yleensä loppukäyttäjälle halutaan tarjota vain välttämätön informaatio. Arkkitehtuuria kehitettäessä onkin tiedostettava avoimuuden myötä muodostuvat tietoturvariskit ja- uhat, jotta pystytään mitoittamaan tietoturvan toteutus oikein. Tiedonsiirron salaaminen, keskitetty oikeuksien hallinta sekä muut tietoturvalta vaadittavat ominaisuudet tuovatkin omat vaatimuksensa käytettäville tiedonsiirtoprotokollille sekä väliohjelmistoille rakennettaessa avointa järjestelmää.

2.2 Hajautetun järjestelmän erilaiset kommunikointimetodit

Monitasoarkkitehtuuriin pohjautuvissa avoimissa järjestelmissä välitetään paljon tietoa eri järjestelmien välillä, jolloin vaatimukset kommunikoinnin toteutukselle eroavat eri tilanteissa. Kommunikoinnin suunnittelussa ja toteutuksessa käytettävät kommunikointimetodit esitetään kuvassa 2. Kuvassa esitettävät metodit ovat yksinkertaistettu, joten niitä voi soveltaa ja jatkokehittää tarpeita vastaaviksi.



Kuva 2: *Väliohjelmistojen käyttämät kommunikointimetodit [GG0799]*

Keskustelu (*engl. conversational*) on kommunikointimetodi, jossa järjestelmät ovat keskenään kiinteässä vuorovaikutuksessa. Järjestelmien välinen sidos on tiukka, sillä keskusteluun osallistuvien järjestelmien on oltava samanaikaisesti toiminnassa. Keskustelu on synkroonista ja yleensä lukitsevaa, joten sen käyttö on varsin vähäistä erilaisissa väliohjelmistoissa [GG0799].

Pyyntö ja vastaus (*engl. request and reply*) on kaksisuuntainen ja vuorovaikutukseen perustuva kommunikointimalli. Se eroaa keskustelusta siinä, että pyyntöjä ja vastauksia on vain yksi kappale kumpaakin. Metodi on lukitseva ja synkroninen, joten järjestelmien välille syntyy tiukka sidos. Väliohjelmistoilla metodia toteutettaessa jaetaankin molemmat operaatiot yleensä erillisiksi yksisuuntaisiksi operaatioiksi [GG0799].

Sanoman välitys (*engl. message passing*) suora järjestelmien välinen kommunikointimetodi. Järjestelmien välinen kommunikointi on yhteydellistä, mutta se voi olla joko synkronista tai asynkronista [ISG0297]. Koska kommunikointi on yhteydellistä, järjestelmien välille syntyy tiukka sidos.

Sanomajono (*engl. message queuing*) epäsuora järjestelmien välinen kommunikointimetodi, jossa kommunikointi tapahtuu sanomajonon kautta yhteydettömästi. Tekniikka tarjoaa joustavuutta järjestelmien väliseen kommunikointiin, sillä järjestelmien ei tarvitse olla samaan aikaan toiminnassa ja niissä voi olla myös suorituskyvyssä eroja [ISG0297]. Tekniikka vaatii väliohjelmistolta sanomien levylle kirjoittamista, joka on hitaampaa, mutta myös samalla luotettavampaa [GG0799].

Julkaiseminen (*engl. publish and subscribe / producer and consumer*) kommunikointimetodi, jossa jokin järjestelmä tuottaa tietoa verkkoon ja toiset järjestelmät noutavat sen käyttöön tarpeidensa mukaan. Järjestelmät ovat löyhällä sidoksella liitetty toisiinsa. Metodi on varsin joustava ja se tarjoaa mahdollisuuden tehokkaaseen tiedon levittämiseen, joten se on noussut varsin merkittäväksi kommunikaation toteutustavaksi varsinkin erilaisissa sähköisen kaupankäynnin järjestelmissä [ISG0297].

2.3 Tapahtumakäsittelyn toteuttaminen hajautetussa järjestelmässä

Tapahtumakäsittelyn suorituskyvylle asetetaan tiukat vaatimukset, sillä se on yksi tärkeimmistä osista hajautettujen järjestelmien toiminnassa. Tapahtumien hallintaan on myös kiinnitettävä huomiota, sillä luotettavuus ja tehokkuus ovat tärkeä osa toimivaa järjestelmää. Skaalautuvuus tapahtumamäärien muuttuessa on tärkeätä, koska järjestelmille asettavat vaatimukset kuormituksen suhteen voivat muuttua nopeasti avoimia järjestelmiä toteutettaessa.

Monitasoarkkitehtuuria rakennettaessa ja toimittaessa hajautetussa ympäristössä tulee yleensä tarve päivittää useita eri tietokantoja yhtenä kokonaisuutena, jolloin tapahtumaeheyden saavuttaminen vaikeutuu. Jos tapahtumakäsittely hajautetussa ympäristössä halutaan toteuttaa ajantasaisesti ehyenä, tehdään se kaksivaiheisen sitoutumisen 2PC:n (*Two Phase Commit*) avulla [EJ99], muutoin riittävä eheys voidaan varmistaa tekemällä lokitiedostojen vertailua tai käsittelemällä tietoja kahteen kertaan. 2PC:n käyttö vaatii sovelluskehityksessä tarkkuutta, koska sen aiheuttamat virhetilanteet saattavat olla vaikeita selvittää ja toisaalta sen käyttö lisää verkkoliikennettä. Tekniikan käytölle ei ole sinänsä esteitä, sillä tapahtumamonitorit ja relaatiotietokannat tarjoavat yleisesti mahdollisuuden toteuttaa 2PC.

