Palvelukeskeisen arkkitehtuurin toteutus IBM-suurkoneen IMS-järjestelmälle

Ari Kivioja

5.6.2008

Joensuun yliopisto
Tietojenkäsittelytiede
Pro gradu -tutkielma

Tiivistelmä

Palvelukeskeinen arkkitehtuuri on ajattelumalli ja abstrakti arkkitehtuurin toteutustapa. Sitä sovelletaan nykyisin erityisesti suurten yritysten tietojärjestelmien suunnittelua ja toteutusta ohjaavana mallina. Palvelukeskeisessä arkkitehtuurissa olennaista on tietojärjestelmien toiminnallisuuden tarjoaminen riittävän rajatuissa kokonaisuuksissa siten, että niiden uudelleenkäyttö olisi mahdollista. Sille on keskeistä myös tietojärjestelmien integrointi ja perinnejärjestelmien uudelleenkäyttö.

Tässä tutkielmassa tarkastellaan erilaisia tapoja toteuttaa palvelukeskeisen arkkitehtuurin mukaisia palveluita IBM-suurkoneen IMS-järjestelmälle. Tutkielman alussa tuodaan esiin palvelukeskeisen arkkitehtuurin peruskäsitteitä sekä IBM-suurkoneen ja IMS-järjestelmän ominaisuuksia. Palveluiden toteuttaminen IMS-järjestelmälle toteutetaan integroitumalla joko sen tapahtumankäsittelyjärjestelmään tai tietokannanhallintajärjestelmään. Erilaisten integrointitapojen teknisen tarkastelun lisäksi selvitetään kuinka ne sopivat erilaisiin palvelukeskeisen arkkitehtuurin palveluiden toteutusmalleihin ja perinnejärjestelmien muutosstrategioihin.

ACM (ACM Computing Classification System, 1998 version): C.2.4, D.2.2, D.2.11, H.3.4

Avainsanat: palvelukeskeinen arkkitehtuuri, suurkone, keskustietokone, perinnejärjestelmä, web-palvelutekniikka, integrointi, web-palvelu, SOA, IMS, IBM

Sisältö

Taulukot

Kuvat

1 Johdanto

Palvelukeskeinen arkkitehtuuri on yksi tämän hetken keskeisimmistä IT-arkkitehtuuriteemoista erityisesti yritysarkkitehtuurien osa-alueella. Tietotekniikan trendit ovat seuranneet yleismaailmallista kehitystä aina. Maailman muuttuminen ympärillä saa aikaan muutosta tietotekniikan alueellakin. Osa muutoksesta johtuu myös tietotekniikan alan omasta kehityksestä. Teknologian kehittyminen ja uudet innovaatiot mahdollistavat uudenlaisten ohjelmistojen kehittämisen ja uusia käyttötapoja tietotekniikan hyödyntämiselle. Ohjelmistoliiketoiminta on kasvanut niin suureksi, että osa yrityksistä pyrkii kasvattamaan liiketoimintaansa myös uusien tuotteiden ja teknologioiden esiintuomisella. Muutoksia tapahtuu koko ajan ja suurin osa muutoksista on pieniä, joten pidemmällä aikavälillä tarkasteltuna voitaisiin hyvin puhua mielummin evoluutiosta kuin revoluutiosta [Jos07].

Markkinatalouden muutos paikallisesta ja alueellisesta globaaliksi ohjaa yritysten toimintaa. Yritysten jatkuvan menestyksen tärkein tekijä nykyään on joustavuus ja kyky mukautua muuttuvaan kilpailutilanteeseen [Jos07]. Yleisiä ovat myös yrityskaupat ja fuusiot sekä liiketoimintamallien ja jopa toimintasektoreiden muutokset. Näihin muutoksiin täytyy yritysten tietotekniikan sopeutua. Pahimmillaan muutokset ovat suuria ja tapahtuvat lyhyessä aikajaksossa. Toisinaan tietojärjestelmät korvataan mahdollisesti kokonaan uusilla järjestelmillä ilman merkittävää muutosta liiketoiminnassa ainakaan paikallisella tasolla. Muutokset kohdistuvat ensin suuriin globaaleilla tai alueellisilla markkinoilla toimiviin yrityksiin, mutta vähitellen niiden vaikutukset leviävät myös paikalliselle tasolle.

Muutospaineita tietotekniikan alueelle luo myös nykyisin yleinen kustannustehokkuuteen perustuva yritysten johtamistapa. Aikaisemmin tietoteknologia nähtiin yhtenä yrityksen tärkeistä kilpailutekijöistä ja jopa kilpailuedun tuottajana. Nykyisin tarkastellaan tämän lisäksi usein myös tietojärjestelmien aikaansaamia kustannuksia. Tietotekniikan arkipäiväistyessä tietojärjestelmistä on tullut yhä enemmän kustannustekijä.

Palvelukeskeisen arkkitehtuurin käyttöönoton motivaattoreista ehkä tärkein on kyky uudelleen käyttää olemassa olevia tietojärjestelmiä. Uusi käyttö voi olla erilainen tapa käyttää entistä tietojärjestelmää tai yritykselle kokonaan uuden jo

muualla käytössä olleen järjestelmän käyttöönotto. Palvelukeskeisen arkkitehtuurin käyttöönottoon ajaa myös tarve käyttää erilaisia ohjelmistoja ja järjestelmiä yhtä aikaa samoja liiketoimintamalleja palvelemaan. [Erl05]

Toisinaan yritysostojen ja fuusioiden takia syntyy myös tarpeita yhdistää useita eri tietojärjestelmiä keskenään. Usein näissä yhdistettävissä järjestelmissä on päällekkäisiä toimintoja, jolloin niitä ei voida automaattisesti käyttää rinnakkain. Tyypillisesti näitä järjestelmiä on pystyttävä myös yhdistämään eli integroimaan keskenään vaikka ne saattavat edustaa hyvin erilaisia ohjelmistoarkkitehtuureita.

Valmisohjelmistojen käyttö yrityksissä on yleistynyt. Niiden käyttöön turvaudutaan usein erityisesti suurten tietojärjestelmämuutosten yhteydessä kustannussyistä. Tietojärjestelmähankkeiden aikataulut ovat usein tiukat ja tuolloin valmisohjelmistojen käytöllä tai muualta hankittujen järjestelmien käytöllä haetaan aikataulusäästöjä. Työn osuus suurissa järjestelmähankkeissa voi olla merkittävä, joten työmäärän vähentämisen kautta saavutetaan myös säästöjä.

Valmisohjelmat eivät yleensä ratkaise yrityksen koko tietojenkäsittelyyn liittyvää tarvetta, vaan voi olla tarve integroida keskenään useita valmisjärjestelmiä tai omia järjestelmiä näiden kanssa. Tämän lisäksi valmisjärjestelmät harvoin täyttävät vain tiettyä aukkoa tietojärjestelmissä, vaan usein ne kattavat tarpeen joko liian laajasti (tulee päällekkäisyyttä eri järjestelmien kanssa) tai liian puutteellisesti (vaaditaan mukauttamista tai räätälöimistä).

Palvelukeskeinen arkkitehtuuri on tapa hahmottaa ja yhdistää tietojärjestelmiä ja niiden tarjoamia palveluita. Se on enemmänkin ajattelumalli kuin konkreettinen arkkitehtuuriratkaisu [Erl05]. Joissakin uudemmissa tietojärjestelmissä on huomioitu jo suunnitteluvaiheessa sen sopivuus palvelukeskeiseen arkkitehtuuriin. Tällöin suunnittelumallina on ollut palvelukeskeisen arkkitehtuurin periaatteet. Palvelukeskeinen arkkitehtuuri yhdistetään usein yhden yrityksen toimialueeseen (Domain) liittyväksi, joten tarkastelutaso on laajempi kuin vain yksi tietojärjestelmä [Jos07]. Yrityksen kaikkia järjestelmiä tarkastellaan silloin kokonaisuutena. Tyypillisesti tällaisessa kokonaisuudessa osa järjestelmistä (tai kaikki) on vanhoja, joten ne vaativat muutoksia soveltuakseen palvelukeskeiseen arkkitehtuuriin. Nämä muutokset ovat tapauskohtaisia, mutta tiettyjä suunnittelumalleja voidaan soveltaa lähes kaikissa tilanteissa.

IBM-suurkoneiden keskeisinä suunnitteluperiaatteina olivat 1960-luvulla muun muuassa skaalautuvuus ja järjestelmien pitkäaikainen yhteensopivuus. Nämä periaatteet ovat myös modernin suurkoneen suunnittelussa tärkeitä. Suurkonejärjestelmien käyttö on muuttunut noista ajoista nykypäivään. 1980- ja 1990-luvuilla järjestelmien käyttöä lisäsivät asiakas-palvelin -arkkitehtuurin mukaiset järjestelmätoteutukset. 2000-luvulta alkaen suurkonejärjestelmien käyttöä kasvattivat erityisesti Internet-pohjaiset palvelut ja niiden myötä lisääntyneet käyttäjämäärät. Palvelukeskeinen arkkitehtuuri on eräänlainen laajennus tähän kehityspolkuun. [SR06]

Tässä tutkielmassa tarkastellaan toteutusmalleja, joita voidaan käyttää toteutettaessa palvelukeskeisen arkkitehtuurin mukaisia palveluita IBM-suurkoneen IMS-järjestelmälle ja erityisesti sen tapahtumankäsittelyjärjestelmälle toteutetuille sovelluksille. Toteutusmallien lisäksi selvitetään mitä erilaisia muutosstrategioita yrityksellä on perinnejärjestelmän (esimerkiksi IMS) liittämiseksi osaksi palvelukeskeistä arkkitehtuuria.

Keskeisenä tarkastelun kohteena tutkielmassa on IMS-järjestelmän ohjelmien ja -tietokantojen integrointi. Erilaiset IMS-tapahtumankäsittelyjärjestelmän ja -tietokannanhallintajärjestelmän integrointitavat käydään läpi ja lopuksi tarkastellaan miten ne sopivat palveluiden toteutusmalleihin ja perinnejärjestelmien muutosstrategioihin. Lukijan johdattelemiseksi aiheeseen käydään ensin läpi palvelukeskeisen arkkitehtuurin perusperiaatteita sekä IBM-suurkoneen ja IMS-järjestelmän ominaisuuksia ja soveltuvuutta palvelukeskeiseen arkkitehtuuriin.

2 Palvelukeskeinen arkkitehtuuri

Palvelukeskeinen arkkitehtuuri (Service Oriented Architecture, SOA) tietotekniikassa on parina viime vuotena ollut paljon esillä. Termiin käyttöön liittyy paljon epätäsmällisyyksiä, virheellistä tietoa, lupauksia tietojärjestelmien revoluutiosta ja markkinahypeä. SOA:sta esitetyt kritiikit väittävät, ettei se tuo mitään varsinaisesti uutta tietotekniikan alueelle vaan lähinnä käyttää uutta terminologiaa vanhoista ajatuksista. SOA:n puolesta lausutuissa kommenteissa puolestaan yleensä todetaan, että varsinaiset hyödyt saadaan, kunhan nähdään SOA oikealla tasolla sekä ymmärretään, että kysymys on enemmänkin evoluution omaisesta tietojärjestelmien kehittämisestä kuin suuresta kertamuutoksesta. Palvelukeskeisessä arkkitehtuurissa on kyse erityisesti olemassa olevien tietojärjestelmien elinkaaren pidentämisestä ja erilaisten järjestelmien integroimisesta yhteen sekä niiden tarjoamien palveluiden uudelleenkäytöstä [Erl05].

Palvelukeskeinen arkkitehtuuri on enemmän abstrakti ajattelutapa ja arkkitehtuurimalli kuin konkreettinen arkkitehtuuritoteutus. Se on myös teknologiariippumaton eikä ota suoraan kantaa toteutusteknologiaan, vaikka web-palvelutekniikka (Web Services) onkin kehittynyt yhtenä keskeisenä toteutusteknologiana SOA:n rinnalla, ja joissakin tapauksessa ne myös (virheellisesti) rinnastetaan toisiinsa. [Erl05]

Yritysarkkitehtuureita (Enterprise Architecture) toteutettaessa palvelukeskeinen arkkitehtuuri on yksi mahdollinen arkkitehtuurimalli. Yritysarkkitehtuuri on yhden yrityksen tietotekniikka-arkkitehtuuri, joten se kuvaa tällöin jonkin toimialueen tietotekniikan mallin. Tähän malliin palvelukeskeinen arkkitehtuurimalli sopii hyvin. Järjestelmien tarjoamat palvelut voidaan määritellä yritystasolla useiden eri järjestelmien välillä. Joissakin tapauksissa on ongelmallista, mikäli useampi kuin yksi järjestelmä tarjoaa samanlaista tai samaa tietoa sekä mekanismeja tietojen käsittelyyn. Tällöin on päätettävä mikä järjestelmä toteuttaa kunkin määritellyn palvelun.

Palvelukeskeinen arkkitehtuuri sopii erityisesti suurten hajautettujen tietojärjestelmien toteutustavaksi ja sen merkityksen ymmärtää helpommin, mikäli ymmärtää tällaisiin järjestelmäkokonaisuuksiin liittyviä haasteita. Se soveltuu myös ympäristöihin, joissa on edelleen käytössä vanhoja tietojärjestelmiä. Näitä järjes-

telmiä kutsutaan usein *perinnejärjestelmiksi (Legacy Systems)*, mikä tarkoittaa yleensä ikääntyneitä, tyypillisesti jopa yli 20 vuotta käytössä olleita järjestelmiä. Josuttis [Jos07] mainitsee palvelukeskeiseen arkkitehtuuriin siirtymisen syiksi perinnejärjestelmien käytön lisäksi myös tilanteet, joissa tietojärjestelmillä on useita eri omistajia yrityksen sisällä. Myös järjestelmien heterogeenisuus on yksi SOA:n käyttöönottoon motivoiva syy. Heterogeeniset järjestelmät voivat olla eri liiketoiminnan alueille sijoittuvia tai ne voivat olla eri teknologioilla toteutettuja.

Palvelukeskeisen arkkitehtuurin olemusta voidaan hahmottaa myös tarkastelemalla sen mukanaan tuomia etuja. Erl [Erl05] mainitsee tällaisinä hyötyinä muun muuassa tehokkaan integraation, palveluiden luontevan yhteistoiminnan ja uudelleen käytön, virtaviivaiset arkkitehtuuriratkaisut, perinnejärjestelmien hyödyntäminen, standardoitu tietojen esitystapa (XML), keskitetyt tietoliikenneratkaisut, vaihtoehdot tietojärjestelmävalinnoissa sekä organisaation ketteryys. Näiden lisäksi Hau & al. [HEHB08] mainitsevat SOA:n hyötyinä liiketoimintalähtöisyyden sekä palveluiden modulaarisuuden ja itsenäisyyden.

SOA:sta on liikkeellä paljon virheellisiä käsityksiä. Näiden seikkojen tunnistaminen ja niiden sisällön ymmärtäminen auttaa mahdollisten virheiden välttämisessä käyttöönottoprojektin aikana. Lewis & al. [LMSW07] ovat tunnistaneet yksitoista eri virhekäsitystä. Seuraavassa luettelossa listataan heidän havaitsemansa virheluulot sekä asian oikea tulkinta heidän mukaansa:

- SOA tuottaa täydellisen arkkitehtuurin: SOA on vain arkkitehtuurimalli eikä sitä voida ostaa ohjelmistotoimittajilta.
- Perinnejärjestelmät ovat helposti integroitavissa: Osa perinnejärjestelmistä ei ole muutettavissa palvelukeskeiseen arkkitehtuuriin sopivaksi ilman merkittävää työpanosta tai niiden arkkitehtuuri ei muuten sovellu uuteen malliin.
- SOA:ssa on kyse vain standardeista eikä muuta tarvita: Vaihtoehtoisia standardeja on paljon ja ne myös kehittyvät ja muuttuvat koko ajan.
- SOA on teknologiaa: Palvelukeskeinen arkkitehtuuri on tällä hetkellä ohjelmistovalmistajien kasvava markkinasegmentti, mutta eniten on silti merkitystä palveluilla liiketoiminnan näkökulmasta sekä niiden hallintomallista.

- Standardien käyttäminen takaa palveluiden välisen kommunikoinnin: Standardit mahdollistavat vain palveluiden välisen syntaktisen yhteensopivuuden, mutta eivät semanttista yhteensopivuutta.

- Palveluiden pohjalta on helppo kehittää sovelluksia: Oikeiden palveluiden löytäminen kehitys- ja ajonaikaisesti voidaan vielä ratkaista suhteellisen helposti, mutta palveluiden sisällön oikeellisuuden, turvallisuuden, suorituskyvyn eikä palveluiden loogisen sisällön ratkaisu ole silti helppoa.

- Palveluita, joita kuka tahansa voi käyttää on helppoa: Yleiskäyttöisen palvelun toteutus on sinällään helppoa, mutta palvelurajapinnan tekeminen ei.

- Palveluiden yhdistäminen dynaamisesti ajon aikana on helppoa: Palveluiden löytäminen ja niiden käyttäminen monimutkaisessa ympäristössä on käytännössä hankalaa johtuen osittain puutteellisista palveluiden kuvausmekanismeista.

- Palvelut voivat olla vain liiketoimintapalveluita: Liiketoimintapalveluihin keskitytään paljon ja ne ovat olennainen osa SOA-ympäristöä, mutta silti järjestelmäpalveluita (muun muuassa tietoturva ja lokitus) tarvitaan.

- Palveluita käyttävien sovellusten testaus ei ole erilaista kuin muiden sovellusten: Palveluiden monikäyttöisyys tuo mutkikkuutta testaukseen.

- SOA voidaan toteuttaa nopeasti: Todellisuudessa onnistuminen vaatii huolellista suunnittelua, pitkän tähtäimen suunnitelman sekä sitoutumista.