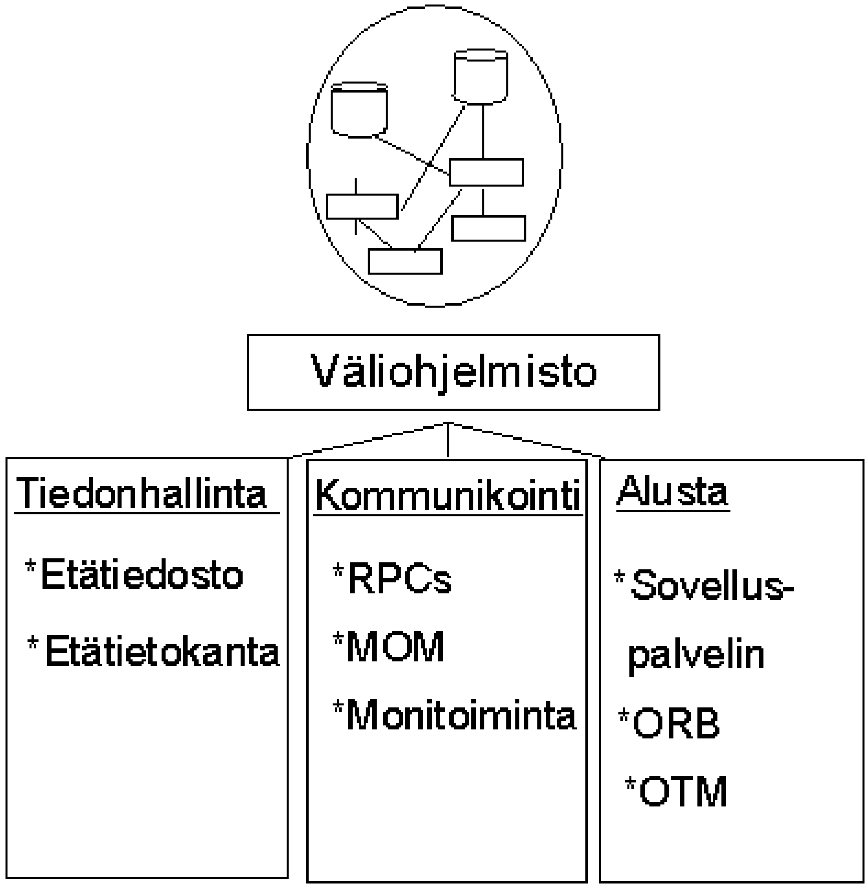
Tapahtumakäsittelyn toteutuksessa tulee huolehtia tapahtuman eheysominaisuuksista, jotka on hajautetussa järjestelmässä pystyttävä takaamaan. Eheys- eli ACID-ominaisuudeet sisältävät jakamattomuuden (*engl. atomicity*), vakioisuuden (*engl. consistency*), itsenäisyyden (*engl. isolation*) ja pysyvyyden (*engl. durability*) [EJ99]. Jakamattomuudella tarkoitetaan jakamatonta tehtäväkokonaisuutta, jossa kaikki osatehtävät suoritetaan tai jätetään suorittamatta. Vakioisuudella taataan, että järjestelmä jää tapahtuman suorituksen jälkeen vakioon lopputilaan tai muuten tapahtuma on peruttava. Tapahtuman itsenäisyyteen ei saa vaikuttaa muut samaan aikaan suorituksessa olevat tehtävät ja jaettujen resurssien käyttö tulee sarjallistaa sekä samalla taata, että samanaikaiset tapahtumat eivät vaikuta toistensa toimintaan. Pysyvyydellä tarkoitetaan tilaa, joka on järjestelmässä sitoutumisen jälkeen ja sen tulee säilyä järjestelmän vikaantuessakin.

3 Erilaiset väliohjelmistot ja niiden perusominaisuudet

Väliohjelmisto määritellään ajonaikaiseksi järjestelmäksi, jolla mahdollistetaan sovellustason yhteentoiminta hajautetussa ympäristössä [GG0799]. Väliohjelmisto on kerros sovelluskerroksen ja alemman tason kerrosten, kuten käyttöjärjestelmän, tiedonhallinnan ja verkko palveluiden, välissä. Välikerroksen käytöllä mahdollistetaan heterogeenisten alustojen ja protokollien yhteenliittäminen [ISG0297]. Väliohjelmistot tarjoavat ylemmän tason rajapinnan, jolla peitetään protokollapinot ja tiedonsiirron monimutkaiset ominaispiirteet. Väliohjelmistolla huolehditaan tiedon esitystavasta siirron aikana (*engl. marshaling and unmarshaling*). Tiedonsiirtoprotokollat suorittavat pakettien reitittämisen, virheentarkistukset sekä muut tiedonsiirrolle tyypilliset ominaisuudet. Väliohjelmistojen käytöstä huolimatta on järjestelmien väliseen tiedonsiirtoon myös kiinnitettävä huomioitava, sillä tiedonsiirto on hajautetun järjestelmän selkäranka.

Väliohjelmistoilla suoritetaan kuormituksen tasausta, lisätään skaalautuvuutta ja saadaan samalla suorituiskykyisempiä järjestelmiä käyttöasteen kasvaessa. Korkean käytettävyyden (*engl. high availability*) järjestelmissä luotettavuuden saavuttaminen onkin ensiarvoisen tärkeätä.

Väliohjelmistoista on monia erilaisia toteutuksia, sillä eri tekniikat soveltuvat paremmin eri tarkoituksiin. Perusväliohjelmistot jakautuvat ominaisuuksiensa perusteella kolmeen luokkaan, jotka ovat kuvassa 3 esiteltävät kommunikointi-, sovellusalusta- ja tiedonhallintaväliohjelmistot.



Kuva 3: *Perusväliohjelmistojen jaottelu [GG0799]*

3.1 Kommunikointiväliohjelmistot

Kommunikointiväliohjelmistoja (*engl. communication middleware*) käytetään toteutettaessa sovellusten välistä kommunikointia. Väliohjelmistolla toteutetaan ylemmän tason rajapinta, jolla peitetään alemman tason tiedonsiirtopalvelut. Tekniikalla helpotetaan ja nopeutetaan hajautettujen järjestelmien kehittämistä, koska ohjelmointi suoraan tietoliikennerajapintoihin on huomattavasti monimutkaisempaa ja hitaampaa kuin käytettäessä ylemmän tason rajapintaa. Erityisiä kommunikointiväliohjelmistoja ovat etäproseduuri kutsuun RPC (*engl. Remote Procedure Calls*) ja sanomapohjaiseen väliohjelmistoon MOM (*engl. Message Oriented Middleware*) pohjautuvat toteutukset.

3.1.1 Etäproseduurikutsuun perustuva väliohjelmisto

Etäproseduurikutsuun RPC perustuva väliohjelmisto on ollut ensimmäinen ohjelmien väliseen kommunikointiin soveltuva rajapinta, mutta sen rooli ei ole kuitenkaan koskaan noussut merkittäväksi [GG0799]. RPC:n päälle on erikseen rakennettava tapahtumakäsittely ja lisäksi erilaiset kommunikointimetodit ovat vaikeita toteuttaa sen avulla.

RPC:llä voidaan toteuttaa synkroniseen kyselyyn pohjautuvia ratkaisuja, joissa asiakas ja palvelin ovat tiukasti sidottu toisiinsa. Asynkroninen kysely voidaan toteuttaa RPC:llä callback-toiminnon avulla siten, että asiakas keskeyttää suorituksensa tiettyyn kohtaan ja jää odottamaan palvelimelta tulevaa vastausta. RPC-toteutuksissa voi olla kutsuvassa päässä myös jonotusmekanismin toteutus.

RPC ei ole tekniikkana yksistään riittävä toteutettaessa monitasoarkkitehtuuria, mutta sen ominaisuuksia hyödynnettään pääasiassa osana muita väliohjelmistoja. RPC-tekniikka on lähtenyt yleistymään viime aikoina hajautettujen komponenttimallien myötä, sillä niissä sitä käytetään yleisesti kommunikoinnin toteutuksessa.

3.1.2 Sanomapohjainen väliohjelmisto

Sanomapohjainen väliohjelmisto MOM on nykyisin yksi merkittävimmistä väliohjelmistoista, sillä se tarjoaa monia hyödyllisiä erityisominaisuuksia. Sen avulla voidaan rakentaa yhteydetön kommunikointi, jolloin sanomat laitetaan jonoon ja välitetään perille joko aikaohjattuna tai yhteyksien palauduttua. Lähes reaaliaikaisen tiedonsiirron toteuttaminen sekä erilaiset ryhmälähetykset (*engl. multicast*) usealle eri järjestelmälle on mahdollista toteuttaa MOM:n avulla. Tekniikalla voidaan yhdistää helposti erilaisia järjestelmiä, koska sanoman formaattia voidaan muuntaa eri järjestelmien tukemiin muotoihin ja MOM toteutuksia löytyy useille eri alustoille [MOM97].

MOM-tuotteet tarjoavat tyypillisesti ylemmän tason rajapinnan, jonka kautta sanomien välitys ohjelmoidaan. Sanoman välitykselle tarjotaan yleensä erilaisia QoS-parametreja. Luotettava sanomavirta (*engl. reliable message delivery*) takaa pakettien perille menon verkossa. Taattu sanomavirta (*engl. guaranteed message delivery*) huolehtii sanoman perille välittömästi, jos yhteys on pystyssä, mutta muutoin sanaomanvälitys toteutetaan heti kun se on mahdollista, tosin määritellyn viipeen rajoissa. Vakuutettu sanomavirta (*engl. assured messsage delivery*) huolehtii, että sanomat lähetetään vain kerran [ISG0297]. Luotettavuuden toteuttaminen MOM:n kaatuessa vaatii sanomien levylle kirjoittamista, joka on hitaampaa, mutta myös samalla takaa toipumisen virhetilanteesta.

MOM-tuotteissa on reitityksen toteutus eri järjestelmien välillä ja sanoman välityksen onnistumisesta on mahdollista saada eritasoisia kuittauksia. Sanomien priorisoinnilla ja kuormituksen tasauksella voidaan hallita ruuhkatilanteita sekä mahdollistaa kriittisien sanomien perille meno. Lisäksi joissakin MOM-tuotteissa on järjestelmien hallintapiirteitä, jolloin voidaan käynnistää ohjelmat tarvittaessa, sekä yhteyksiä eri järjestelmien välillä voidaan keskittää, jolloin voidaan vähentää tietoliikenteen kustannuksia ja vähentää kaistan käyttöä verkossa [ISG0297].