Edellä mainittujen seikkojen lisäksi Erl [Erl05] tuo esiin syitä, jotka tyypillisesti johtavat epäonnistuneeseen palvelukeskeisen arkkitehtuurin toteutukseen. Tärkein syy on se, että yritys ei erota hajautetun järjestelmän arkkitehtuuria palvelukeskeisestä arkkitehtuurista. Tämän ongelman vakavuuttaa lisää se, että kun yritys ei tiedosta näitä eroja, voi virheellisen toteutusmallin noudattaminen jatkua pitkäänkin. Toinen keskeinen ongelma on standardoinnin puute. Palvelukeskeisen arkkitehtuurin toteutus siihen soveltuvilla ohjelmistoilla ei pelkästään riitä. Yrityksen täytyy kehittää myös oma arkkitehtuuri, jossa määritellään muun muuassa tiedon esitysmuodot sekä palvelukuvausten sisältö. Kunnollisen siirtymäsuunnitelman laatiminen on myös tärkeää ja sen noudattamisella vältetään hätäisten

ja virheellisten ratkaisujen syntyminen. Näiden lisäksi Erl [Erl05] mainitsee, että palvelukeskeisen arkkitehtuurin suorituskykyvaikutuksiin ja tietoturvaratkaisuihin tulee erityisesti panostaa. Hän pitää myös tärkeänä, että XML otetaan heti alussa laajalti käyttöön kuvauskielenä, ja että käytettyjen tuotteiden ja standardien kehitystä seurataan ja siinä pysytään mukana.

2.1 Palvelukeskeisen arkkitehtuurin historia

Termin palvelukeskeinen arkkitehtuuri mainitaan usein saaneen alkunsa Gartnerin analyytikoilta. Josuttis [Jos07] mainitsee Roy Schulten maininneen hänelle keskustelussa, että SOA:n periaatteet keksi alunperin Gartnerin analyytikko Alexander Pasik jo vuonna 1994. Hän keksi termin *Server Orientation* erottamaan ns. uudentyyppiset asiakas-palvelin (Client-Server) -toteutukset perinteisistä. Näissä uusissa toteutuksissa asiakas-osuus toteutti paljon liiketoimintalogiikkaa ja palvelinosuus lähinnä huolehti tietokantayhteyksistä. Tässä uudessa mallissa termi asiakas-palvelin olivat käytössä laitekeskeisenä terminä, joten hän käytti uutta termiä ohjaamaan kehittäjiä palvelukeskeiseen (palvelin keskeisessä roolissa) malliin. Varsinainen ensimmäinen julkaisu aiheesta oli vuodelta 1996 Gartnerin Roy Schulten ja Yefim Natisin toimesta [Jos07]. Web-palvelutekniikoita pidetään usein synonyyminä SOA:lle, mutta niiden määrittely on käynnistynyt vasta myöhemmin vuonna 2000 SOAP (Simple Object Access Protocol) -protokollan standardoinnin myötä [Erl05]. SOAP:n ja muiden web-palvelutekniikoiden standardoinnin ja laajan käyttöönoton myötä myös tietoisuus SOA:sta on kasvanut.

Kooijmans et al. [KdGRY06] esittelevät ohjelmointityylien ja integrointitapojen kehitysaskeleet kohti palvelukeskeistä arkkitehtuuria. Ohjelmointityylit ovat kehittyneet perinteisestä proseduraalisesta ohjelmoinnista, oliopohjaisen ja komponenttipohjaisen ohjelmoinnin kautta palveluiden ohjelmointiin. Integrointitapana on heidän mukaansa ensin ollut pisteestä-pisteeseen (Point-to-Point) pohjainen integrointi useilla eri yhteyskäytännöillä (Protocol). Seuraavassa vaiheessa integroitavat järjestelmät kommukoivat keskenään keskitetyn viestipohjaisen ratkaisun kautta. Tämän jälkeen integrointitapana käytettiin yleisesti yritystason integrointia (Enterprise Application Integration, EAI). Siinä sovellukset kommukoivat keskitetyn pisteen (Hub) kautta, joka mahdollistaa entistä suuremmat erilaisten yhteyksien määrät sekä muunnosoperaatioiden suorittamisen ja viestien

reitittämisen. Palvelukeskeinen integrointi on väyläpohjainen integrointimalli, joka lisää yritystason integrointiin joustavuutta reititykseen, standardeja rajapintoja sekä palveluiden suoritusjärjestyksen eli koreografian hallinnan. Vastaavasti palveluiden ohjelmoinnissa voidaan palveluihin kuvata niiden keskinäisiä suoritusjärjestyksiä ja tällöin puhutaan myös prosessien koreografiasta.

2.2 Palvelukeskeisen arkkitehtuurin osa-alueet

Palvelukeskeinen arkkitehtuuri on pohjimmiltaan abstrakti arkkitehtuurimalli. Se on käytännössä myös jotain konkreettista, kuten olemassa olevat järjestelmät ja niiden välinen integraatio sekä järjestelmiin joko liiketoiminnallisella tai teknisellä tasolla liittyvät ihmiset. Palvelukeskeisen arkkitehtuurin keskeisimpänä osana ovat palvelut ja niiden ominaisuudet. Palveluiden lisäksi Josuttis [Jos07] mainitsee SOA:n käytännön toteutuksissa keskeisinä elementteinä:

- Tekninen järjestelmäarkkitehtuuri (Infrastructure): Kuvataan järjestelmän palveluiden tekniset liityntärajapinnat sekä erilaisia tukipalveluita, kuten tietoturva, lokitus ja hallinta. Voidaan toteuttaa teknisesti esimerkiksi yritystason palveluväylän (alakohta 2.2.2) avulla.
- Arkkitehtuuri (Architecture): Määritellään omilla arkkitehtuurikuvauksilla toteutukselle raamit. SOA itsessään ei tuo tarkkoja toteutuskuvauksia, joten tietomallien, sääntöjen ja toimintatapojen määrittely on toteutettava.
- Prosessit (Processes): Prosseista keskeisimpiä ovat liiketoimintaprosessit. Liiketoimintaprosessien mallinnuksella (alakohta 2.2.3) kuvataan kuinka palveluista koostetaan laajempia toimintoketjuja. Palveluiden elinkaaren hallinnan (Service Lifecycle) Josuttis sijoittaa prosesseihin kuuluvaksi, mutta usein se nähdään osana hallintomallia, kuten mm. Derler ja Weinreich [DW07] esittävät.
- Hallintomalli (Governance): Hallintomalli on eräänlainen metaprosessi. Se kuvaa palveluarkkitehtuuriin liittyvän organisaation, toimintatavat ja prosessit.

Palvelukeskeisen arkkitehtuurin keskeisin komponentti on palvelu ja siihen liittyvät osapuolet. Erlin [Erl05] mukaan käytännön toteutuksissa SOA on kuitenkin

muuttunut ja siihen on vaikuttanut erityisesti ohjelmistoteollisuuden kehitystyö. Hän käyttää termiä *nykyinen SOA* (Contemporary SOA) kattamaan näitä alkuperäisen SOA:n laajennuksia, jotka ovat vakiintuneet tietyn sisältöisiksi. Erl on löytänyt 19 tällaista SOA:n idean laajennusta, ja niistä tärkeimpinä hän mainitsee nykyisen SOA:n pohjautuvan avoimille standardeille, olevan arkkitehtuurisesti yhdisteltävissä ja kykenevän parantamaan palvelun laatua. Hän mainitsee nykyisen SOA:n myös tukevan ja kannustavan muun muuassa jatkuvaan palveluiden uudelleen käyttöön, riippumattomuuteen ohjelmistotoimittajista, palvelukeskeiseen liiketoimintamallinnukseen sekä palveluiden välisiin löyhiin kytkentöihin yritystasolla.

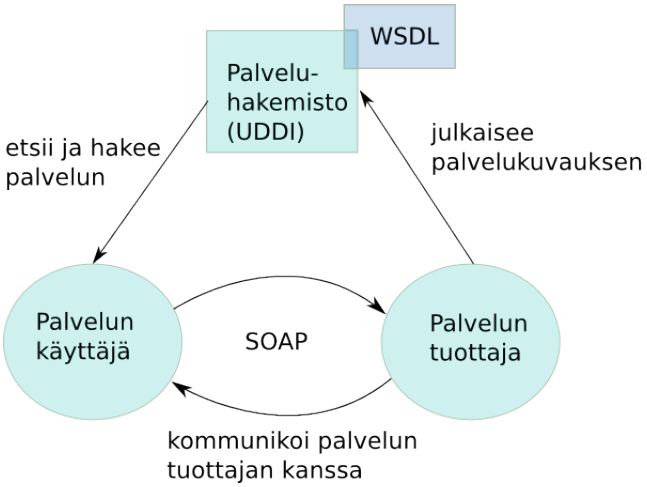
2.2.1 Palvelun määrittely

Palvelua voidaan pitää palvelukeskeisen arkkitehtuurin keskeisenä komponenttina. Erl [Erl05] käyttää termiä *alkukantainen SOA (Primitive SOA)* kuvaamaan palvelua ja sen tärkeimpiä ominaisuuksia. Näitä palveluiden keskeisiä ominaisuuksia ovat palvelukeskeinen analogia, logiikan kätkeminen, palveluiden välinen suhde toisiinsa, niiden välinen kommunikointi sekä palveluiden suunnittelu- ja toteutusmallit. Suunnittelumallien osalta Erl [Erl05, Erl07] mainitsee kahdeksan keskeistä palvelun suunnitteluperiaatetta. Nämä periaatteet ja niiden keskeiset ominaisuudet ovat:

- Löyhä kytkentä (Loose Coupling): Palvelut ovat mahdollisimman vähän riippuvaisia toisistaan ja käytännössä ainoastaan säilyttävät tietoisuuden muiden palveluiden olemassa olosta.
- Palvelusopimus (Service Contract): Palvelut noudattavat palvelukuvausten ja muiden dokumenttien kuvaamaa yhteistä kommunikointisopimusta.
- Itsenäisyys (Autonomy): Palveluilla on täysi kontrolli omaan logiikkaansa.
- Abstraktio (Abstraction): Palvelu piilottaa sellaisen logiikan, mikä ei ilmene palvelukuvauksesta.
- Uudelleenkäytettävyys (Reusability): Logiikka on paloiteltu palveluiden kesken siten, että se edistää uudelleenkäyttöä.

- Yhdistettävyys (Composability): Useita palveluita voidaan hallita ja yhdistää siten, että niistä voidaan muodostaa yhdistelmäpalveluita.
- Tilattomuus (Statelessness): Palvelut pyrkivät minimoimaan toimintaan liittyvän tiedon säilyttämisen.
- Löydettävyys (Discoverability): Palveluiden kuvaukset suunnitellaan siten, että palvelut voidaan niiden avulla helposti paikallistaa ja että niiden sisältö voidaan arvioida.

Palvelukeskeinen arkkitehtuuri voidaankin kuvata yksinkertaistettuna kolmen keskeisen komponentin avulla: 1) *palvelun tuottaja (Service Provider)*, 2) *palvelun käyttäjä (Service Requestor)* ja 3) *palvelurekisteri (Service Registry)*. Kuvassa 1 esimerkkinä web-palvelutekniikkaan pohjautuva palvelumalli, jossa palvelun tuottaja tuottaa palvelun ja julkaisee palvelun kuvauksen UDDI (Universal Description Discovery and Integration) -hakemistoon. Palvelun kuvaus on WSDL (Web Services Description Language) -muodossa. Palvelun käyttäjä etsii tarvitsemaansa palvelua palvelukuvausten pohjalta ja hakee sen hakemistosta. Palvelun käyttö tapahtuu kommunikoimalla palvelun tarjoajan kanssa SOAP-protokollalla. Palvelurekisteristä käytetään termiä palveluvälittäjä (Service Broker), jos käytössä on useita palvelurekistereitä, joita käsitellään välittäjän kautta [Jos07].



Kuva 1: Palveluarkkitehtuurin osapuolet [Erl05].

Palveluiden löyhä kytkentä on yksi keskeisimpiä edellä esitetyistä suunnitteluperiaatteista. Palvelun hienojakoisuus liittyy niiden väliseen löyhään kytkentään.

Mitä hienojakoisempi palvelu, sen helpompi se on pitää irrallaan muista palveluista. Hienojakoisuus helpottaa myös palveluiden uudelleenkäytettävyyttä. Onnistuneesti määriteltyä palvelua on helpompi uudelleenkäyttää, sillä sen kutsurajapinta on monipuoliseen käyttöön soveltuva. Hienojakoisista palveluista on myös helppo tehdä yhdistelmäpalveluita (Composite Service). [Erl07]

Papazoglou [Pap07] jakaa palvelut kahteen eri kategoriaan niiden monimutkaisuuden mukaan. Yksinkertaisia palveluita (Informational Services) ovat sellaiset palvelut, jotka pääsääntöisesti tuottavat käyttäjälle informaatiota pyyntö/vastaus (Request/Response) -tyyppisesti tai tarjoavat taustajärjestelmän liiketoimintapalvelun toiselle sovellukselle. Monimutkaiset palvelut (Complex Services) käyttävät tyypillisesti hyödykseen useita muita palveluita. Ne sisältävät mahdollisesti myös ohjelmointilogiikan kaltaista päättelyä tai vaativat käyttäjän vuorovaikutusta kesken palvelun suoritusta. Monimutkainen palvelu on siis käsitteellisesti jo lähellä prosessia.

2.2.2 Palveluväylä

Palvelukeskeinen arkkitehtuuri koostuu yleensä eri teknologioilla toteutetuista järjestelmistä. Näiden järjestelmien integrointi keskenään on hankalaa johtuen osittain teknologista eroista, mutta myös eroista niiden sisältämän tiedon suhteen. Nämä seikat voidaan ratkaista joko toteuttamalla tarvittava logiikka jokaiseen moduliin tai ottamalla käyttöön keskitetty kommunikointiväylä, jossa pyritään ratkaisemaan näitä ongelmia. [PvdH07]

Palveluväyläohjelmistot olivat alunperin viestiväylä-pohjaisia ohjelmia (Messge Oriented Middleware), joiden päätehtävä oli toimittaa saamansa viestit perille kohdejärjestelmään. Sittemmin näihin ohjelmiin tuli lisää ominaisuuksia, kuten viestien muokkaus, keskitetyt tietoturva- ja auditointiratkaisut sekä tuki erilaisille protokollille. [PvdH07]

Palveluväylä (Enterprise Service Bus, ESB) on yksi keskeisiä palvelukeskeisen arkkitehtuurin toteutusteknologioista. Palveluväylän avulla voidaan toteuttaa joitakin keskeisimpiä palvelukeskeisen arkkitehtuurin vaatimuksista. ESB on tyypillisesti avoin viestipohjainen väylä palveluiden liittämiseksi toisiinsa. Se mahdollistaa palveluille yhtenäisen liitäntärajapinnan, ikään kuin väylän, jonka kaut-

ta palvelut voivat kommunikoida keskenään ilman, että niiden on kytkeydyttävä suoraan toisiinsa. Papazoglou [Pap07] mainitsee tyypillisimpinä palveluväylän ominaisuuksina:

- Olemassa olevien toteutusten hyödyntäminen (Leveraging Existing Assets): Tavoitteena on olemassa olevien perinnejärjestelmien sovellusten hyödyntäminen.
- Palveluiden välinen kommunikaatio (Service Communication): Palveluiden välisen kommunikaation on toimittava protokollista riippumatta ja useilla eri protokollilla. Palvelukontekstiin liittyvä lisäinformaatio kuten tietoturvaelementit, transaktiokonsepti ja viestin korrelaatioriippuvuus on pystyttävä välittämään palveluiden kesken.
- Dynaaminen liitettävyys (Dynamic Connectivity): Palveluiden tulee pystyä liittymään keskenään dynaamisesti ilman staattista ohjelmointirajapintaa (API) tai jokaiselle palvelulle olemassa olevaa välityspalvelinta (Proxy). Palveluiden toteutusten täytyy olla myös toteutusprotokollasta (esimerkiksi SOAP ja RMI) riippumattomia.
- Aihe- ja sisältöpohjainen reititys (Topic/Content-Based Routing): Palvelun reititys täytyy olla mahdollista myös sisältöpohjaisesti pohjautuen deklaratiiviseen reitityssääntöön eikä pelkästään aihepohjaisesti.
- Muunnokset ja mappaus (Transformation and Mapping): Varmistetaan, että kaikkien palveluiden saama tieto on siinä muodossa, missä ne odottavat. Tämä toteutetaan muunnosoperaatioilla, joista kevein on mappaus. Mappauksen ja muunnosten takia osapuolten ei tarvitse olla tietoisia toistensa toteutusten yksityiskohdista.
- Palvelun orkestrointi, aggregointi ja prosessin hallinta (Service Orchestration, Aggregation and Process Management): Hienojakoiset palvelut tulisi yhdistää karkeamman tason palveluiksi. Tähän tarkoitukseen on olemassa myös erillisiä tuotteita.
- Palvelun löydettävyys ja palvelutaso (Endpoint Discovery With Multiple QoS: Palvelukeskeisessä arkkitehtuurissa on löydettävä, paikallistettava ja kytkettävä palveluita toisiinsa eri liiketoimintasisällöillä. Useilla eri malleilla

on ratkaistavissa, että palvelut löytävät ajonaikaisesti kulloinkin parhaan mahdollisen palveluinstanssin käyttöönsä.

- Luotettava viestinvälitys (Reliable Messaging): Tukee asynkronista talleta ja välitä mallia viestien välityksessä. Myös taattu viestin perilletoimitus on keskeinen vaatimus.