Sanomavälityksellä voidaan toteuttaa asynkronisia kyselyitä, joka on yksinkertaisuudessaan sanomien välittämistä joko prosessien tai eri järjestelmien välillä. Tekniikkaa mahdollistaa yhteydettömän kommunikoinnin, jolloin järjestelmien välinen liitos on löyhä ja pääpaino on välitettävällä tiedolla. Sanomien säilymisestä ja perillemenosta on varmistuttava, joten toteutuksen luotettavuuden voi määritellä edellä kuvattujen QoS-parametrien käytöllä.

Sanomajonossa on sanomavälitykseen lisätty sanomien jonotusmalli, jossa voidaan nimetä jonot ja kohdistaa tietyt prosessit palvelemaan tiettyjä jonoja. Tekniikka soveltuukin päivityksen sekä asynkronisen päivityksen kuittauksella toteuttamiseen. Sitä käytetään järjestelmäintegraation toteuttamiseen tilanteissa, joissa järjestelmien välinen yhteys ei tarvitse olla reaaliaikainen. Sanomajono sopii myös organisaatioiden väliseen integrointiin hyvien tietoturvaominaisuuksiensa puolesta. Järjestelmäintegraatiossa on sanomajonot toteutettava vikasietoisina, jotta jonossa olevat sanomat säilyvät virhetilanteissa ja lisäksi silloin voidaan varmistua sanomien perille meneminen, vaikka kohdejärjestelmä ei olisi kokoajan käytössä.

Monipuolisimmat väliohjelmistopalvelut tarjoavat sanomabrokerit (*engl. mes-*

sage broker), jotka sisältävät sanomaliikenteen muodon ja kulun hallintaan erilaisia palveluita. Niillä voidaan muuttaa sanomien muotoa eli toteuttaa transformaatio eri järjestelmien välillä, ajoittaa sekä reitittää sanomia eri kohteisiin. Sanomabrokerit tarjoavat ratkaisun julkaisemisen toteuttamiselle, jolloin brokeri toimii eri järjestelmien välisen tiedon julkaisijana. Ratkaisu tarjoaakin vaihtoehdon, joka on toteutettu aikaisemmin yhdistämällä eri väliohjelmistojen palveluja sanomajonoihin ja sanomavälitykseen.

3.2 Sovellusalustaväliohjelmistot

Sovellusalustaväliohjelmisto (*engl. platform middleware*) pitää sisällään kommunikointiväliohjelmiston, sekä sen lisäksi toimintoja resurssien hallintaan, ohjelmien käynnistämiseen ja pysäyttämiseen, kuormituksen tasaukseen ja sovellusten hallintaan. Tapahtumakäsittelyn toteuttaminen vaihtelee erilaisia väliohjelmistoja käytettäessä, mutta yleensä se toteutetaan käyttämällä tapahtumakäsittelymonitoreja (*engl. transaction monitor*) tai sitten tapahtumakäsittelyn hallinta on sisällytetty väliohjelmiston toteutukseen.

Sovellusalustaväliohjelmistojen toteutuksissa käytetään pääasiassa kolmea eri komponenttimallia, jotka ovat OMG:n (*engl. Object Management Group*) CORBA (*engl. Common Object Request Broker Architecture*), Microsoftin DCOM (*engl. Distributed Component Object Model*) [META0398], sekä Sunin Java RMI (*engl. Remote Method Invocation*) [SUN0599]. Eri väliohjelmistototeutukset tukevat yhtä tai useampaa mallia, joiden lisäksi erilaisia komponenttipohjaisia väliohjelmistoja voidaan integroida yhdyskäytävien (*engl. gateway*) avulla.

OTM (*engl. Object Transaction Monitors*) tarjoaa alustan komponenttien ja niiden tilojen hallintaan. OTM sisältää ORB:n tarjoamat palvelut sekä niiden lisäksi komponenttien ja tapahtumien hallintaan erilliset palvelut [GG0897]. OTM:stä on määritelty kaksi erillistä standardia, joista EJB (*engl. Enterprise Java Beans*) on JavaSoftin ja CCM (*engl. CORBA Component Model*) OMG:n määrittelemä.

3.2.1 Tapahtumakäsittelymonitorit

Ensimmäiset tapahtumakäsittelymonitorit TP-monitorit (*engl. Transaction Processing Monitors*) on otettu käyttöön jo rakennettaessa keskuskoneeseen (*engl. Mainframe*) pohjautuvia toteutuksia. TP-monitorilla hallitaan tapahtumia ja suoritetaan kuormituksen tasausta, jotta alla oleva järjestelmä selviytyy suuristakin asiakasmääristä [EJ99].

TP-monitori tarjoaa liitynnän heterogeenisiin alustoihin ja tietokantoihin, sillä liitynnät toteutetaan standardirajapintojen kautta. Raskaan tapahtumakäsittelyn (*engl. TP-Heavy*) tarve syntyy, kun halutaan tehdä liitynnät useampaan eri tietokantaan ja toteuttaa monimutkaisempia kommunikointiratkaisuja. Kevyen tapahtumakäsittelyn (*engl. TP-Lite*) toteutuksessa kutsutaan yleensä toimittajakohtaisen rajapinnan kautta tietokantaproseduureja ja tapahtumakäsittelyn hallinta on osana tietokantaa. Raskaan tapahtumankäsittelyn toteutuksessa määritellään rajapinnat IDL-kielellä (*engl. Interface Definition Language*), käytetään erilaisia kommunikointimalleja ja lisäksi toteutukseen yhdistetään erilaisia tietoturva ja hakemistopalveluja, jotta saadaan avoin toteutus [EJ99].

TP-monitorit tukevat yleensä joko yhtä tai useampaa kommunikointimallia [EJ99], mutta tapahtumankäsittelyn hallinta tuo niihin lisävaatimuksia. Kommunikointimalleihin on liitettävä lisäominaisuuksina sitoutumisen synkronointi, vain kerran semantiikka (*engl. only once semantics*) sekä palvelimien hallinta- ja yhteyksien toipumisominaisuuksia. Tapahtumien hallinta ominaisuuksien myötä kaikki kutsuttavat resurssit tulevat osaksi tapahtumaa, jolloin myös sekä järjestelmien väliset että asiakkaan ja palvelimen väliset sidokset tiukkenevat [EJ99].

3.2.2 DCOM

DCOM on Microsoftin hajautettu komponenttimalli, joka pohjautuu Microsoftin käyttöjärjestelmissä käytettävään COM:iin (*engl. Common Object Model*). DCOM:n kehittyessä ja uusien käyttöjärjestelmäversioiden tullessa Microsoft muuttaa mallin nimeksi COM+, joka sisältää myös uusia palveluita, kuten ajonaikaisen objektien hallinnan [QUOIN98]. COM-infrastuktuuri mahdollistaa erilaisten ActiveX- ja OLE-palveluiden käytön sovelluksissa, kun suoritusalusta on Microsoftin tukema. Hajautusta tukeva tapahtumakäsittely toteutetaan MTS:n (*engl. Microsoft Transaction Server*) avulla ja sanomanvälitykseen pohjautuvissa ratkaisuissa käytetään MSMQ:ta (*engl. Microsoft Message Queue*). DCOM:ssa kommunikointi perustuu RPC:hen, joka käyttää yhteydetöntä UDP/DCOM-kommunikointiprotokollaa [META0398]. DCOM:n tarjoamat mahdollisuudet järjestelmäintegraatiolle ja avoimien järjestelmien toteuttamiselle ovat rajalliset, koska toteutus on alustariippuvainen ja sen käyttämä kommunikointiprotokolla ei ole yleistynyt muissa väliohjelmisto-toteutuksissa, joten yhteistoiminta muihin komponenttimalleihin joudutaan toteuttamaan erillisten yhdyskäytävien avulla.

3.2.3 CORBA

Erilaiset komponenttipohjaiset väliohjelmistot pohjautuvat yleensä CORBA-standardiin, joka takaa järjestelmille alusta- ja kieliriipumattomuuden. Komponenttien välinen kommunikointi toteutetaan ORB:n (*engl. Object Request Brokers*) avulla. Sen kautta voidaan kutsua objekteja joko paikallisesti tai hajautetusti sekä ottaa objekteilta tulevia vastauksia vastaan [OMG0699]. CORBA:n käyttämät kommunikointiprotokollat ovat GIOP (*engl. General Inter-ORB Protocol*) sekä ja TCP/IP-verkoissa IIOP (*engl. Internet Inter-ORB Protocol*).

CORBA:ssa rajapintamäärittelyt tehdään IDL-kielellä, josta määrittelyt käännetään standardin tukemille kielille, joilla objektien toiminnallisuus toteutetaan [META0398]. Käännöksessä syntyy asiakkaan puolelle IDL-tynkä (*engl. IDL stub*) ja objektin toteukseen palvelimen puolelle IDL-runko (*IDL skeleton*), joiden kautta kutsut toteutetaan. Ratkaisulla voidaan piilottaa objektien ja asiakkaiden toteutus- ja kutsutapa sekä sijainti. Lisäksi objektien hallinta on kokonaisuudessaan ORB:n vastuulla.

CORBA tarjoaa kaksi erilaista pääsyä objekteihin, SII (*engl. Static Invocation Interface*) ja DII (*engl. Dynamic Invocation Interface*). SII on varsin samantyyppinen kuin RPC-rajapinta. Rajapinnan kautta voidaan toteuttaa synkroninen kysely, jossa IDL:llä toteutetulla sidonnalla huolehditaan parametrien välitys verkossa. Toteutus ei ole kieliriippuvainen, mutta muutoksia tehtäessä objektit on käännettävä uudelleen. DII on yleinen rajapinta, joka mahdollistaa ajonaikaisten kyselyiden rakentamisen. Uudet palvelinobjektit määritellään kuvauskannassa, josta voidaan kysyä tarvittavat tiedot. Saadun tiedon perusteella muodostetaan dynaaminen kutsu objektille [ISG0297].

CORBA:n standardissa määritellään monia palveluita, mutta kaikkia palveluita ei ORB-toteutuksista vielä löydy [OF0399]. Toteutetuista palveluista hyödyllisimpiä ovat avoimien järjestelmien kehittämisessä nimipalvelu ja meklaripalvelu [OF0599], mutta myös monet objektien hallintaan liittyvät palvelut on toteutettu ja niitä käytetäänkin sovelluskehityksen yhteydessä.