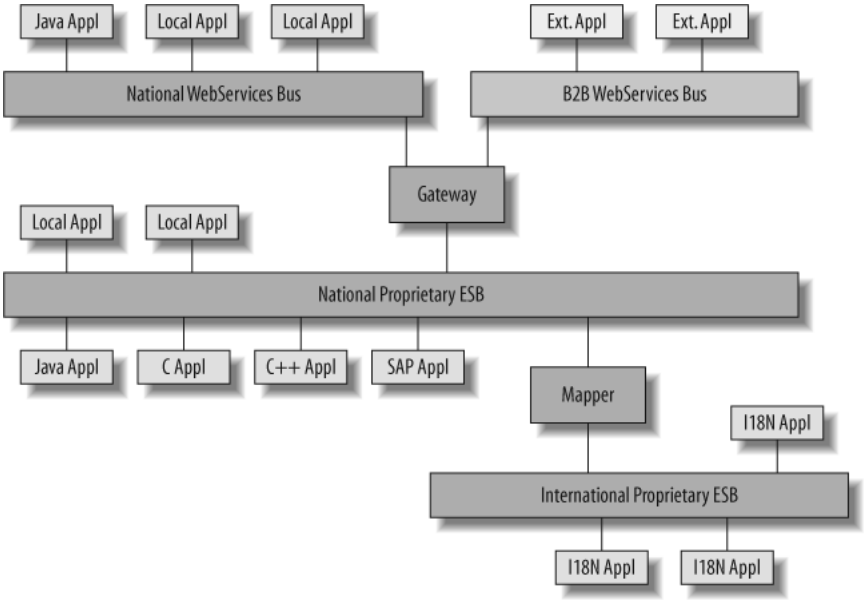
- Tietoturvamalli (Security Model): Palvelun käyttäjille on tarjottava tietoturvamalli sekä tuettava useita eri tapoja totetuttaa turvallisuus palveluiden suuntaan. On tuettava turvallisuutta palveluiden välillä sekä tarjottava päästä päähän toimiva tietoturvamalli.

- Monitorointi ja hallinta (Monitoring and Management): Palveluiden hallinta ja monitorointi on pystyttävä toteuttamaan siten, että se ulottuu järjestelmien rajojen ylitse.

Monimutkaisessa järjestelmäkokonaisuudessa on mahdollista käyttää myös useita palveluväyliä, jotka on kytketty toisiinsa erillisellä sillalla (Gateway). Pääsääntöisesti saman ohjelmistovalmistajan ohjelmien käyttö on helpompaa, mutta tietyistä syistä palveluväyläohjelmistot voivat olla myös erilaisia. Toisinaan joitakin palveluita halutaan eristää verkkoteknisesti muista palveluista vaikkapa tietoturvasyistä. Kuvassa 2 on esimerkki järjestelmäkokonaisuudesta, jossa on useita eri palveluväyliä (Heterogeneous ESBs). [Jos07]

2.2.3 Liiketoimintaprosessien mallinnus

Palvelukeskeisessä arkkitehtuurissa käytetään liiketoiminnallisen tapahtuman suorittavasta palvelusta termiä liiketoimintapalvelu. Kun yhden tai useamman tällaisen palvelun suoritusta halutaan kontrolloida ajonaikaisessa ympäristössä, puhutaan *liiketoimintaprosessien hallinnasta (Business Process Management, BPM)*. Palvelun hallinta kattaa prosessin tilan hallinnan, myös pitkäkestoisessa prosessissa sekä prosessin tarvisemien palveluiden löytämisen ja niiden kutsumisen. *Liiketoimintaprosessien mallinnus (Business Process Modeling, BPM)* liittyy myös prosessien hallintaan, mutta vain suunnitteluvaiheessa. Mallinnuksessa suunnitellaan prosessin toiminta loogisella tasolla ja suunnittelun tuloksena syntyy jollakin prosessin suoritusta kuvaavalla kielellä (esimerkiksi BPEL) ajon-



Kuva 2: Järjestelmäkokonaisuus, jossa on useita palveluväyliä [Jos07].

aikaista hallintaa varten kuvaustiedosto. Liiketoimintaprosessi voi sisältää myös manuaalisia ihmisen suorittamia tehtäviä.

Palveluprosessien kuvauskieliä on useita ja niitä määrittelevät useat eri standardointiorganisaatiot. Toistaiseksi yleisimmät ovat olleet Web Services Choreography Description Language (WS-CDL), Business Process Execution Language for Web Services (WS-BPEL), Business Process Modeling Notation (BPMN), Unified Modeling Language (UML), Process Definition Language (XPDL). Näistä OASIS-organisaation standardoima WS-BPEL eli lyhyemmin BPEL näyttää saavuttaneen tällä hetkellä laajimman suosion mallinnuskielenä. Liiketoimintaprosessien yhteydessä puhutaan usein myös työnkulusta (Workflow). Työnkulku kuvaa kuinka jonkin tehtävän tulokset voidaan saavuttaa. Prosessin suorituksen kuvaus, esimerkiksi BPEL kuvaus, on osa tällaista työnkulkua. [Jos07]

Liiketoimintapalvelu voi olla yksittäinen palvelu tai se voi olla yhdistelmäpalvelu, jolloin se hyödyntää muita palveluita. Erl [Erl05] jakaa palvelut kahteen kategoriaan myös niiden suorituksen kestoajan mukaan. Hän käyttää termiä atominen tapahtuma (Atomic Transaction), kun tapahtuma pystyy oman suorituksensa jälkeen toteamaan yksikäsitteisesti oliko se onnistunut vai ei. Tarvittaessa palvelu pystyy tämän jälkeen tekemään tapahtumankäsittelyjärjestelmälle tapahtuman vahvistus- tai peruutuspyynnön (Commit/Rollback). Pitkäkestoista tapahtumaa Erl kutsuu liiketoimintatapahtumaksi (Business Activity). Tällaisen tapahtuman

kestoaika on tyypillisesti tunteja, päiviä tai viikkoja ja se saattaa vaatia myös manuaalisia ihmisen suorittamia operaatioita.

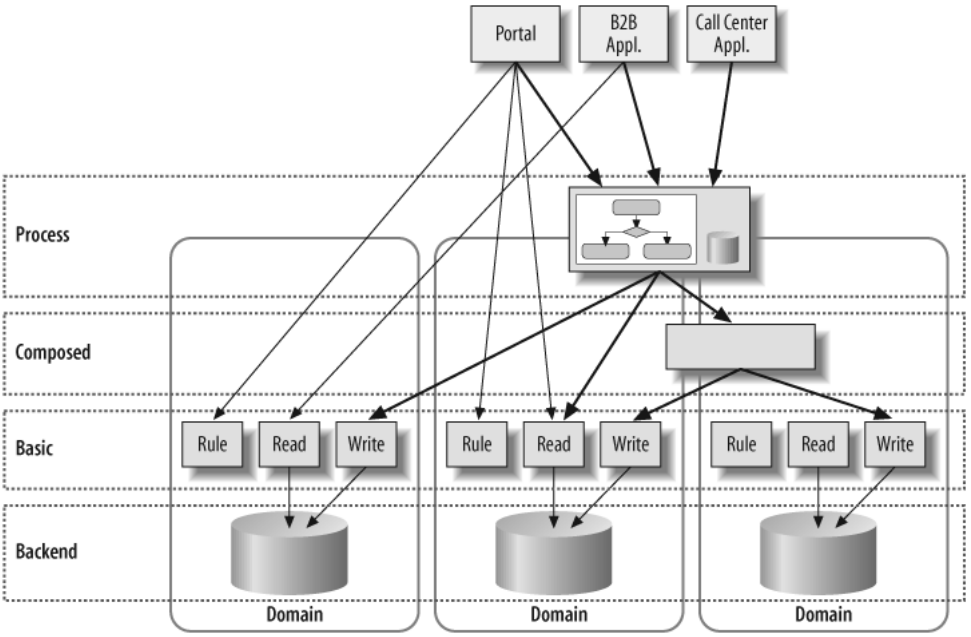
Prosessien hallinnan yhteydessä puhutaan usein prosessien orkestroinnista (Process Orchestration) ja koreografiasta (Process Choreography). Orkestrointi mahdollistaa palveluiden välisen kommunikoinnin ilman, että sitä toteutetaan palvelukohtaisesti. Tyypillisesti tämä toteutetaan keskitetyn palveluväylän avulla, jolloin kaikki palveluiden välinen kommukointi käyttää tätä väylää. Arkkitehtuurimallina orkestrointi oli käytössä jo ennen palvelukeskeisen arkkitehtuurin yleistymistä, erityisesti yritysintegrointimalleissa [Erl05]. Mallinnuskielistä monet, esimerkiksi BPEL, ovat hyvin orkestrointi-orientoitununeita [Jos07].

Toisinaan prosessiin liittyy eri organisaatioiden välistä vuorovaikutusta ja yhteistyötä. Tällaisen prosessin suorittamisessa koreografiaa voidaan käyttää hyödyksi. Koreografiassa prosessiin osallistuvien osapuolten täytyy vaihtaa niiden väliseen vuorovaikutukseen liittyviä sääntöjä keskenään. Tämä vuorovaikutus voidaan toteuttaa käyttämällä siihen sopivaa kuvauskieltä. Erl [Erl05] mainitsee esimerkkinä tällaisesta kielestä WS-CDL:n. Keskeisin ero orkestroinnin ja koreografian välillä on se, että orkestroinnissa prosessin ohjausta suorittaa ulkoinen osapuoli ja koreografiassa ohjaus suoritetaan osapuolten kesken [Jos07].

Palveluilla, yhdistelmäpalveluilla ja prosesseilla on suhde toisiinsa sekä niitä käyttäviin sovelluksiin. Kuvassa 3 kuvataan miten atomiset palvelut (Basic) käsittelevät taustajärjestelmien tietoja ja yhdistelmäpalvelut (Composed) useita atomisia palveluita. Kuhunkin prosessiin voi liittyä sekä atomisia palveluita että yhdistelmäpalveluita. Kuvan esimerkkisovellukset puolestaan käyttävät atomisia palveluita, yhdistelmäpalveluita ja prosesseja. [Jos07]

2.3 Palvelukeskeisen arkkitehtuurin standardointi

Palvelukeskeisen arkkitehtuurin abstraktista perusrakenteesta huolimatta, liittyy siihen ja sen toteutukseen teknologisia valintoja. Teknologisista valinnoista palveluiden rajapintojen toteutustapa on keskeisin. Web-palvelutekniikka on siinä kaikkein yleisimmin käytetty toteutustapa ja SOA:n yleistyminen liittyykin paljolti juuri web-palvelutekniikan yleistymiseen. Web-palveluiden (Web Service) toteutus ei vielä tarkoita, että tehtäisiin palvelukeskeisen arkkitehtuurin mukaisia



Kuva 3: Palvelut, yhdistelmäpalvelut ja prosessit [Jos07].

palveluita. Myöskään SOA:n toteutus ei vaadi web-palvelutekniikan käyttöä, vaan toteutusteknologiana voi olla jokin muukin, kuten esimerkiksi REST (Representational State Transfer), SOAP Over JMS, XML, XML Over RPC (XML-RPC), Corba tai JavaScript Object Notation Over RPC (JSON RPC). [SBC+07]

Palvelukeskeiseen arkkitehtuurin perusperiaatteista mm. löyhä kytkentä, itsenäisyys ja tilattomuus ovat mahdollistavia tekijöitä sille, että järjestelmien toteutuksia voi olla useilla eri teknologioilla, eri laitteistoarkkitehtuureilla sekä eri ohjelmistotoimittajilta. Tämän mahdollistamiseksi täytyy eri tuotteiden välillä olla jotain yhteisesti sovittua kuten kommunikointiprotokollat sekä tiedon ja palveluiden kuvausmekanismit. Eri ohjelmistovalmistajat tuovat usein omia ratkaisujaan mielummin muiden käyttöön, kuin käyttävät toisten tekemiä. Toisinaan myös omia ratkaisuja pyritään kilpailusyistä pitämään suljettuina. Palvelukeskeisessä arkkitehtuurissa tarvitaan kuitenkin yhteisesti sovittuja pelisääntöjä. Eri valmistajien tuotteiden toimivuus keskenään voidaan varmistaa vain avoimilla standardeilla, joita kaikki osapuolet noudattavat.

Nopeasti kehittyvällä alalla standardin määritelmä ei aina ole yksiselitteinen. Toisinaan samaan asiaan liittyy myös useita keskenään kilpailevia standardeja tai standardiehdotuksia. Toisinaan on myös epäselvää mikä on standardi ja mikä ei. Standardi (Standard) on hyväksytty teollisuusstandardi. SOA-ympäristössä

esimerkiksi kaikki ensimmäisen sukupolven web-palvelutekniikat ja useat XML-standardit ovat saavuttaneet tämän tilan. Määrittely (Specification) puolestaan on joko ehdotettu tai hyväksytty standardi. Kaikki edellä esimerkkeinä mainitut standardit kuuluvat tähän joukkoon sekä niiden lisäksi muun muuassa WS* laajennukset. Laajennuksia (Extension) puolestaan ovat esimerkiksi WS* määrittelyt. [Erl05]

Standardeja kehittää usea eri organisaatio, joista ISO (International Organization for Standardization) lienee tunnetuin. Palvelukeskeisen arkkitehtuurin standardien kehityksessä on vahvasti ollut esillä web-palvelutekniikoiden kehittäminen, usein puhutaankin WS* standardeista. Erl [Erl05] mainitsee kolme organisaatiota, joiden merkitys ja vaikutus palvelukeskeisen arkkitehtuurin standardoinnin alueen standardien kehittymiselle on merkittävää.

World Wide Web Consortium (W3C) on vanhin Erlin [Erl05] mainitsemista organisaatioista. Se on perustettu vuonna 1994 ja on ollut vahvasti vaikuttamassa Internetin kehittymiseen erityisesti HTTP-protokollan (Hypertext Transfer Protocol) ja HTML-kuvauskielen (Hypertext Markup Language) kehittämisen avulla. Palvelukeskeisen arkkitehtuurin osa-alueella voidaan näiden lisäksi mainita muun muassa SOAP-protokolla ja XML-kuvauskieli (Extensible Markup Language) sekä useita web-palvelutekniikka -standardeja.

Toinen Erlin [Erl05] mainitsema organisaatio on *Organization for the Advancement of Structured Information Standards (OASIS)*. Se on perustettu vuonna 1998 ja se on jäsenmäärältään suurin näistä kolmesta. Se on muodostettu vuonna 1993 toimintansa alunperin aloittaneen SGML Open järjestön tilalle. Vuonna 1998 toiminnan painopiste vaihtui SGML standardeista XML-standardien määrittelyyn ja nimi vaihdettiin samaan aikaan. OASIS on keskittynyt erityisesti määrittelemään XML- ja web-palvelutekniikka -standardien tietoturvalaajennoksia. Sen määrittelyalueelle kuuluu myös SOA prosessien suorituksen määrittelyyn yleisesti käytetty WS-BPEL.

Web Services Interoperability Organization (WS-I) on kolmas Erlin [Erl05] mainitsema organisaatio ja se on aloittanut toimintansa vuonna 2002. Sen tarkoituksena ei ole luoda uusia standardeja, vaan huolehtia, että standardeista saadaan niin toimivat ja avoimet, että ne mahdollistavat eri valmistajien tuotteiden välisen yhteiskäytön. Tämän järjestön kasvu on ollut voimakasta ja käytännössä

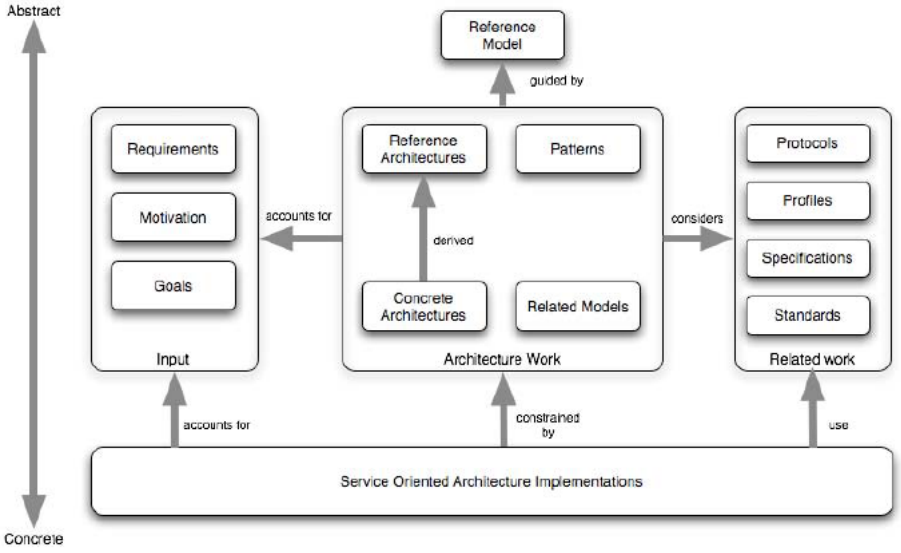
kaikki SOA alueelle ohjelmistoja tekevät yritykset ovat mukana sen toiminnassa. WS-I määrittelee lähinnä erilaisia profiileja, joissa määritellään mitä standardeja kunkin profiilin mukaisessa toteutuksessa täytyy noudattaa, että saavutetaan riittävän tasoinen yhteiskäyttö eri tuotteiden välillä.

Edellä mainittujen kolmen Erlin [Erl05] mainitseman lisäksi voidaan mainita *OpenSOA (OSOA)* -yhteenliittymä. Se koostuu palvelukeskeisen arkkitehtuurin alueella toimivista ohjelmistotoimittajista ja he pyrkivät määrittelemään ohjelmointikieli- ja teknologiariippumattomia rajapintoja palvelukeskeisen arkkitehtuurin alueelle. OSOA ei tuota standardeja, vaan pyrkii edistämään ja kehittämään määrittelyitä. Kun nämä määrittelyt ovat riittävän kypsiä, ne viedään jollekin standardointiorganisaatiolle hyväksyttäväksi ja jatkokehitettäväksi. OSOA:n merkittävimmät tulokset ovat olleet SDO ja SCA määrittelyiden luovutus OASIS-järjestölle standardointia varten.