CORBA ORB:t ovat ominaisuuksiltaan paljon monipuolisempia kuin etäproseduurikutsu tai sanomapohjainen väliohjelmisto, mutta silti niidenkin on hyödynnettävä muita väliohjelmistoja joissain toiminnoissa. Esimerkiksi tapahtumien hallinta toteutetaan OTS-rajapinnan (*engl. Object Transaction Service*) ja CORBA:n eri rinnakkaisuutta hallinnoivien palveluiden avulla. ORB-toteutukset käyttävät yleensä perinteisiä tapahtumankäsittelymonitoreita, jolloin kommunikointi ja hajautus toteutetaan ORB:n avulla. Lisäksi joissakin ORB-tuotteissa on toteutettu integraatio sanomaväliohjelmiston tarjoamiin palveluihin [META0398].

3.2.4 Java RMI

Java RMI tarjoaa yksinkertaisen ja suoran ratkaisun toteuttaa hajautettuja komponentteja Javalla. Tekniikka on Java pohjainen, joten siinä voidaan hyödyntää Javan tarjoamia siirrettävyys- ja tietoturva ominaisuuksia hajautetussa ympäristössä [SUN0599]. Käytössäolevien komponenttien ympärille voidaan rakentaa kääröt (*engl. wrapper*) ja toteuttaa rajapinnat, jolloin voidaan hyödyntää olemassaolevia komponentteja sekä vähentää samalla kieliriippuvuutta. JNI (*engl. Java Native Interface*) tarjoaa rajapinnan, jonka avulla voidaan toteuttaa metodien ja objektien ristiinkutsu Javan ja C/C++:n välillä. RMI:n kommunikointi perustuu RPC:hen ja kommunikointiprotokollana käytetään joko RMI:n natiivia protokollaa JRMP:tä (*engl. Java Remote Method Protocol*) tai CORBA:n määrittelemää IIOP:tä, jolloin voidaan helposti toteuttaa integrointi EJB- ja CORBA-järjestelmien välillä.

3.2.5 EJB

EJB-palvelin tarjoaa ajoympäristön EJB-teknologialla kehitetyille komponenteille. Se hallinnoi järjestelmän resursseja sekä tarjoaa laatikkoon (*engl. container*) Javalla toteutettavien EJB-komponenttien alustaksi. Laatikko tarjoaa komponentille hallinnollisia palveluita, joita ovat palveluhakemistot, tapahtumien hallinta, objektien tilojen ylläpidon, säikeistyksen ja tietoturvaominaisuuksia [PSG1298].

EJB-komponentit voivat olla joko istuntopohjaisia (*engl. session beans*) tai kohdepohjaisia (*engl. entity beans*). Istuntopohjaisen komponentin ilmentymä tai olio on yksilöllinen kullekin asiakkaalle ja se pystyy ylläpitämään tietoa asiakkaan tilasta jäsenmuuttujissaan, jolloin kyseessä on tilallinen olio (*engl. stateful session bean*). Tilattomia olioita (*stateless session bean*) käytettäessä voi useampi asiakas käyttää saman olion metodeita, koska silloin asiakkaasta ei tarvita yksilöllistä tilatietoa. Tällöin voidaan hyödyntää käytettävien olioiden varastoa ja samalla hallita väliohjelmiston muistin käyttöä.

Kohdepohjaisia olioita käytetään kuvaamaan tietoalkioita, jotka ovat esimerkiksi järjestelmän taustalla toimivassa tietokannassa. Tietoalkioita käsitellään näiden olioiden metodeilla, jotka vastaavat haluttua toimintalogiikkaa. Jokaisen kohdepohjaisen olion ilmentymä voi vastata esimerkiksi yhtä riviä tietokantataulusta tai useasta taulusta luotua näkymää. EJB-teknologialla rakennettava tietokantaan pohjautuva sovellus toimii yleensä niin, että asiakas kutsuu istuntopohjaisen olion kautta palvelinta, joka tekee tietokantakutsun kohdepohjaisen olion kautta. Istuntopohjainen olio vastaa tiettyä asiakasta ja kohdepohjainen olio tiettyä tietokantariviä, jolloin voidaan yhtäaikaisesti ylläpitää sekä asiakkaan että tietokantarivin tilatietoja. Istuntopohjaiset oliot hallitsevat yleensä hajautettuja tapahtumia JTS-rajapinnan (*engl. Java Transaction Service*) kautta, joka pohjautuu CORBA:n OTS:ään [PSG1298].

EJB-komponenttien kehittäminen on toimintalogiikan kirjoittamista, koska EJB-laatikot suorittavat tapahtumien hallinnan ja EJB-palvelimien vastuulla on komponenttien -ja järjestelmien välinen kommunikointi. Standardin mukaan kommunikointi voi Javan ja CORBA:n tasolla tapahtua RMI:llä käyttäen CORBA:ssa määriteltyä IIOP-protokollaa, mutta silloin ei voida hyödyntää kaikkia RMI:n tarjoamia ominaisuuksia [PSG1298]. IIOP:tä käytettäessä EJB-palvelin pystyy palvelemaan muitakin kuin pelkkiä Javalla toteutettuja asiakkaita ja lisäksi osa toteutetuista palveluista voi olla CORBA-pohjaisia. Esimerkiksi EJB-komponenttien nimihakuun käytettävä palveluhakemisto JNDI (*engl. Java Naming Directory Interface*) toimii yhdessä CORBA:n nimipalvelun kanssa.