2.3.1 OASIS SOA-viitekehys

OASIS määrittelee useita XML-pohjaisia standardeja ja on näiden lisäksi määritellyt viitekehyksen palvelukeskeisen arkkitehtuurin kuvaamiseen (Reference Model for Service Oriented Architecture). Kuvaus esittelee palvelukeskeisen arkkitehtuurin abstraktin viitekehyksen (Framework). Viitekehys kuvaa SOA:n keskeiset käsitteet ja niiden väliset suhteet. Sen tarkoituksena on toimia teknologia- ja ohjelmistotoimittajariippumattomana kuvauksena, jonka avulla palvelukeskeisen arkkitehtuurin eri käsitteet ja niiden kytkökset eri arkkitehtuureihin ja teknologioihin voidaan kuvata, kuten kuvassa 4. [MLM+06]

OASIS SOA-viitekehyksen on tarkoitus toimia irrallaan teknologioista ja ohjelmistotoimittajista, mutta myös irrallaan varsinaisista referenssiarkkitehtuurikuvauksista. Se kuvaa suuren osan palvelukeskeisen arkkitehtuurin käsitteistä. Sen sijaan, että kuvauksissa otetaan kantaa siihen miten käsitteitä tulisi toteuttaa, se keskittyy kuvaamaan mitä käsitteitä tulisi varsinaisessa viitearkkitehtuurikuvauksessa kuvata.



Kuva 4: OASIS palvelukeskeisen arkkitehtuurin viitekehys [MLM+06].

2.3.2 Palvelukeskeisen arkkitehtuurin standardeja

Palvelukeskeinen arkkitehtuuri ja erityisesti sen toteutukset pohjautuvat vahvasti standardeihin ja niiden tiukka noudattaminen on keskeistä eri ohjelmistojen ja teknologioiden yhteensopivuuden kannalta. Web-palvelutekniikan varaan toteutettuun palvelukeskeisen arkkitehtuurin ratkaisuun liittyy myös muita standardeja, joista WSDL, SOAP ja UDDI ovat keskeisimmät HTTP:n ja XML:n lisäksi [Erl05].

Palvelukeskeisen arkkitehtuurin ohjelmointimalli, IBM:n mukaan, sisältää SCA, SDO ja BPEL määrittelyiden käyttöä palveluiden ja prosessien toteutuksessa [SBC+07]. Näistä määrittelyistä SCA ja SDO ovat tällä hetkellä erityisesti IBM:n ja Bea Systemsin tuotteissaan käyttämiä ja niiden standardointi on siirretty OASIS-organisaatiolle, mutta kattavaa hyväksyntää teollisuusstandardina ne eivät ole vielä saavuttaneet. BPEL on myös kehittymässä standardiksi OASIS-organisaatiossa.

WSDL on web-palveluiden XML-pohjainen kuvauskieli. Se koostuu kolmesta osasta, joista ensimmäinen kuvaa palvelun operaatiot syöttö- ja tulostustietoineen. Toinen osa kuvauksesta kertoo palvelun kytkökset eli mitä protokollia ja tietomuotoja palvelu tukee. Kolmannessa osassa kerrotaan palvelun fyysinen osoite, esimerkiksi Uniform Resource Locator (URL). WSDL-määrittelyn versiot 1.1 ja

2.0 poikkeavat kieliopiltaan jonkin verran toisistaan, mutta sisältävät samat tiedot. Esimerkki WSDL-kuvaustiedostosta on liitteessä 1 [Jos07]

SOAP oli ensimmäinen varsinainen web-palvelutekniikkaan liitetty standardi. Myöhemmissä toteutuksissa kävi ilmi, että vaikka SOAP lyhenne tulee termeistä *Simple Object Access Protocol*, eivät toteutukset olleet kuitenkaan niin yksinkertaisia. SOAP määrittelyn version 1.2 myötä tuosta termistä onkin luovuttu ja toisaalta lyhenne on jo niin yleistynyt, että se kuvaa asiaa riittävästi. SOAP on myöskin XML-pohjainen kuvauskieli. SOAP-tiedostoa käsittelevät tyypillisesti erilaiset muunninohjelmistot (Adapter), joten sen sisällön tarkka ymmärrys ei ole välttämätöntä. Karkealla tasolla SOAP-tiedosto (liite 2) koostuu otsakeosasta (Header) ja varsinaisesta viestistä (Body). Otsake saattaa sisältää valinnaisia osuuksia, kuten tietoturvaratkaisun määrittelyitä. [Jos07] SOAP-viesti voidaan välittää useita eri kuljetusprotokollia käyttäen. Yleisin näistä protokollista on HTTP, joskin JMS on myös kohtuullisen yleisesti käytössä erityisesti viestipohjaisissa ratkaisuissa.

UDDI oli alunperin hieman laajempi määrittely nimeltään Universal Description, Discovery, and Integration Business Registry (UBR) ja sen tarkoitus oli toimia maailman laajuisena palveluhakemistona palvelun tuottajien, palvelun käyttäjien ja palveluiden välillä. Vuoden 2006 lopulla sen määrittelystä luovuttiin ja se jäi elämään hieman suppeammassa roolissa lähinnä paikallisena UDDI-hakemistona. Palveluhakemistolle on olemassa muitakin kilpailevia määrittelyitä, joten sen rooli tulevaisuudessa on SOA:n keskeisistä määrittelyistä hatarin. [Jos07]

Palvelukomponenttiarkkitehtuuri (Service Component Architecture, SCA) on keskeinen osa useiden ohjelmistovalmistajien SOA-toteutusta. SCA on teknologiariippumaton kuvaus palveluiden rajapintojen ja toteutusten mallintamisesta. Korkean tason malli mahdollistaa varsinaiseen liiketoimintapalveluun keskittymisen, eikä aikaa tarvitse käyttää toteutusteknologian miettimiseen. SCA kuvaa mallin palveluiden toteuttamiseen ja niiden rajapintojen esittämiseen. Se mahdollistaa myös palveluiden toteuttamisen useilla eri ohjelmointikielillä. Toteutuskieliä ovat sekä oliopohjaiset kielet, kuten C++ ja Java, proseduraaliset kielet kuten Cobol, XML-orientoituneet kielet kuten WS-BPEL ja XSLT sekä deklaratiiviset kielet kuten XQuery ja SQL. SCA-mallissa toteutetaan palvelusta rajapinta (Interface), viite (Reference) ja toteutus (Implementation). [KBCG06]

SCA-mallin osista toteutus sisältää jollakin kielellä tehdyn toteutuksen liiketoimintapalvelusta. Viitteiden avulla palvelut kuvaavat mitä muita palveluita ne tarvitsevat. Sekä palvelut että niiden viitteet tyypitetään rajapinnoin. Näiden rajapintojen tyyppi on vapaasti valittavissa (esimerkiksi Java tai WSDL portType), mutta SCA-malli sinällään suosii yksinkertaista WS-I:n standardoimaa pyyntö/vastaus-mallin (Single-input/Single-output) mukaista toteutustapaa. [CFNS05]

Palvelutieto-olio (Service Data Object, SDO) määrittelee ohjelmointikielistä riippumattoman mallin tiedon käsittelyyn. Mallia voidaan käyttää useiden eri tietolähteiden sisältämän tiedon käsittelyyn. Nämä tietolähteet voivat olla myös keskenään erityyppisiä. Mallin avulla voidaan yhtenäistää erilaisten tietolähteiden tiedon käsittelymekanismit ja eriyttää tiedonkäsittely muusta ohjelmalogiikasta. Se tukee sekä staattisia että dynaamisia ohjelmointikirjastoja ja mahdollistaa tiedon offline-käsittelyn (Disconnected Usage Pattern of Data Access). SDO määrittelyitä on toteutettu ainakin Java, C, C++ ja Cobol-kielille. [Res07]

3 Suurkoneet palvelukeskeisessä arkkitehtuurissa

Suurkoneesta (Mainframe Computer) käytetään usein myös nimitystä keskustietokone. Toisinaan se sotkeutuu kuitenkin termiin supertietokone (Supercomputer), joka tarkoittaa hieman erilaista, yleensä tiettyyn käyttöön tarkoitettua, tehokasta tietokonetta. Suurkoneesta on aikanaan käytetty myös nimeä Big Iron [EOO06], jolla tehtiin eroa nimenomaan pienempiin osastokohtaisiin tietokoneisiin.

Suurkonetta käytetään paljon yrityksissä ja tyypilliset käyttötavat ovat muun muuassa suurten tietomäärien käsittely sekä tapahtumakeskeisten järjestelmien tapahtumien hallinta. Näiden koneiden hinnat ovat aina olleet suhteellisen korkeita, joten niitä on käytössä lähinnä suurissa yrityksissä ja suurissa tietojärjestelmissä. Suurkoneet poikkeavat supertietokoneista siinä, että niille on tyypillistä tietynlainen yleiskäyttöisyys. Samaa konetta voidaan siis käyttää hyvinkin erilaisissa käyttötarkoituksissa myös yhtäaikaisesti, mikäli koneen kapasiteetti riittää. [EOO06]

Termi suurkone on osittain peräisin tietokoneiden historiasta. Ensimmäiset suurkoneet olivat kooltaan suuria, jopa huoneen kokoisia, joten tämä nimi oli aikanaan hyvinkin kuvaava. Noista ajoista koneiden koko on pienentynyt huomattavasti teknologian kehittymisen myötä. Nykyiset suurkoneet eivät juurikaan eroa kooltaan esimerkiksi suurimmista unix-palvelimista. Unix-palvelimet ovat myös lähentyneet arkkitehtuuriltaan ja suorituskyvyltään suurkoneita, joskin merkittäviäkin eroja vielä löytyy. Unix-tyyppisistä koneista käytettiin yhteen aikaan termiä minitietokone. Tällöin näissä laitteissa käytettiin yleensä valmistajien omia käyttöjärjestelmiä, kun nykyään ne ovat pääsääntöisesti unix-pohjaisia.

Hunter [Hun06] toteaa, että keskustietokone käsitteenä on hämärtymässä, sillä monet näistä perinteisistä valmistajista käyttävät koneissaan nykyään yhä useammin suorittimena Sparc- tai Intel-suorittimia ja käyttöjärjestelmänä Unixia tai Linuxia perinteisen oman suorittimen ja käyttöjärjestelmän sijaan.

Suurkoneet yhdistyvät usein IBM:n suurkoneisiin ja esimerkiksi Ebbers & al. [EOO06] toteavat termin käytön alkaneen varsinaisesti 60-luvulla IBM:n System/360 sarjan koneiden myötä. Nämä koneet olivat ensimmäisiä, joissa oli käytössä standardoituja laitteistokomponentteja ja ohjelmia. Tästä syystä niistä tuli ensimmäisiä yleiskäyttöisiä tietokoneita. Suurkoneiden valmistajia on ollut ja on

edelleenkin muitakin yrityksiä, kuten Bull, Fujitsu-Siemens, Unisys, Hitachi ja Burroughs.

1990-luku oli suurkoneille suuren kriisin aikaa muun muassa erilaisten minitietokoneiden ja hajautettujen klusteroitujen järjestelmien menestyksen sekä joidenkin suurkonevalmistajien laskusuhdanteen takia. Hunter [Hun06] mainitsee, että 2000-luku on hieman yllättäen ollut suurkoneiden uuden tulemisen aikaa 1990-luvun heikon menestyksen jälkeen. Hän toteaa, että menestys on tosin vahvasti yhdistynyt IBM:n zSeries suurkone-sarjan menestykseen ja että jopa 490 Fortune500-listan yrityksistä käyttää zSeries sarjan laitetta liiketoimintakriittisen sovelluksen ajoympäristönä.

3.1 IBM-suurkone

IBM-suurkoneen varsinaisena alkuna voidaan pitää vuonna 1964 esiteltyä System/360 sarjan laitetta. Sen vahvuutena oli ensimmäistä kertaa standardoidut laitteistokomponentit ja ohjelmistot. Seuraavien sukupolvien laitteet ovat hyödyntäneet näitä samoja periaatteita ja vaikka teknologinen kehitys on 1960-luvulta edennyt voimakkaasti nykypäivään, on osa tuon aikaisista ja erityisesti 70-luvulla kehitetyistä ohjelmista edelleen käyttökelpoisia. Koneiden arkkitehtuuri on 60-luvulta lähtien kehittynyt noin 10 vuoden välein tapahtuneiden suurempien muutosten lisäksi niiden välissä olleiden pienempien kehitysaskeleiden myötä.

System/360 laitteiston aikana kehitettiin IBM:n vielä nykyäänkin käytössä olevat tapahtumankäsittely- ja tietokannanhallintajärjestelmät Customer Information Control System (CICS) ja Information Management System (IMS). 1970-luvulla esiteltiin System/370, jonka yksi suurimmasta edistysaskelista oli moniprosessorituki. Alun perin koneet pystyivät käyttämään kahta suoritinta, mutta kehitysversioissa määrä kasvoi, kuten myös suorittimien teho. 1980-luvulla S/370XA version koneiden myötä osoiteavaruus kasvoi 24:sta 31 bittiin. Vuosikymmenen loppupuolella otettiin käyttöön koneiden jako niin sanoittuihin loogisiin osioihin (Logical Partitions). Loogisten osioiden myötä suurkone voitiin jakaa erillisiin koneisiin, jotka kuitenkin käyttivät samoja laitteistoresursseja. Tämä on myös yksi tärkeä modernin suurkoneen ominaisuus edelleenkin. [EOO06]

1990-luku oli System/390 sarjan (ESA/390) laitteiden aikaa. Menestyksen kannalta aika oli huonoa, mutta teknisiä parannuksia S/390-sarjan myötä tuli käyttöön. Näistä teknisistä parannuksista keskeisimpiä olivat järjestelmän klusterointi ja tiedon jakomekanismit (Parallel Sysplex), mahdollisuus kasvattaa koneen kapasiteettia järjestelmää uudelleen käynnistämättä sekä loogisten osioiden tuki laajeni kattamaan 15 loogista osiota. Varsinainen keskustietokoneen uudelleen tuleminen alkoi 2000-luvulla zSeries laitteiston myötä. zSeries pohjautui ESA/390 arkkitehtuuriin, mutta laajensi esimerkiksi osoiteavaruuden 64 bittiin jo sarjan ensimmäisessä z900 laitteessa. [EOO06]

zSeries arkkitehtuurin seuraava kone z990 lisäsi loogisten partitioiden maksimimäärän viidestätoista kolmeenkymmeneen. z990 laitteen keskeisin muutos oli laitteistoarkkitehtuurin muutos siten, että se mahdollisti neljän erillisen suoritinmodulin käytön entisen yhden sijaan. Harrer & al. [HKB06] mainitsevat, että tämän muutoksen johdosta z990 ja z9 sarjan koneiden prosessoreiden maksimimääräksi tuli 64. z9 sarjan koneet noudattavat pääsääntöisesti samaa arkkitehtuuria, mutta tarjoavat enemmän muistia ja prosessoreita sekä kasvattivat loogisten partitioiden maksimimäärän 60:een.

zSeries laitteiden myötä IBM:n suurkoneisiin on tullut myös erilaisia apuprosessoreita. Näiden prosessoreiden tarkoitus on siirtää kuormaa pois pääprosessoreilta ja tehdä koneiden käytöstä houkuttelevampaa (ja halvempaa) myös laajemman käyttäjämäärän palveluissa. Näistä apuprosessoreista yksi on nimeltään System z Integrated Information Processor (zIIP) ja sen käyttö helpottaa pääsuorittimien kuormaa erityisesti hajautetusta ympäristöstä tulevaan DB2 tietokannan käyttöön. System z Application Assist Processor (zAAP) puolestaan on tarkoitettu keventämään suurkoneella suoritettavien Java-ohjelmien pääprosessoreille aiheuttamaa kuormaa. Kolmantena apuprosessorina voidaan mainita salausprosessori (Crypto Engine). [KBCG06]

Suurkoneiden käyttöjärjestelmät ovat myös kehittyneet vuosien kuluessa. S/370 laitteiden alusta lähtien oli käytössä MVS-pohjaiset käyttöjärjestelmät. Niistä edellinen on 1990-luvun puolivälissä esitelty OS/390, joka korvaantui 2000-luvulla zSeries-laitteiden myötä z/OS-järjestelmällä. zSeries laitteet tukevat myös muita käyttöjärjestelmiä, kuten esimerkiksi z/VM, z/VSE, z/TPF ja Linux (for zSeries). Samassa koneessa voidaan suorittaa yhtäaikaisesti useita käyttöjärjestelmiä. [EOO06]

Unix System Services (USS) palvelu on z/OS-käyttöjärjestelmään sisäänrakennettu unix-ympäristö. Siitä käytetään myös nimeä z/OS Unix. Unix-palvelut tuotiin vuoden 1991 jälkeen suurkoneen käyttöjärjestelmään mukaan Yhdysvaltojen viranomaisten (FIPS) vaatimuksesta. Ensimmäiset toteutukset tehtiin MVS-järjestelmään ja nimenä oli silloin OpenEdition. OS/390 käyttöjärjestelmän aikana unix-palveluiden nimeksi tuli OS/390 Unix System Services. z/OS Unix toimii ajoympäristönä useille suurkoneen tietojärjestelmille, kuten esimerkiksi sovelluspalvelimille ja palveluväylille. [RAB+06]

3.2 Suurkone palveluarkkitehtuurin toteutusalustana

Palvelukeskeistä arkkitehtuuria toteutettaessa ovat olemassa olevat järjestelmät usein keskeisessä roolissa. Yksi tärkeimmistä motivaatioista SOA:n käyttöönotolle on nimenomaan entisten järjestelmien hyötykäyttö uudella tavalla ja niiden elinkaaren pidentäminen. Suurkone on usein tässä roolissa johtuen siitä, että palvelukeskeisen arkkitehtuurin tyypillisin käyttöönotto tapahtuu suurissa yrityksissä, ja koska tällöin siitä saatavat hyödyt ovat suurimmat. Suurilla yrityksillä on myös keskimääräistä usemmin yksi tai useampia keskustietokoneita käytössään. Niillä on usein myös muita laajoja IT-järjestelmiä, jolloin näistä arkkitehtuurimuutoksista saadaan hyötyä. Suurilla yrityksillä on myös takanaan usein pitkä historia, johon mahtuu yrityskauppoja tai fuusioita, joiden seurauksena on päädytty mahdollisesti hyvinkin heterogeenisten järjestelmien omistajaksi.

Suurkoneilla olevat tietojärjestelmät ovat tyypillisesti vanhoja ja niiden nykyiseen käyttöön liittyy tyypillisesti riskejä. Yksi keskeisistä huolenaiheista on järjestelmien ikääntyminen ja se kuinka järjestelmät mukautetetaan tukemaan nykyisiä liiketoimintamalleja. Toinen merkittävä haaste liittyy siihen, kuinka järjestelmät mukautuvat uuteen entistä hajautetumpaan tietojärjestelmämalliin sekä mahdollisesti merkittävästi kasvaviin käyttäjämääriin. [KBCG06]

Toisaalta suurkone soveltuu hyvin palvelukeskeisen arkkitehtuurin toteutusalustaksi useista eri syistä. Ebbers & al. [EBA+06] mainitsevat näiden syiden liittyvän yleensä seuraaviin kriteereihin: kapasiteetti (Capacity), skaalautuvuus (Scalability), eheys ja tietoturva (Integrity and Security), saatavuus (Availability), pääsy suuriin tietomääriin (Access to Large Amounts of Data), järjestelmän hallinta (Systems Management) ja itsenäisyys (Autonomic Capabilities). Näiden lisäk-

si keskitetty tietojenkäsittelymalli (Centralized Computing Model), virtualisointi ja kuormanhallinta (Virtualization and Workload Management), luotettavuus (Reliability), tapahtumankäsittely (Transaktion Processing) ja eräajojen käsittely (Batch Processing) mainitaan suurkoneen vahvuuksina palvelukeskeisen arkkitehtuurin kannalta [KdGRY06].

Keskustietokoneen operaatiot jakautuvat karkealla tasolla kahteen kategoriaan: eräajoprossit ja reaaliaikaprosessit. Eräajoprosessoinnissa yleensä käsitellään tietoa siten, että suoritetaan peräkkäin useita samanlaisia tapahtumia. Tapahtumat hyödyntävät tyypillisesti peräkkäistiedostoja syöttö- ja tulostusvirtoinaan, mutta toki ne voivat operoida myös muun kaltaisten tietolähteiden, kuten tietokantojen ja ulkoisten yhteyksien kanssa. Eräprosessi käynnistetään usein ajastamlla tai jonkin muun tapahtuman ohjaamana. Reaaliaikaprosessointi taas on tyypillisesti tapahtumakeskeistä (Online) ja kukin tapahtumapyyntö on toisistaan erillinen. [EOO06] Reaaliaikaprosessit ovat palvelukeskeiselle arkkitehtuurille tyypillisempiä.

Palvelukeskeistä arkkitehtuuria suurkoneelle toteutettaessa on ensin valittava strategia suurkoneen suhteen. Vaihtoehtoina voi olla suurkoneen käyttäminen tietovarastona tietokantapalvelimen roolissa, aktiivisena palveluita tuottavana osapuolena tai mitä tahansa tältä väliltä. Palveluiden toteutus voidaan toteuttaa integroitumalla tietokannanhallintajärjestelmään tai tapahtumankäsittelyjärjestelmään. Jälkimmäisessä tapauksessa palvelun toteutus voi olla myös suurkoneella. Mikäli suurkone on osallisena palveluiden tuottamisessa, se voi toimia myös järjestelmäalustana palveluväylälle. [KBCG06]

3.3 IBM Information Management System

IBM Information Management System (IMS) on vuonna 1968 julkistettu ohjelmisto tapahtumankäsittelyyn ja tietokannan hallintaan. Järjestelmän kehitys käynnistettiin vuonna 1966 ja sen alkuperäinen käyttötarkoitus liittyi Apollo-avaruusohjelman tukitoimintoihin. Tietojärjestelmäksi IMS on poikkeuksellisen iäkäs ja se täyttää tänä vuonna 40 vuotta. Näiden vuosien aikana ohjelmiston käyttöalusta (laitteet ja käyttöjärjestelmät) on muuttunut useaan kertaan. Se on alun perin valmistunut System/360 järjestelmän aikana ja nyt uusimmat versiot on toteutettu z-arkkitehtuurille ja z/OS-käyttöjärjestelmälle. Näiden vuosien ai-

kana on IMS-järjestelmästä julkaistu useita versioita (viimeisin on v10), joissa sen toiminta on jossain määrin muuttunut. [LHHN00]

IMS-ohjelmia on pitkään tehty monilla eri ohjelmointikielillä, kuten Assemblerilla, Cobolilla, C-kielellä, PL/1:llä, Pascalilla sekä REXX:llä [LHHN00] ja uusimmat versiot tukevat myös Javaa. Tietokannan talletusrakenne on IMS-järjestelmissä hierarkinen. Uusimmat versiot tukevat myös XML-muotoisen tiedon talletusta suoraan tietokantoihin ilman erillistä muunnosoperaatiota.

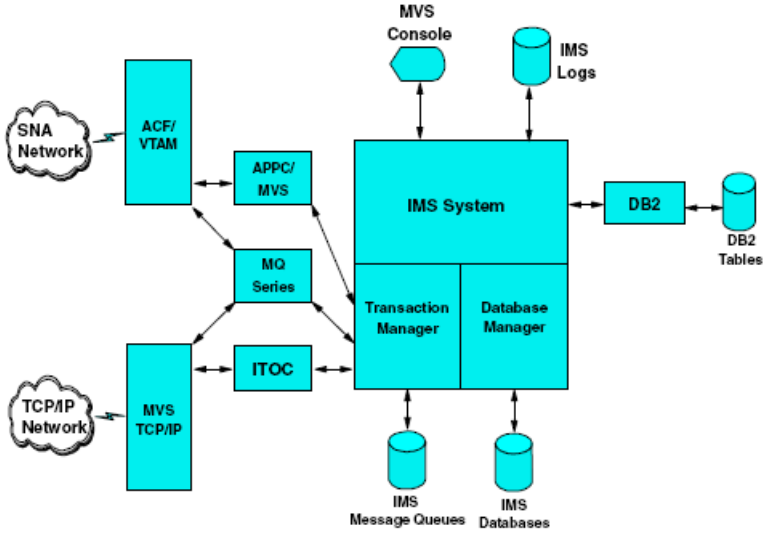
Vaikka IMS on myös tietokannan hallintajärjestelmä, se tunnetaan erityisesti tapahtumankäsittelyjärjestelmästään. IMS:n lisäksi voidaan mainita kaksi muuta keskeistä tapahtumankäsittelyjärjestelmää. Näistä toinen on IBM:n CICS ja toinen Bea Systemsin Tuxedo. Tuxedo oli alunperin Novellin ohjelmisto, jonka Bea Systems osti. Bea Systems on nykyään Oraclen omistuksessa.

IMS-järjestelmän keskeisimmät kaksi komponenttia ovat tapahtumankäsittelyjärjestelmä ja tietokannan hallintajärjestelmä. Näiden lisäksi järjestelmän kolmantena keskeisenä osana on niin sanottu järjestelmäkomponentti, joka tuottaa järjestelmätason palvelut kahdelle ensin mainitulle osa-alueelle. Näitä järjestelmäpalveluita ovat mm. kahden pääjärjestelmän käynnistys- ja sammutustoiminnot, tietoturvapalvelut ja sovellusten hallintapalvelut. Kahta keskeistä osajärjestelmää voidaan tarvittaessa käyttää myös toisistaan riippumatta. [LHHN00]

IMS on kehitetty erityisesti korkean käytettävyyden ja suurten tapahtumamäärien käsittelykykyä vaativien sovellusten ajoympäristöksi. Sen perustoiminnallisuus on pitkästä elinkaaresta huolimatta säilynyt vuosien varrella ennallaan. Muutoksia on tapahtunut lähinnä liitettävyydessä ja rajapinnoissa, joita tapahtumankäsittelyjärjestelmä tarjoaa ulospäin. Uusimmissa versioissa on mukaan tullut useita ominaisuuksia ja teknologioita, jotka tukevat palvelukeskeisen arkkitehtuurin vaatimuksia. Tapahtumankäsittelyjärjestelmä pystyy käyttämään IMS-tietokantojen lisäksi DB2 tietokannanhallintajärjestelmän tietokantoja.

IMS-järjestelmän komponentit ja keskeisimmät ulkoiset yhteydet on esitelty kuvassa 5. *Tapahtumankäsittelyjärjestelmästä (IMS Transaction Manager, IMS TM)* käytetään myös termiä tietoliikenteen hallintajärjestelmä (IMS Data Communication Manager, IMS DC). Tapahtumankäsittelyjärjestelmä on viestipohjainen järjestelmä ja se tarjoaa verkossa oleville käyttäjille ja muille ohjelmille pää-

syn IMS-ohjelmiin. IMS TM tukee ja on aikaisemmin tukenut ainoastaan IBM:n Systems Network Architecture (SNA) verkkoyhteyttä, joka on tarjottu VTAM-verkkokomponentin kautta. Nykyään tuettuna on myös TCP/IP-protokolla, mikä helpottaa ulkoisten järjestelmien integrointia ja IMS-ohjelmien käyttöä standardeilla rajapinnoilla. TCP/IP-yhteys vastaanotetaan IMS TCP/IP OTMA Connector (ITOC) -liittymän kautta. Tämän lisäksi viestejä vastaanotetaan MQ-ohjelmistolta MQ IMS Bridge -ohjelmiston kautta. [LHHN00] Uudemmisssa IMS-järjestelmissä kuvassa 5 vielä näkyvä ITOC on korvautunut IMS Connect -ohjelmistolla, joka hoitaa TCP/IP-pohjaiset yhteydet [JLG+02]



Kuva 5: IMS-järjestelmäkuva ja ulkoiset rajapinnat [LHHN00].

Tietokannan hallintajärjestelmä (IMS Database Manager eli IMS DB) hoitaa tiedon hallintaan ja tiedon talletukseen liittyvät operaatiot. Tiedon hallinnalla taataan mm. tiedon eheys ja mahdollistetaan sen yhtäaikainen käyttö usean käyttäjän toimesta. Tiedon talletukseen IMS DB pystyy käyttämään seuraavia tietokantoja [LHHN00]:

- *IMS Full Function Database* on IMS:n standarditietokanta. Se on ns. täysin toteutettu tietokanta, josta toisinaan käytetään vanhaa nimeä DL/1-tietokanta (Data Language 1). IMS-tietokantojen talletusrakenne on hierarkkinen. Tiedon haku on mahdollista tietoalkio kerrallaan tai peräkkäisesti sekä muissa ennakkoon suunnitelluissa järjestyksissä. Tietokantaan liittyy fyysinen kokorajoitus, joka on joko 4 gigatavua tai 8 gigatavua riippuen

käyttöjärjestelmän talletusmekanismista.

- *IMS Data Entry Database (DEDB)*, josta käytetään myös nimeä Fast Path (Data Entry) Database. Se soveltuu suuriin tietokantajärjestelmiin tai tilanteisiin, joissa tarvitaan hyvää suorituskykyä ja datan saatavuutta tai edullista käyttökustannusta.

- *IMS Main Storage Database (MSDB)* oli alkuperäinen DEDB-tietokantojen hakutapa (Access Method), mutta se on korvaantunut nykyisin Virtual Storage Option (VSO) -hakutavalla.

- *IMS High Availability Large Database (HALDB)* on laajennus Full Function -tietokantaan. HALDB tarjoaa paremman saatavuuden (Availability) sekä mahdollisuuden käyttää suurempia tietokantoja kuin alkuperäinen Full Function -tietokanta. [JAK+04]

IMS-ohjelmat voivat käyttää talletusrakenteena myös IBM Database 2 eli DB2-tietokantajärjestelmää. Tällöin tietokannan hallinta tapahtuu DB2-tietokantapalvelimen tiedonhallintajärjestelmän toimesta, eikä IMS DB ole osallisena tiedon talletuksessa. DB2 on IBM:n relaatiopohjainen tietokantatoteutus ja se soveltuu hyvin joustavuutta vaativiin talletusrakenteisiin ja erityisesti sellaisiin käyttötilanteisiin, joissa tulevia hakumekanismeja ei ennakkoon vielä tunneta varmuudella. Prosessointikustannus on merkittävästi suurempi kuin IMS-tietokannoissa. [LHHN00]

3.4 IMS-järjestelmän integrointi

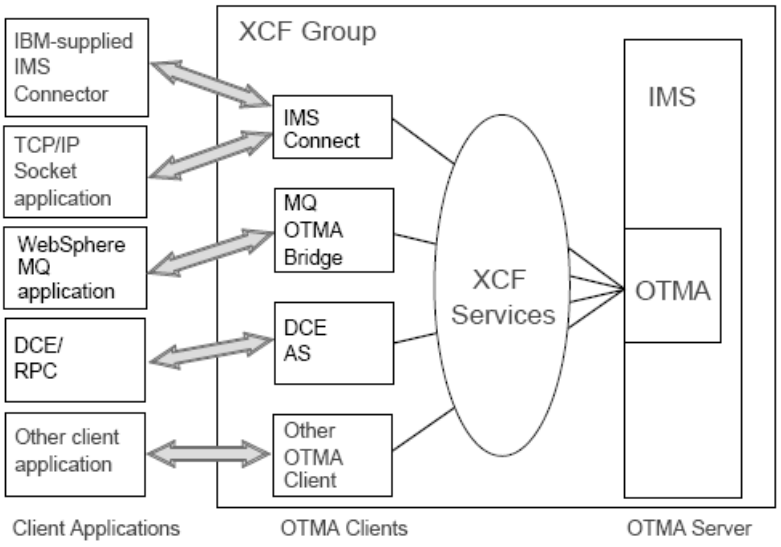
IMS-järjestelmä jakaantuu kahteen loogiseen osaan, tapahtumankäsittelyjärjestelmään ja tietokannanhallintajärjestelmään. Ohjelmat voivat käsitellä IMS-järjestelmässä olevaa tietoa näiden osien kautta, joten järjestelmän integrointi voidaan myös jakaa kahteen pääkategoriaan.

3.4.1 Tapahtumankäsittelyjärjestelmän integrointi

IMS-tapahtumankäsittelyjärjestelmään keskustietokoneen ulkopuolelta tulevat tapahtumapyynnöt tulevat verkkoyhteyskomponentin ja *Open Transaction Ma-*

nager Access (OTMA) -rajapinnan kautta. OTMA on tapahtumapohjainen yhteydetön asiakas-palvelin -protokolla, joka tarjoaa rajapinnan viestien ja tiedon lähettämiseen ja vastaanottamiseen IMS-järjestelmästä. Se on toteutettu z/OS-käyttöjärjestelmän MVS Cross System Coupling Facility (XCF) toteutusta tukevaksi, joten skaalatuvuus klusteroidussa järjestelmässä (Parallex Sysplex) on tuettu suoraan käyttöjärjestelmän tasolta. [JLG+02]

OTMA on suunniteltu suorituskykyiseksi protokollaksi ja sen tuki eri verkkoprotokollille on toteutettu useiden eri OTMA-asiakasohjelmien (client) avulla. OTMA-yhteydet hoidetaan sisäisen väylän kautta, jolloin niiden suorituskyky on lähes verrattavissa muistissa tapahtuvaan käsittelyyn. Jäntti & al. [JLG+02] esittelevät kuvassa 6 järjestelmään kuuluvat komponentit ja esimerkkejä erilaisista OTMA-asiakasohjelmista. OTMA tarjoaa myös ohjelmointirajapinnan Callable Interface (OTMA C/I), jonka avulla voidaan itse toteuttaa asiakasohjelmisto. [JLG+02]



Kuva 6: OTMA-järjestelmän komponentit ja asiakasohjelmia [JLG+02].

IMS Connect on erillinen ohjelmisto, jonka avulla TCP/IP-verkon asiakasohjelmat voivat muodostaa yhteyden IMS-järjestelmään. Se mahdollistaa useiden yhtäaikaisten asiakasohjelmien käyttää useita eri IMS-sovelluksia samanaikaisesti. Yhteyden voi muodostaa mikä tahansa ohjelmisto TCP/IP-protokollalla socket-pohjaista yhteyttä käyttäen. Tämän lisäksi voidaan käyttää IBM:n toteuttamaa IMS Connector for Java -ohjelmaa, joka myös kommunikoi IMS Connectin kanssa TCP/IP-socketilla. IMS Connector for Java -ohjelmistosta voidaan käyttää myös

lokaalia yhteydenmuodostustapaa, mikä ei ole TCP/IP-pohjainen. Tämä yhteys on mahdollinen vain, mikäli sitä suoritetaan z/OS-käyttöjärjestelmässä käynnissä olevasta prosessista (esimerkiksi WebSphere Application Server for z/OS). [JLG+02]. IMS Connect korvaa entisen, kuvassa 5 näkyvän IMS TCP/IP OTMA Connection (IMS TOC) eli ITOC-palvelun. ITOC oli aikaisemmin pohjana useille yhteyspalveluille, kuten esimerkiksi IMS e-Business Connectors, IMS TOC Connection Connector for Java sekä IMS Web. [LJH+99].