3.2.6 CCM

CORBA:n komponenttimallin avulla voidaan kehittää ja levittää CORBA-sovelluksia. Malli tarjoaa arkkitehtuurin, jossa voidaan määritellä palvelinkomponentit ja niiden yhteistoiminta. Mallissa on laatikkopohjainen sovelluskehys, jossa voi toteuttaa komponentteja kieliriippumattomasti ja liittää niihin CORBA:n tarjoamia palveluita. Laatikkopohjainen sovelluskehys tarjoaa itsenäisen ympäristön, jossa komponenttien tapahtumien hallinta ja tietoturva on toteutettu [OMG0699]. Sovelluskehitykseen tarkoitettu rajapinta on siis korkeammantason kuin käytettäessä teknisempiä CORBA-palveluita. CCM:stä löytyvät vastaavat palvelut kuin edellä kuvatusta EJB:stä, jonka lisäksi molempien komponenttimallien välinen yhteistoiminta on mahdollistettu.

3.3 Tiedonhallintaväliohjelmistot

Tiedonhallintaväliohjelmistot (*engl. data management middleware*) tarjoavat pääsyn heterogeenisiin tietolähteisiin, jolloin voidaan kirjoittaa ja lukea tietoa hajautetuista tietokannoista tai tiedostojärjestelmistä [GG0799]. Tekniikkaa tarvitaan, sillä yleensä tarvittavasta tiedosta ei ole kuin osa tietokannoissa ja loput tiedoista on haettava erilaisista tietovarastoista (*engl. data warehouse*) tai tiedostoista. Tiedonhallintaväliohjelmistot tukevat parhaiten asiakas-palvelin-mallia, mutta niitä voidaan myös käyttää monitasoarkkitehtuurissa. Etätiedostojärjestelmä RFC (*engl. Remote File System*) ja etätietokanta RDA (*engl. Remote Database Access*) ovat eniten käytetyt tiedonhallintaväliohjelmiston toteutukset. Tämän tyyppiset toteutukset eivät yleensä ole erillisiä tuotteita, vaan ne on sisällytetty muihin ohjelmistoihin, mutta järjestelmäintegraatiota varten on kehitetty myös erillisä tuotteita.

4 Avoimen järjestelmän sovelluskehys

Sovelluskehys tarjoaa sovelluskehittämiselle ympäristön, jossa on kuvattu rajapinnat ja toteutettu liitynnät alapuoliseen järjestelmäinfrastruktuuriin. Sovelluskehyksen käyttäminen lisää sovellusten komponenttien uudelleenkäytettävyyttä ja samalla pystytään parantamaan sovellusten laatua. Kehyksen kautta saadaan käyttöön tarvittavat komponentit ja palvelut järjestelmäkehitystä varten, jolloin sovelluskehittäjä voi keskittyä toiminnallisuuden kehittämiseen ja tekninen ohjelmointi vähenee.

Komponenttipohjaisen ratkaisun joustavuutta lisätään hyödyntämällä kuvauskantoja sovelluskehyksien määrittämiseen ja ylläpitämiseen. Avoimen järjestelmän sovelluskehyksessä palvelut kuvataan metatietona kuvauskantoihin, josta komponentteja voidaan kutsua. Komponentit voidaan toteuttaa aikaisemmin kuvattujen väliohjelmistojen ominaisuuksia hyödyntämällä. Sovelluskehyksen tulisikin tarjota liitynnät erilaisiin väliohjelmistoihin ja sovelluspalveluihin, koska niitä tarvitaan avoimen järjestelmän toteuttamisessa.

4.1 Sähköisen kaupankäynnin sovelluskehyksen rajapinnat eri palveluihin

Sunin määrittelemä J2EE [SUN1299] tarjoaa alustan, jossa on toteutettu rajapinnat erilaisiin avoimen järjestelmän toteutuksessa tarvittaviin palveluihin. J2EE:n ajonaikainen ympäristö määrittelee erilaisia sovelluskomponentteja ja niille tarvittavan ajoympäristön. Alla olevissa taulukoissa on eritelty J2EE:ssä määriteltyjä standardeja ja rajapintoja, joita käyttäen voi hyödyntää hajautetun infrastruktuurin tarjoamia palveluita. Taulukot pohjautuvat IBM:n toteuttamaan sähköisen kaupankäynnin sovelluskehykseen [IBM1199], joka tarjoaa Javaan pohjautuvan sovelluskehitysympäristön. Siinä on liitynnät erilaisiin sovellus- ja väliohjelmistopalveluihin sekä lisäksi on otettu käyttöön XML (*eXtensible Markup Language*) tiedon esitystavan määrittelemiseen, jotta esimerkiksi tiedon transformaation toteuttaminen helpottuu.