3.4.2 Tietokannanhallintajärjestelmän integrointi

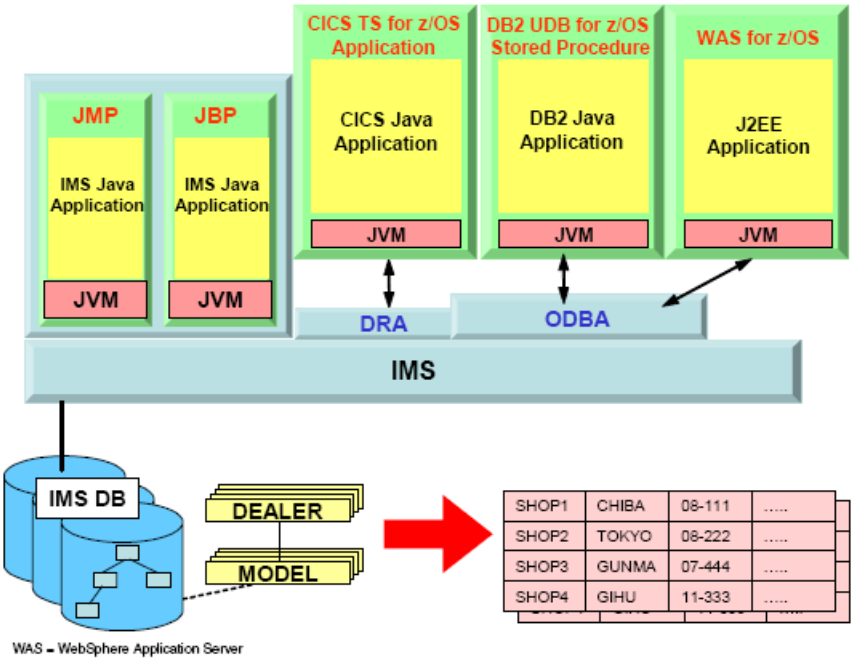
IMS-tietokannanhallintajärjestelmän tietokantojen käyttö voidaan tehdä *IMS Open Database Access (ODBA)* -kutsurajapinnan kautta. Rajapintaa voi käyttää mikä tahansa z/OS-järjestelmän sovellus, joka käyttää myös Recovery Resource Services (RRS) -palvelua. Rajapinnan kautta on käytettvissä sekä Full Function -tietokannat että Data Entry -tietokannat. [JLG+02]

ODBA käyttää *IMS Database Resource Adapter (DRA)* -palvelua yhteyden muodostamiseksi tietokantaan. IMS-alijärjestelmien tiedot (mm. yhteydenmuodostamistapa ja säikeiden määrä) on kirjattu DRA-käynnistystauluun, josta ne luetaan järjestelmän käynnistyessä. ODBA-kutsurajapinnan modulit hoitavat järjestelmään liittyneitä säikeitä sekä huolehtivat tietyistä järjestelmäpalveluista. Ne tarjoavat asiakassovelluksille myös DL/1-tietokantakutsujen rajapinnat. [JLG+02]

ODBA-yhteyttä käyttävät sovellukset käyttävät RSS-palvelua tietokannan transaktioiden hallintaan. DRA-palvelu puolestaan tarjoaa yhteyksien lisäksi monisäikeisyyden yhtäaikaisuuden hallinnan eri sovellusten välillä. [JiPS+06]

IMS-järjestelmän Java-ohjelmat voivat käyttää IMS-tietokantoja JDBC-yhteyden kautta. Yhteyden käyttö on vaatii sellaisten IMS-alueiden (Region) käyttöä, joilla Java-virtuaalikoneen tuki on käytössä. Tällaisia alueita löytyy kaksi: Java Message Processing (JMP) viestipohjaisten tapahtumien käyttöä varten ja Java Batch Processing (JBP) eräajopohjaista käyttöä varten. Kummastakin ympäristöstä on mahdollista käsitellä IMS-viestijonoja, IMS-tietokantoja sekä z/OS-järjestelmän DB2-tietokantoja [JiPS+06]. IMS JDBC -yhteyden käyttö mahdollistaa sen, että Java-ohjelmista voidaan käsitellä IMS:in hierarkisia tietokantoja SQL-kielen avulla. IBM:n toteuttamaan Java-kieliseen JDBC-ohjelmistopakettiin on toteutettu

osajoukko JDBC/SQL -määrittelystä. Talletusrakenteiden välisten erojen käsittelyssä käytetään apuna DLIModel Utility-apuohjelmistoa. Suurimpana haasteena muunnoksissa on hierarkisen talletusrakenteen esittäminen relaatiopohjaisena taulurakenteena. [JiPS+06]



Kuva 7: IMS-tietokantajärjestelmän yhteydet [JiPS+06].

IBM-keskustietokoneelle toteutettu versio WebSphere-sovelluspalvelimesta on pystynyt jo pitkään käsittelemään suoraan IMS-tietokantoja. Tämä yhteys on toteutettu sovelluspalvelimella olevan JDBC-rajapinnan kautta. Rajapinta muodostaa SQL-kyselyistä DL/1-kyselyitä ja ne välitetään ODBA-rajapinnan ja DRA:n kautta IMS-tietokannanhallintajärjestelmälle. IMS Remote Database Services (RDS) -palvelu mahdollistaa myös hajautetulla alueella (Unix, Windows) olevien sovelluspalvelinten käyttää IMS-tietokantoja. Tällöin sovelluspalvelimesta on oltava asennettuna instanssi myös z/OS-järjestelmässä, sillä varsinainen tietokannan käsittely tapahtuu suurkoneen sovelluspalvelimen toimesta. Hajautetun alueen sovelluspalvelin voi toteuttaa sovelluslogiikan, mutta IMS-tietokantojen käsittelyyn tarvittavat JDBC-komennot on reititettävä z/OS-järjestelmään suoritettavaksi (tarvitaan lokaali yhteys). [JiPS+06]

4 Palvelukeskeisen arkkitehtuurin toteuttaminen

Palvelukeskeisen arkkitehtuurin toteutus uusiin, vasta rakennettaviin järjestelmiin ja ohjelmistoihin on ainakin näennäisesti yksinkertainen asia. Suunnittelussa ja toteutuksessa on tärkeää noudattaa oikeita suunnitteluperiaatteita sekä laatia oma sovellusarkkitehtuuri, jota noudatetaan [Erl05]. Vanhojen järjestelmien muuttamiseen liittyviä ongelmia ei tällöin jouduta kohtaamaan.

Olemassa olevien vanhojen järjestelmien sopeuttaminen palveluarkkitehtuuriin on vaikeampaa kuin uusien järjestelmien toteuttaminen. Mitään yleispätevää, kaikkiin tilanteisiin sopivaa, toimintamallia ei ole olemassa. Mikäli arkkitehtuuriin liittyy useita erilaisia eri teknologioilla toteutettuja järjestelmiä, tilanne on vielä hankalampi. Tällöin täytyy löytää sopivat menetelmät kunkin järjestelmän mukauttamiseksi sekä yhteinen integrointi- ja palvelukerros näiden järjestelmien kytkemiseksi samaan palvelualustaan.

Vanhoista, olemassa olevista järjestelmistä käytetään tyypillisesti termiä perinnejärjestelmä. Perinnejärjestelmä tarkoittaa yleensä iältään kohtuullisen vanhaa, suurta ja monimutkaista järjestelmää tai järjestelmäkokonaisuutta. Usein kyseessä on myös järjestelmä, joka on vaatinut korkean käytettävyyden (High Availability) ja suuren läpimenon (Throughput) sen aikaisilla laitteistoilla. Järjestelmäkokonaisuuden sisällä eri järjestelmien toiminnallisuus ja tieto on usein toisistaan irrallaan (siiloutunut). Termiä käytetään myös kun puhutaan tuotannossa olevasta sovelluksesta tai sovelluksesta, joka ei ole loppukäyttäjän sovellus [Hes05]. Termin käyttöä ei ole täysin vakiintunutta ja sillä on monia muitakin merkityksiä. Tyypillisesti termi kuitenkin yhdistetään suurkone-ympäristöihin sekä perinnejärjestelmiin.

Perinnejärjestelmä-termiä käytetään usein hieman vaihtelevasti eri ikäisistä eri teknologialla tehdyistä järjestelmistä. Sneed [Sne06] esittää perinnejärjestelmien luokitteluun ohjelmointikieliin pohjautuvaa mallia. Hänen mukaansa tyypillisesti 70-luvun ohjelmointikielillä, kuten Fortran, Cobol ja C, toteutetut järjestelmät voidaan lukea perinnejärjestelmiksi. Samoin monet 80-luvun neljännen sukupolven (4GL) kielillä (Ideal, Oracle Frames) ja jopa osa 90-luvun olio-pohjaisilla kielillä, kuten C++ ja Smalltalk voidaan lukea perinnejärjestelmiksi. Hän lukee jopa varhaisimmat Java-kielellä tehdyt sovellukset samaan kategoriaan.

Usein perinnejärjestelmä-termi yhdistyy tilanteeseen, missä ao. järjestelmä ei ole enää aktiivisen kehitystyön kohteena ja sen toteutusteknologia saattaa olla tämän lisäksi vanhentunutta. Toisinaan taas järjestelmään ei ole mahdollista enää saada aikaiseksi edes pieniä muutoksia.

4.1 Järjestelmien integrointi

Perinnejärjestelmiä voidaan mukauttaa palvelukeskeiseen arkkitehtuuriin sopiviksi joko muuttamalla järjestelmiä siten, että ne tuottavat palveluita SOA:n mukaisesti tai toteuttamalla erillinen palvelukerros järjestelmän oheen. Kummassakin tapauksessa järjestelmän sovellukset täytyy liittää muihin järjestelmiin. Tätä järjestelmien liittämistä toisiinsa kutsutaan integroinniksi. Perinnejärjestelmien integrointiin voidaan käyttää Iocolan [Ioc07] mukaan seuraavia erilaisia malleja:

- Tiedonsiirto (File Transfer Service Integration): Perinteisin perinnejärjestelmän integrointitapa, jossa tietoa siirretään kahden järjestelmän välillä pisteestä toiseen. SOA:n mukaisesti tässä mallissa järjestelmät ovat toisistaan irralliset ja tieto siirretään esimerkiksi palveluväylän kautta.
- Asynkroninen integraatio (Asynchronous Integration): Tässä mallissa järjestelmien osat ovat täysin erillään toisistaan. Tietoa tuottava järjestelmä toimittaa informaation esimerkiksi palveluväylälle ja voi sen jälkeen välittömästi jatkaa omaa suoritustaan. Palveluväylä on vastuussa tiedon toimittamisesta oikeaan kohteeseen. Viestipohjainen (Message Oriented) palveluväylä sopii tähän tarkoitukseen hyvin.
- Synkroninen integraatio (Synchronous Integration): Palvelua käyttävä järjestelmän osapuoli jää odottamaan vastausta suorittamalleen palvelupyynnölle. Käytetään tyypillisesti reaaliaikapalveluissa hajautettujen järjestelmien osana.

Swithinbank & al. [SBC+07] jakavat integrointitavat kahteen eri pääkategoriaan, prosessikeskeiseen (Process-focused Integration) ja tietokeskeiseen integraatioon (Data-focused Integration). Sama jako noudattelee myös IMS-järjestelmän keskeisiä integrointitapoja, johtuen siitä, että IMS jakaantuu tapahtumankäsittelyjärjestelmäksi ja tietokannanhallintajärjestelmäksi.

Prosessikeskeinen integraatio tunnetaan entuudestaan nimellä yrityssovellusten integraatio (EAI). Sille on keskeistä sovellusten välinen prosessiorientoitunut integraatio, jossa tavoitteena on tuottaa liiketoiminnallisesti kattavat tietoalkiot, mahdollisesti keräämällä tarvittava tieto useasta eri järjestelmästä. Prosessipohjainen integraatio on tyypiltään viestipohjaista ja tapahtumakeskeistä sovellusten välistä integraatiota, joka voidaan toteuttaa pisteestä pisteeseen tyyppisesti tai keskitetyn palveluväylän avulla. EAI:stä saatavat tärkeimmät hyödyt ovat Swithinbank & al. [SBC+07] mukaan:

- Integraatiossa on mukana sekä tieto että prosessit.
- Liiketoimintaprosessien ja -tiedon uudelleenkäyttö sekä jakaminen.
- Helpottaa sovellusten yhdistämistä, koska integraatio toteutetaan sovellusten yksityiskohtaisen toteutuskerroksen yläpuolella.

Tietokeskeinen integraatio voidaan jakaa kahteen eri alakategoriaan: tiedon keräys, muunnos ja lataus tyyppinen integraatio (Extract, Transform and Load, ETL) ja yritysinformaation integraatio (Enterprise Information Integration, EII). ETL integraatio koostuu kolmesta eri vaiheesta. Ensimmäisessä vaiheessa tieto kerätään useista eri lähteistä (Extract), toisessa vaiheessa se muunnetaan liiketoimintaa tukevaan muotoon (Transform) ja lopuksi ladataan kohteena olevaan tietojärjestelmään (Load). ETL:ssä käytetään usein tiedostoja tai relaatiotietokantoja, mutta myös IMS-järjestelmän hierarkkiset tietokannat soveltuvat siihen hyvin. EII puolestaan tarjoaa keskitetyn ja optimoidun tiedonsaanti- ja muunnoskerroksen sovelluksille. Se tarjoaa reaaliaikaiset luku- ja kirjoitusoperaatiot tietoon, suorittaa tietomuunnokset sekä huolehtii tiedon eheydestä ja saatavilla olosta. [SBC+07]

4.2 Palveluiden valintastrategiat

Palvelukeskeistä arkkitehtuuria suunniteltaessa on määriteltävä ensin palvelut. Mikäli käytössä on useita eri järjestelmiä, tehdään tämä määrittely ensin kokonaisuuden osalta. Sen jälkeen voidaan suunnitella kunkin järjestelmän palveluita itsenäisesti. Palveluiden valintaan on käytettävissä erilaisia lähestymistapoja, joita voidaan soveltaa.

Palveluita toteutettaessa on käytettävissä useita erilaisia vaihtoehtoja. Valmiita palveluita voidaan ostaa kaupallisilta ohjelmistojen tarjoajilta tai käyttää mahdollisesti avoimen lähdekoodin ratkaisuja. Tämän lisäksi palvelut voidaan kehittää itse tai käyttää olemassa olevia järjestelmiä niiden tuottamiseksi [Sne06]. Valmiiden palveluiden hankkimisen mahdollisuus on suuresti riippuvainen markkinoilla olevasta tarjonnasta, mikä taas puolestaan riippuu toimialasta. Palveluiden kehittäminen itse yrityksen IT-hankkeena on mahdollista, mutta vaatii yleensä suuria työpanoksia. Vaihtoehtona on hyödyntää olemassa olevia järjestelmiä. Niiden hyödyntämiseen palvelukeskeisen arkkitehtuurin keinoin on useita eri tyyppisiä lähestymistapoja.

Palvelukeskeisen arkkitehtuurin rakentamisvaiheessa määritellään aluksi toteutettavat palvelut. Valintavaiheessa on oltava tarkkana, sillä tämän tyyppisiin hankkeisiin kohdistuu usein myös kriittistä tarkastelua projektin ulkopuolelta. Valittujen palveluiden tulisi olla sellaisia, että niiden hyödyntämisellä saadaan mahdollisimman suuria tai laajalti näkyviä liiketoiminnallisia hyötyjä. Hyötyjen tuleekin alkuvaiheessa olla ensisijaisesti liiketoiminnallisia [Sne06]. Kustannukset ovat myös merkittäviä ensimmäisessä vaiheessa. Tämän vuoksi palveluiden valinnassa tulee tehdä valintoja, jotka ovat myös kustannustehokkaita toteuttaa. Palveluiden toteutustapa vaikuttaa myös kustannuksiin, joten sitä täytyy tarkastella erikseen. Vaikka kustannuksiin kiinnitetäänkin paljon huomiota, on palvelukeskeisen arkkitehtuurin käyttöönottoprojekti tyypillisesti kalliimpi kuin perinteinen kehitysprojekti [Lie07].

Valittaessa palveluita uuteen palvelukeskeiseen arkkitehtuuriin voidaan mahdollisesti valittaville ohjelmistokomponenteille asettaa valintakriteereja [Ioc07] siten, että palveluiden täytyy

- tuottaa kokonainen liiketoiminnallinen toiminto
- olla julkaistavissa standardilla määrittelykielellä ja järjestelmäriippumattoman rajapinnan kautta
- integroitua tai liittyä nykyiseen palvelujärjestelmään

4.3 Palveluiden toteutusmallit

Valittujen palveluiden toteutuksessa voidaan käyttää erilaisia toteutusmalleja. Nämä samat toteutusmallit ovat muistakin yhteyksistä tuttuja etenemistapoja erilaisiin toteutustehtäviin ja niitä käytetään usein myös integrointiratkaisuja toteutettaessa. Al Belushi ja Baghdadi [BB07] esittelevät kaksi erilaista mallia: ylhäältä-alas (Top-Down) ja alhaalta-ylös (Bottom-Up) -mallit. Seubert ja Ralsch [SR06] mainitsevat näiden kahden lisäksi yhdistämismallin (Meet-in-the-Middle), joka on ikään kuin kahden edellisen mallin välimuoto.

Ylhäältä alas -mallissa palveluita mallinnetaan palvelun liiketoimintalogiikan näkökulmasta tarkastellen. Tällöin painopiste on palvelun suunnittelussa ja lopputuloksena päästään lähemmäksi palvelun aitoa tarvetta. Tämä malli on myös lähimpänä palvelukeskeisen arkkitehtuurin periaatteita, mutta toisaalta siinä saatetaan ajautua tarpeettoman etäälle olemassa olevista tietojärjestelmistä ja niiden toteutuksesta. Toteutusmallin ja olemassa olevan järjestelmän välinen ero lisää palveluarkkitehtuuriin kuluvaa työmäärää. [BB07, SR06]

Alhaalta ylös -mallissa palvelukeskeisen arkkitehtuurin palvelut suunnitellaan olemassa olevien järjestelmien toiminnallisuuden pohjalta. Näissä järjestelmissä olevat sovellusten toteutukset analysoidaan ja analyysin pohjalta valituista osuuksista toteutetaan palvelukeskeisen arkkitehtuurin uudet palvelut. Alhaalta-ylös -mallissa muutokset olemassa oleviin järjestelmiin pysyvät minimissä. Toteutuksen kelvollisuus tässä vaihtoehdossa on suuresti kiinni vanhan järjestelmän toteutuksesta. Joissakin tapauksissa olemassa olevat tapahtumat ovat riittävän atomisia ja rajapinnoiltaan yleisiä, että niistä voidaan muodostaa hyviä palveluita. Keskustietokoneelle laadituista ohjelmista osa sopii hyvin tämäntyyppiseen malliin. Jos taas olemassa olevat palvelut on sopeutettu jo suoraan esimerkiksi monimutkaisen käyttöliittymän totettamiseen, saattaa tämän mallin noudattaminen olla hankalaa tai ainakin päädytään huonoihin palvelurajapinnan palveluihin. [BB07, SR06]

Kolmantena mallina on *yhdistämismalli*. Se on kahden edellä mainitun välinen hybridi, jossa hyödynnetään molempien mallien parhaita puolia. Olemassa olevista sovelluksista valitaan palveluiksi sopivimmat ja suunnitellaan uuden palvelun palvelurajapinnat. Näiden pohjalta suunnitellaan varsinaisen palvelun toteutus sekä sovelluksen ja uuden palvelun välinen tietojen ja kutsuparametrien muunnos. [SR06]

Tuntuisi ehkä houkuttelevalta noudattaa aina yhdistämismallia, koska se tuntuu toimivan useimmissa tapauksissa ainakin kohtuullisen hyvin. Näin ei kuitenkaan ole, vaan tarkastelu kannattaa tehdä aina tilannekohtaisesti sillä tietyissä tilanteissa ylhäältä-alas tai alhaalta-ylös -mallilla saadaan paras lopputulos tai ainakin työmäärä on kaikkein pienin. [SR06]

4.4 Perinnejärjestelmien muutosstrategiat

Perinnejärjestelmien toteuttaminen palvelukeskeisen arkkitehtuurin mukaiseksi on yritykselle muutosprosessi, josta Seubert ja Ralsch [SR06] käyttävät termiä yrityksen muutosprosessi (Enterprise Transformation). Perinnejärjestelmien olemassaolo ohjaisi helposti alhaalta-ylös -mallin mukaiseen palveluiden toteuttamiseen. Toisinaan näiden järjestelmien ylläpito ja jatkokehitys on kuitenkin hankalaa, johtuen niissä käytetyn teknologian tai ohjelmointikielen vanhentuneisuudesta. Tällöin yrityksen on pohdittava järjestelmien merkitystä pitkällä aikavälillä ja tehtävä päätöksiä siitä, miten niihin suhtaudutaan pidemmän aikavälin strategiassa. Perinnejärjestelmien elinkaareen liittyen voidaan käyttää kolmea erilaista muutosstrategiaa: kehittäminen (Improve), mukauttaminen (Adapt) ja uudistaminen (Innovate). Tyypillisessä SOA-toteutuksessa käytetään useampaa kuin yhtä näistä strategioista. [SR06]

Kehittämisstrategia pyrkii toteuttamaan vanhojen järjestelmien palveluistamisen käyttöliittymätasolla. Tässä strategiassa keskeistä on myös se, että vanhoihin järjestelmiin eikä järjestelmäarkkitehtuuriin tehdä muutoksia, tai jos tehdään niin ne ovat vähäisiä. Vanhan järjestelmän palvelu muunnetaan liiketoimintapalveluksi käyttöliittymän tasolla. Vaihtoehtoisesti mitään tietosisällön muunnosta ei tehdä, vaan ainoastaan tuodaan sovelluksen tiedot käyttöön modernimmalla käyttöliittymällä, kuten portaali tai muu selainkäyttöinen liittymä. Tämä strategia vastaa alhaalta-ylös -toteutusmallia.

Mukauttamisstrategia puolestaan toteuttaa vanhojen järjestelmien integroinnin ja mahdolliset tietomuunnokset keskitetyllä ratkaisulla. Tässä strategiassa valitaan tekninen integrointitapa (tai useita), joilla vanhoihin järjestelmiin tehdään integraatio. Keskitetyssä kerroksessa puolestaan tarjotaan palvelut niitä käyttäville uusille järjestelmille. Usein tällaisessa ratkaisussa on mukana myös keskitetty palveluväylä tai sen kaltaista toiminnallisuutta on muuten toteutettu. Tällä

strategialla saavutetaan vanhojen järjestelmien laajempi käyttöaste uusissa järjestelmissä nimenomaan keskitettyjen integrointipisteiden ansiosta.

Uudistamisstrategiaa noudatettaessa modernisoidaan olemassa olevia järjestelmiä. Tämä strategia on lähimpänä ylhäältä-alas -toteutusmallia. Uudistamisstrategiassa suunnittelu aloitetaan liiketoimintaprosesseista ja sieltä edetään palveluihin. Vasta tämän jälkeen tulee palveluiden toteutus, joten käytännössä tämän strategian mukaisesti edettäessä tehdään eniten nykyjärjestelmiin liittyvää kehitystyötä. Se saattaa jopa johtaa uusien järjestelmien ja palveluiden tekemiseen. Strategia on myös lähimpänä puhdasveristä palvelukeskeisen arkkitehtuurin ideologiaa.

Perinnejärjestelmien muuttamista palvelukeskeiseen arkkitehtuuriin voidaan tarkastella myös eri modernisointitapojen näkökulmasta kuten Erradi & al. [EAK06] tekevät. Heidän esityksensä sisältö on lähellä edellisissä kappaleissa Seubertin ja Ralschin esittämiä kolmea eri strategiaa, he tosin jakavat menettelytavat vain kahteen eri kategoriaan. Näiden kahden modernisointitavan välillä merkittävin ero on siinä, minkä verran perinnejärjestelmän kehittämiseen panostetaan. *Perinnejärjestelmien kehittämiseen* tähtäävä modernisointitapa (Legacy Transformation) panostaa paljon olemassa olevien järjestelmien kehittämiseen ja jopa niiden osittaiseen uudelleen rakentamiseen (Invasive Reengineering). Tavoitteena on järjestelmien elvyttäminen ja sopeuttaminen palvelukeskeiseen arkkitehtuuriin. Tämä työ vaatii järjestelmien syvällisen tarkastelun ja muutostöiden tekemisen suoritettuun analyysiin pohjautuen. Toinen modernisointitapa on heidän mukaansa *perinnejärjestelmien muuttaminen palveluiksi integroinnin avulla* (Legacy Integration and Service Enablement). Tässä tavassa perinnejärjestelmiä ei kehitetä, vaan tavoitteena on pidentää niiden käyttöikää mahdollistamalla järjestelmien uusi tai laajempi käyttö. Tämä toteutetaan tekemällä palvelurajapinta perinnejärjestelmän ulkopuolelle. Palvelut tarjotaan palvelurajapinnassa standerdeilla rajapinnoilla ja niiden toteutukseen käytetään erilaisia integrointitapoja. Integrointi voi tapahtua suoraan perinnejärjestelmien tietokantoihin tai käyttämällä niiden ohjelmarajapintoja. Ulkopuolinen palvelu on vastuussa liiketoimintapalvelun toteuttamiseen liittyvistä yksityiskohdista, kuten tietomuunnoksista ja parametrien muokkaamisesta.

Näiden kahden strategian välinen merkittävin ero liittyy perinnejärjestelmän odotettavissa olevaan elinaikaan. Jälkimmäisessä mallissa tämä jäänee lyhyemmäk-

si, koska se on selkeästi lyhyen tähtäimen malli, jossa ratkaistaan ainoastaan integrointiin liittyviä ongelmia. Tämä voi toki olla myös yrityksen valitun kokonaisstrategian mukaista ja jopa käytännön pakko esimerkiksi perinnejärjestelmän jatkokehitykseen liittyvien resurssiongelmien takia. Ulkoinen palvelurajapinta mahdollistaa myös myöhemmin perinnejärjestelmän korvaamisen jollain toisella järjestelmällä tai ainakin se helpottaa sitä.

4.5 Palveluiden toteutustavat

Olemassa olevia järjestelmiä hyödynnetään usein, koska niiden uudelleenkirjoittaminen olisi liian suuri työ. On arvioitu, että jopa 70% maailmassa olevista liiketoimintasovelluksista on kirjoitettu Cobolilla [Ioc07]. Koodin arvo on täten liian suuri hukattavaksi. Koodin uudelleen käyttö on myös nopea tapa lähteä liikkeelle SOA:n kanssa, koska tällöin on mahdollista saavuttaa suuria hyötyjä myös suhteellisen pienellä panostuksella. Vanhojen järjestelmien modernisoinnista saavutettujen hyötyjen lisäksi liitttyy niihin myös epäonnistumisen riski. Yksi suurimmista ongelmista on, että vanhojen järjestelmien tarjoamien palveluiden hienojakoisuus ei välttämättä ole sopiva uusille liiketoimintatarpeille.

Sneed [Sne06] esittää järjestelmien sopeuttamiseksi palvelukeskeiseen arkkitehtuuriin sopivaksi kolme vaihetta: 1) olemassa olevan koodin hyödyntäminen (Salvaging the Legacy Code), 2) koodin kääriminen palveluksi (Wrapping the Salvaged Code) ja 3) palvelun käyttöönotto (Making the Code Available as Web Service). Koodin hyödyntäminen tarkoittaa, että olemassa olevasta koodista etsitään palveluihin sopivia osuuksia. Koodista arvioidaan sen sopivuus liiketoimintasääntöihin nähden, joten keskeistä tarkastelussa on koodin laadun lisäksi sen tietosisältö ja kutsurajapinnat. Kun palveluihin sopivat koodiosuudet on löydetty, niistä voidaan muodostaa tarvittavat palvelut. Palvelun rajapinnat tarjotaan muiden järjestelmien käyttöön jollakin standardilla tavalla, kuten esimerkiksi WSDL. Tässä välissä tarvitaan yleensä tietosisällön mukauttamista (vähemmän tai enemmän parametreja), tietotyyppien muunnoksia ja mahdollisesti erityyppisiä tarkistuksia. Palveluiden käyttöönottovaiheessa palvelut tarjotaan muiden järjestelmien käytettäväksi esimerkiksi palveluväylän kautta tai ne voidaan liittää osaksi palveluprosesseja.

Perinnejärjestelmän toiminnallisuuden mukauttamisesta palvelukeskeiseen arkki-

tehtuuriin sopivaksi palveluksi käytetään usein termiä kääriminen (Wrapping). Al Belushi ja Baghdadi [BB07] käyttävät termejä adapteri (Adapter), välityspalvelin (Proxy) ja kääre (Wrapper) kaikkia hieman eri tarkoituksessa. Heidän mukaansa näiden välisisinä eroina on se, että adapteri ja välityspalvelin mukautetaan palvelun käyttäjälle etukäteen ja kääre taas on täysin riippumaton laitteistosta, ohjelmointikielestä ja verkkoprotokollista. Papazoglou ja van den Heuvel [PvdH07] puolestaan pitävät termejä kääre ja adapteri käytännössä lähes synomyymeinä ja käyttävät termiä yhdistin (Connector) välityspalvelin-termin sijaan.

Al Belushi ja Baghdadi [BB07] esittelevät palveluiden käärimiseksi kolme eri lähestymistapaa. *Istuntopohjainen* (Session Based) tapa sopii tilanteisiin, missä sovelluksen liiketoimintalogiikka ja käyttöliittymä ovat tiiviisti yhteen kytkettyjä. *Tapahtumapohjainen* (Transaction Based) tapa puolestaan sopii järjestelmiin, joissa tiedon käsittely, liiketoimintalogiikka ja esityskerros ovat toisistaan irrallaan. *Tietopohjainen* (Data Based) malli taas sopii tilanteisiin, joissa käsitellään lähinnä tietokantojen tietoa, mutta olemassa olevaa liiketoimintalogiikkaa ei ole.

5 Palvelukeskeisen arkkitehtuurin toteutus IMS-järjestelmälle

Palvelukeskeinen arkkitehtuuri yritystasolla koostuu yleensä useista erilaisista tietojärjestelmistä. Palvelukeskeisyydestä saavutettavat edut ovat suurimmillaan, kun integroitavia järjestelmiä on useita ja niiden teknologia sekä toteutusarkkitehtuurit ovat erilaisia. Palvelukeskeistä arkkitehtuuria toteutettaessa määritellään ensin eri järjestelmien tuottamat palvelut. Tämän jälkeen laaditaan oma yhteinen arkkitehtuuri. Tämä arkkitehtuuri kattaa muun muuassa tietojen esitysmallit, palveluiden kuvausmallit sekä muut yhteisesti eri järjestelmissä tarvittavat tekniset palvelut, kuten tietoturva ja lokitus [Jos07]. Tämän jälkeen on kunkin tietojärjestelmän osalta suunniteltava sen tarjoamat palvelut sekä kuinka ne voidaan muodostaa.

Palvelukeskeisen arkkitehtuurin toteutus IBM-suurkoneen IMS-järjestelmälle aloitetaan analysoimalla sen nykytila sekä miettimällä IMS-järjestelmälle muutosstrategia. Tämän muutosstrategian ja tavoitetilan asettamisen jälkeen voidaan valita toteutusmallit ja niihin sopivat sovellusten ja tietokantojen integrointitavat.

5.1 Palvelukeskeisen arkkitehtuurin toteutusprosessi

Palvelukeskeiseen arkkitehtuuriin siirtyminen on pitkäkestoinen prosessi. Prosessin alkuvaiheessa on ensin määriteltävä liiketoiminnalliset tavoitteet ja se kuinka tavoitteisiin pyritään. Tämän jälkeen analysoidaan tietojärjestelmien nykytila tavoitteiden kuvaamalla tasolla.

Tietojärjestelmien nykytilan analyysissa analysoidaan yrityksen koko tietojärjestelmä. Mikäli yrityksellä on suurkone-järjestelmiä, liittyy yrityksen tietojärjestelmiin Kooijmans & al. [KCL+07] mukaan yleensä seuraavia ominaisuuksia:

- Tietojärjestelmien kokonaisuus on kirjava myös tiedon käsittelytavoiltaan.
- Yrityksellä on sovelluksia sekä z/OS-järjestelmässä että hajautetuissa järjestelmissä. Nämä sovellukset on rakennettu eri aikoina ja niiden asiakasohjelmat ovat keskenään erilaisia ja ne käyttävät erilaisia kommunikointiprotokollia.

- Sovellukset kommunikoivat keskenään z/OS-järjestelmän sisällä ja myös eri järjestelmien välillä. Kommunikointitapoja on useita, mutta asiakas-palvelin -tyyppinen toteutus on yleisin.
- Sovellukset on suunniteltu ilman tietoisuutta palvelukonseptista (palvelun tarjoaja ja palvelun käyttäjä). Ne eivät juurikaan käytä standardeja vaan pohjautuvat lähinnä toteutusaikanaan käytössä olleisiin teknologioihin.
- Useimmilla sovelluksilla ei ole selkeää eroa käyttöliittymän, liiketoimintalogiikan ja tietokerroksen kesken.
- Reaaliaika integrointi sovellusten välillä on vähäistä. Tietoa vaihdetaan sovellusten välillä tyypillisesti eräajoina tai ETL-mallin mukaisesti.
- Sovellukset eivät juurikaan julkaise toiminnallisuuttaan palveluina (muutamaan poikkeusta lukuunottamatta).

Tietojärjestelmien nykytilan kartoituksen pohjalta voidaan muodostaa käsitys yrityksen nykytilan suhteesta palvelukeskeisen arkkitehtuurin perusperiaatteisiin. Kuten aikaisemmin todettiin, on yrityksen järjestelmien välillä eroja niiden toteutuksissa ja integrointitavoissa. Kooijmans & al. [KCL+07] kutsuvat tätä aloitusvaiheen kartoitusta aloitustilanteen (Starting Scenario) määrittelyksi. He ovat löytäneet IBM-suurkoneita käyttävien yritysten järjestelmistä seuraavat vaihtoehtoiset aloitustilanteet: *terminaalisovellus* (3270 Application), *monikanavaratkaisu* (Multichannel), *eräajo* (Batch), *tiedon integrointi* (Data Access and Integration) ja *kotitekoinen SOA* (Homegrown SOA). Heidän mukaansa yrityksen järjestelmät sisältävät yleensä toteutuksia useista näistä aloitusvaihtoehdoista.

Aloitustilanteen kartoituksen jälkeen alkaa varsinainen siirtymisprosessi (Transition Process). Siirtymisprosessissa analysoidaan nykyiset liiketoimintaprosessit ja järjestelmien toiminnallisuus. Järjestelmien analysoinnissa on mahdollista käyttää hyväksi erilaisia etenemismalleja ja työkaluja. Kooijmans & al. [KCL+07] mainitsevat kaksi tällaista mallia: Service Oriented Modeling and Architecture (SOMA) ja Service Integration Maturity Model (SIMM). SOMA-malli käyttää useita työkalu- ja teknologiariippumattomia mallinnus-, analysointi- ja suunnittelutekniikoita SOA:n toteuttamiseksi. Malli muodostuu kolmen vaiheen noudattamisesta: tunnistaminen (Identification), määrittely (Specification) ja toteuttaminen (Realization). SIMM-malli puolestaan luotiin alunperin malliksi, jolla voidaan

tunnistaa yrityksen ja sen liiketoimintaympäristön kypsyys siirtyä palvelukeskeiseen arkkitehtuuriin. Näistä malleista SIMM on suppeampi ja pintapuolisempi ja se voidaan toteuttaa ryhmätyömallilla (Workshop).