Taulukko 1: *Sovelluskehyksen toteuttavan infrastruktuurin standardit ja rajapinnat*

Palvelu	*Palvelun standardi*	*Rajapinta*
Hakemisto	LDAP	JNDI
Tietoturva	SSL, IPsec, SET	JSSL, JCE, JAAS
Tietoliikenne	TCP/IP, WAP	JDK java.net
Tiedosto	-	JDK java.io
Tulostus	-	JDK java.2d, JNPAPI

Taulukko 2: *Sovelluskehyksen sovelluspalvelimien standardit ja rajapinnat*

Palvelu	*Palvelun standardi*	*Rajapinta*
Sähköposti	SMTP, POP3, IMAP4	Java Mail
Tietokanta	ODBC, JDBC	EJB
Tapahtuma	CORBA OTS/IIOP	EJB, JTS, JTA
Sanomajono	-	JMS

Taulukko 3: *Sovelluskehyksen WWW-ympäristön standardit ja rajapinnat*

Palvelu	*Palvelun standardi*	*Rajapinta*
WWW-palvelin	HTTP, HTML, XML	Servletti
Selain	HTTP, HTML, XML	Appletti, DOM
Komponenttimalli	CORBA IIOP	JavaBeans
Business-komponentit	CORBA IIOP	EJB, RMI
Skriptit	-	JSP

5 Yhteenveto

Avoimien järjestelmien kehittämiseen ja standardoimiseen antaa hyvät suuntaviivat ODP:n määrittelemät standardit ja viitemallit. Määritykset ovat abstraktilla tasolla, joten niistä jatkokehitetään standardit, joiden pohjalta syntyvät avoimien järjestelmien kehittämisen mahdollistavat ohjelmistot.

Väliohjelmistot tarjoavat laajan joukon erilaisia palveluita ja ratkaisuja avoimien järjestelmien kehittäjille. Väliohjelmistot koostuvat yleensä useammista eri tekniikoista sekä standardeista, jolloin ominaisuuksien vertailu hankaloituu ja tarpeita vastaavan ratkaisun löytäminen vaikeutuu. Komponenttimallien standardit kehittyvät ja niihin määritellään uusia palveluita, joten väliohjelmistoista ei välttämättä löydy toteutettuna kuin osa standardeissa määritellyistä ominaisuuksista. CORBA, DCOM ja EJB pohjaisia väliohjelmistoja on markkinoilla, mutta esimerkiksi CCM-perustaiset ratkaisut ovat vasta tulossa.

Väliohjelmistojen tarjoamien valmiiden sovelluskehysten lisäksi voi perinnejärjestelmistä luoda sovelluskehyksiä, jolloin voidaan hyödyntää niiden tarjoamia palveluita nykytekniikoillakin. Perusinfrastruktuurin kehittäminen avoimen hajautetun järjestelmän vaatimuksia vastaavaksi ja käytettävien rajapintojen julkistaminen mahdollistaa avoimien järjestelmien toteuttamisen. Metatietona tehtävät kuvaukset kuvauskantoihin määrittelee rajapinnat ja niiden tarjoamat palvelut. Rajapintojen säilyessä alla olevien toteutuksien muutokset eivät vaikuta muihin järjestelmiin, jolloin saadaan entistä joustavampia järjestelmiä [ODP0699].

Viitteet

[EJ99] Jeri Edwars, "3-Tier Client/Server At Work", John Wiley and Sons, 1999.

[MOM99] Rhys Lewis, "Advanced Messaging Applications with MSMQ and MQSeries", Que, joulukuu 1999.

[GG0799] R. Altman, J. Hill, J. Klein, B.Lheurex, Y. Natis, M. Pezzini, R. Schulte, S. Varma, "Middleware: The Glue for Modern Applications", Gartner Group, heinäkuu 1999.

[GG0897] Y. Natis, "Object Transaction Monitors: The Foundation for a Component-Based Enterprise", Gartner Group, elokuu 1997.

[ISG0297] International System Group, "Middleware - The Essential Component for Enterprise Client/Server Applications", International System Group, helmikuu 1997.

[PSG1298] A. Thomas, "Enterprise JavaBeans Technology - Server Component Model for the Java Platform", Patricia Seybold Group, joulukuu 1998.

[META0398] META Group Consulting, "CORBA vs. DCOM: Solutions for the Enterprise", META Group Consulting, maaliskuu 1998.

[MOM97] M. Korhonen, "Message Oriented Middleware (MOM)", Helsinki University of Technology, 1997.

[SUN0599] "Java Remote Method Invocation - Distributed Computing for Java", Sun Microsystems, toukokuu 1999.

[SUN1299] "Java 2 Platform, Enterprise Edition Specification", Sun Microsystems, joulukuu 1999.

[OF0399] P. Moxon, B. Rasmussen, "OpenFusion CORBA Services", PrismTecH, maaliskuu 1999.

[OF0599] P. Moxon, "OpenFusion Trading Service", PrismTecH, toukokuu 1999.

[IBM1199] "IBM Application Framework for e-business: Architecture Overview", IBM, marraskuu 1999.

[ODP0699] P. Linington, “An ODP Approach to the Development of Large Middleware System”, Kentin ylipoisto, Englanti, kesäkuu 1999.

[OMG0699] J. Siegel, “An Overview of CORBA 3”, OMG, kesäkuu 1999.

[ODP0198] L. Kutvonen, “Architectures for Distributed Systems: Open Distributed Processing Reference Model”, Helsingin yliopisto, Suomi, tammikuu 1998.

[QUOIN98] O. Tallman, J. Bradford Kain, “COM versus CORBA: A Decision Framework”, QUOIN, joulukuu 1998.