Siirtyminen perussovelluksista palveluihin voidaan esittää myös mallipohjaisena etenemisenä IBM Patterns for e-Business [IPB] mallien mukaisesti. Tällöin etenemisvaiheet ovat seuraavat: 1) selvitä asiakkaan liiketoimintaan liittyvät vaatimukset, 2) tunnista liiketoimintavaatimukseen liittyvä liiketoimintamalli, 3) valitse integrointi- tai yhdistämismalli, joka rajaa liiketoimintavaatimuksen ratkaisua, 4) valitse sovellusmalli, joka tunnistaa kuinka sovelluksen logiikka on jaoteltu, 5) valitse sopiva IT-ympäristön ajonaikainen malli sekä 6) valitse tuotematriisimallilla sopivat ohjelmistot. [KCL+07]

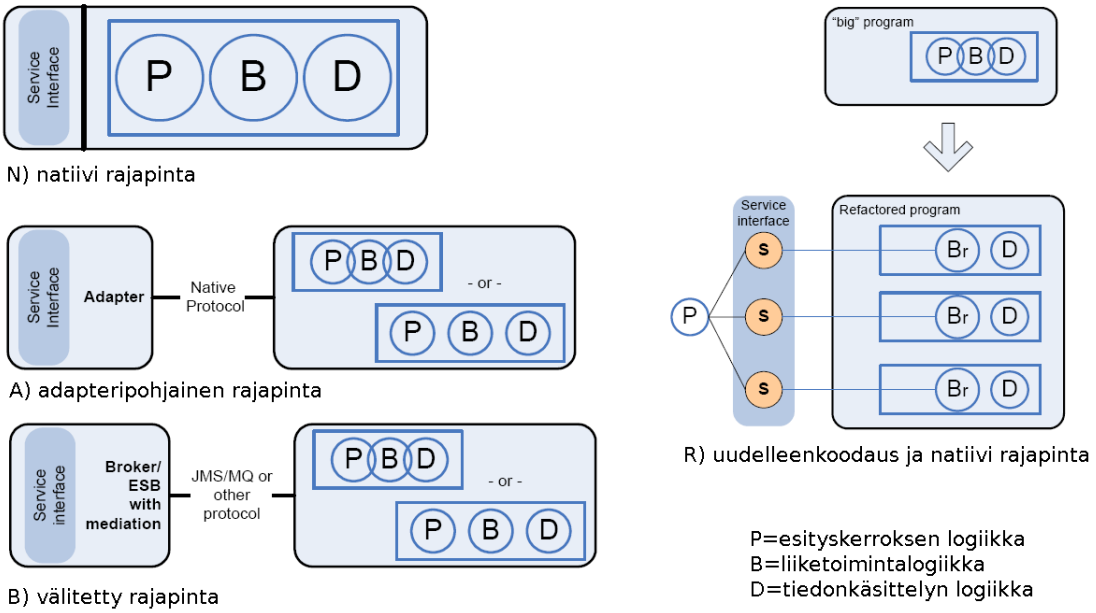
Palvelukeskeisen arkkitehtuurin toteutukseen liittyy aina järjestelmien integrointia. Teknisiä integrointitapoja on useita erilaisia, mutta ne voidaan jakaa neljään pääkategoriaan. Kuvassa 8 on esitetty Kooijmans & al. [KCL+07] mukaisesti erilaiset palvelurajapintamallit (Service Interface Patterns):

- Natiivi rajapinta (Native Interface, N): Perusjärjestelmän integrointiin käytetään olemassa olevaa integrointitapaa, joka on sisäänrakennettu joko tiedonkäsittelyjärjestelmään tai tapahtumankäsittelyjärjestelmään. Tuotteiden uusissa versioissa niihin on rakennettu mukaan tuki muun muassa SOAP ja JMS -protokollille.

- Adapteripohjainen rajapinta (Adapter-provided Service Interface, A): Tilanteissa jossa perusjärjestelmän ohjelmistot eivät kykene tarjoamaan SOA-kelpoista integraatiota, käytetään ulkoista ohjelmistokomponenttia toteuttamaan integraatio. Tässä tilanteessa adapteri on perusjärjestelmän ulkopuolinen komponentti.

- Välitetty rajapinta (Brokered/Mediated Service Interface, B): Välitetty rajapintamalli on lähellä adapteri-pohjaista. Siinä käytetään tyypillisesti erillistä ohjelmaa, kuten palveluväylä, toteuttamaan sovelluksen integrointi. Palveluväylä voi tarjota integroinnin lisäksi tiettyä lisäarvoa, kuten tietomuunnoksia. Integrointi voidaan toteuttaa, jollakin perusjärjestelmän tukemalla protokollalla tai voidaan hyödyntää natiivitoteutukseen pohjauvaa adapter-komponenttia.

- Uudelleenkoodaus ja natiivi rajapinta (Redeveloped Code With Native Service Interface, R): Tässä mallissa olemassa olevat sovellukset uudelleenkirjoitetaan siten, että ne muodostavat SOA-kelpoisia palveluita. Perusjärjestelmä kykenee tässä mallissa myös tarjoamaan palvelunsa SOA-kelpoisella protokollalla.



Kuva 8: Palvelun rajapintamallit [KCL+07].

5.2 SOA-palveluiden toteutus IMS-järjestelmälle

Palvelukeskeinen arkkitehtuuri on pohjimmiltaan teknologia- ja toteutuskieliriippumatonta, mutta web-palvelutekniikka on kuitenkin sen tyypillisin ilmenemismuoto. Ne ovat myös yksi keino toteuttaa SOA-palveluita IMS-järjestelmälle. Keen & al. [KBCG06] sekä Jäntti & al. [JiPS+06] esittelevät tapoja toteuttaa palveluita IMS-järjestelmälle web-palvelutekniikan avulla (kuva 9):

- IMS Connect: Yhteys muodostetaan suurkoneen IMS Connect -palvelun ja OTMA-rajapinnan kautta IMS-sovellukselle. Web-palvelun toteuttava asiakasosuus käyttää IMS Connect -asiakasohjelmistoa yhteyden muodostukseen. Kuvassa esimerkkinä IMS Connector for Java -ohjelmisto.

- MQ Bridge: IMS-sovellusta käytetään suurkoneen MQ-palvelimen sekä MQ-IMS Bridge -komponentin avulla OTMA-rajapintaa hyödyntäen. Asiakasosuus tässä esimerkissä käyttää MQ-jonoa ja JMS-palvelua yhteyden muodostukseen.

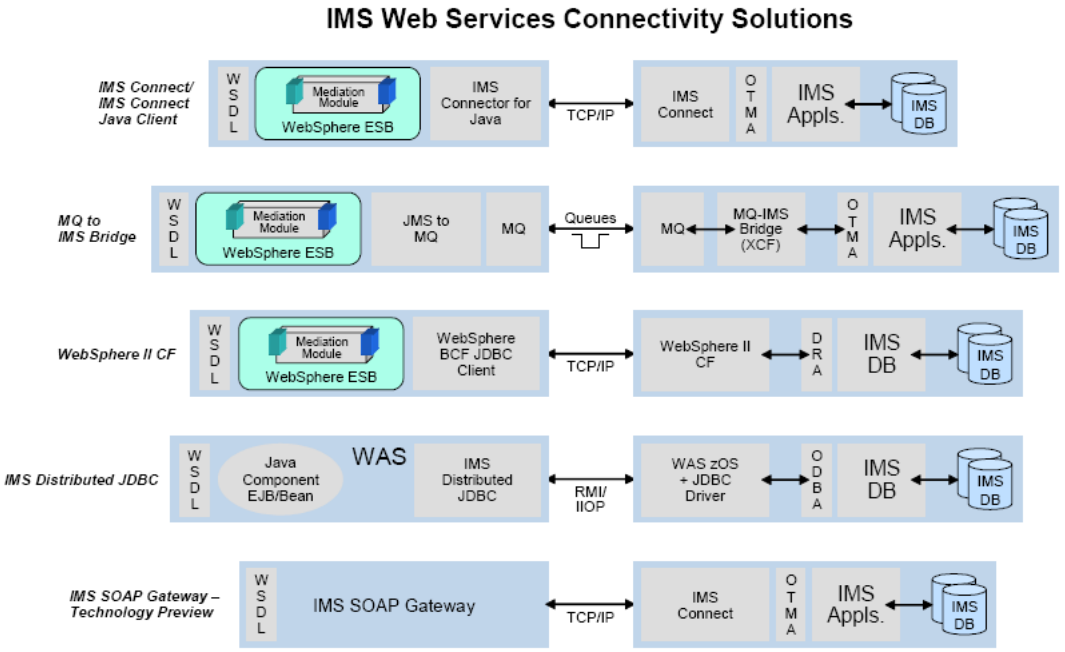
- WebSphere II CF: Tässä mallissa hyödynnetään ohjelmistoa, joka kykenee tekemään tiedosta palvelun (Information as a Service). WebSphere Information Integrator (WebSphere II) pystyy käyttämään tietolähteenään useita eri tietokantoja ja tarjoamaan palvelut useilla eri tekniikoilla. Kuvan esimerkissä käytetään IMS-tietokantaa DRA-rajapinnan kautta ja palvelun tarjoava asiakassovellus käyttää tietokantaa JDBC-yhteyden kautta.

- IMS Distributed JDBC: Hajautetulla alueella palveluita tarjoava WebSphere-sovelluspalvelin voi käyttää JDBC-rajapintaa IMS-tietokantojen käyttöön. Tämä on mahdollista asentamalla WebSphere-sovelluspalvelin z/OS-ympäristöön ja reitittämällä tietokanta-kyselyt sen käsiteltäväksi. Tietokannan käsittely tapahtuu ODBA-rajapinnan kautta.

- IMS SOAP Gateway: SOAP Gateway on erillinen ohjelmisto, joka asennetaan Unix- tai Windows-alustalle [IGW]. SOAP Gateway vastaanottaa SOAP-pyynnöt ja lähettää ne IMS Connect -palvelun kautta IMS:lle käsiteltäväksi. IMS Connect käyttää XML-muunnosta muuntaakseen SOAP-viestin IMS-muotoon ja takaisin XML-muotoon, ennen kuin se lähetetään palvelun käyttäjälle.

Edellä esitellyistä erilaisista tavoista toteuttaa web-palveluita, kolme integroituu IMS-tapahtumankäsittelyjärjestelmän sovelluksiin ja kaksi muuta IMS-tietokannanhallintajärjestelmään. Palveluiden toteuttamiseen voidaan soveltaa myös muita tekniikoita kuin web-palvelutekniikkaa. Edellä olevista malleista SOAP Gateway -ratkaisu rajoittuu vain web-palveluna tarjottaviin palveluihin, muissa tapauksissa voidaan soveltaa myös muita mekanismeja.

Edellä esitettyjen esimerkkitapausten lisäksi erilaisia palveluiden toteutusvaihtoehtoja löytyy lukuisia. Käytettävissä olevia toteutustapoja rajaavat lähinnä yrityksen olemassa olevat järjestelmät sekä niiden toteutustavat (lähtötilanne) ja haluttu tavoitetila SOA-muutosprosessissa. Kooijmans & al. [KCL+07] ovat löytäneet 29 erilaista toteutusvariaatiota, kun he ovat tarkastelleet kuhunkin lähtö-



Kuva 9: IMS-järjestelmän integrointi web-palvelutekniikalla [KBCG06, JiPS+06].

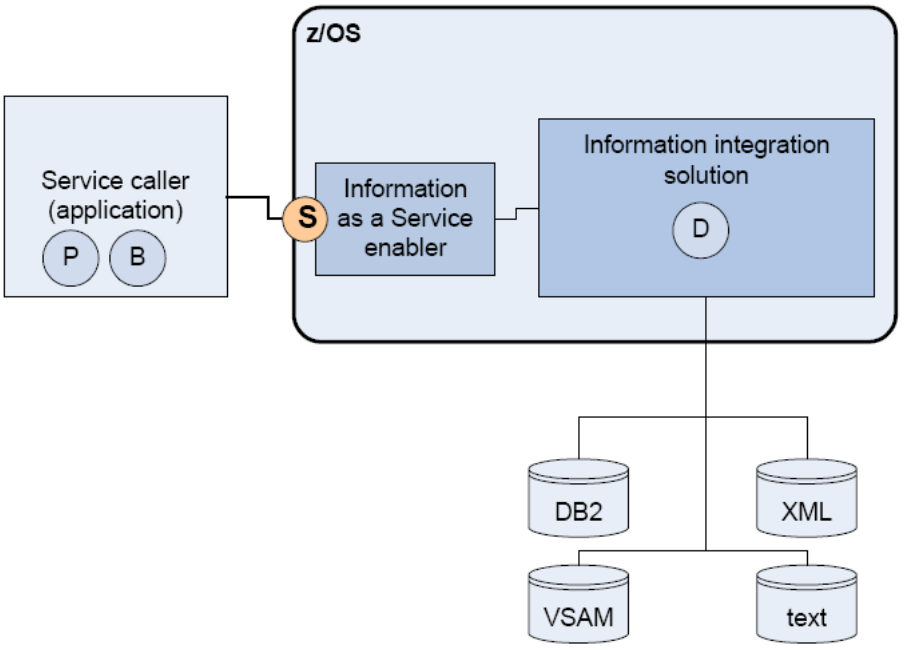
tilanteeseen sopivia tapoja toteuttaa haluttu muutosprosessi. Vaihtoehtoja löytyy vielä enemmän, sillä kuvatut tapaukset ovat lähinnä esimerkkejä erilaisista tilanteista.

5.3 Palveluiden toteutusvaihtoehtoja

Kohdassa 5.1 esiteltiin viisi Kooijmans & al. [KCL+07] tunnistamaa SOA-aloitustilannetta. Vaihtoehtoisista aloitustilanteista tarkastellaan jatkossa erityisesti kahteen niistä liittyviä palveluiden toteutusmalleja. *Terminaalisovellusten* (3270 Application) muuttaminen palvelukeskeiseen arkkitehtuuriin on valittu toiseksi tarkastelukohteeksi siksi, että siihen liittyvät toteutusmallit ovat tyypiltään olemassa olevien sovellusten toiminnallisuuden toteuttamista SOA-kelpoiseksi. Toinen valintaan vaikuttanut kriteeri oli se, että vaikka toteutusmalleissa tarkastellaan erityisesti 3270-sovelluksia, ne ovat pienin muutoksin sovellettavissa myös laajemmin muihin IMS-sovelluksiin. *Tiedon integrointi* on toinen valittu aloitustilanne. Sen valintakriteerinä oli mahdollisuus tarkastella toteutusmalleja, jotka tuottavat uusia SOA-palveluita ilman kytkentää aikaisempiin sovelluksiin.

Heterogeenisissä tietojärjestelmissä on usein päällekkäistä toiminnallisuutta eri

sovellusten välillä. Samoin päällekkäisyyttä löytyy usein järjestelmien tietosisältöjen osalta. Kun palveluita toteutetaan tietolähtöisesti, voidaan eri järjestelmissä oleva tieto esittää yhteisellä palvelulla. Noudatettaessa *tiedon integrointi* -aloitustilanteessa tietolähtöistä lähestymistapaa ohjaa se luontevasti toteutusmalleihin, joissa tiedosta muodostetaan palvelu, kuten kuvassa 10. Näistä menetelmistä voidaan käyttää yleistä termiä *tieto palveluna* (Information as a Service) [KCL+07].



Kuva 10: Tiedon esittäminen palveluna [KCL+07].

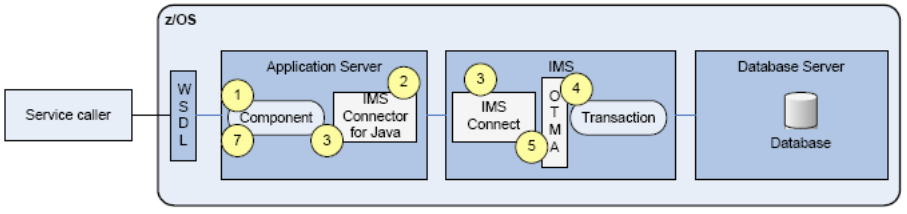
Seuraavissa alakohdissa tarkastellaan erilaisia tapoja toteuttaa palveluita. Näistä tarkastelunäkökulmista neljä liittyy palveluiden toteuttamiseen IMS-sovellusten pohjalta ja kaksi palveluiden toteuttamiseen IMS-tietokantojen pohjalta.

5.3.1 IMS Connect With Java Client

IMS Connect on IMS-tapahtumankäsittelyjärjestelmän keskeisin integrointitapa. Siihen ollaan yhteydessä TCP/IP-yhteydellä käyttäen erillistä asiakasohjelmaa. Tämä asiakasohjelma voi olla itse toteutettu tai se voi olla valmisohjelmisto, kuten IBM IMS Connector for Java. IMS Connector for Java on Java Enterprise Edition (Java EE) Connector Architecture (JCA) määrityksen mukainen yhteysohjelmisto (Connector). Yhteysohjelmisto huolehtii yhteyteen liittyvistä teknisis-

tä toteutuksen yksityiskohdista, kuten yhteyksien avaus ja sulkeminen sekä tietoturva. [JiPS+06]

Palvelun toteutus voidaan muodostaa esimerkiksi generoimalla IMS-ohjelman Cobol-kuvauksesta sanomaa käsittelevät Java-luokat. Palvelun toteutuksen jälkeen sen käyttö tapahtuu kuvan 11 mukaisesti, kuten Kooijmans & al. [KCL+07] esittävät. 1) Sovelluspalvelin vastaanottaa SOAP viestin asiakasohjelmalta. Se prosessoi SOAP otsaketiedot ja välittää pyynnön sitä käsittelevälle komponentille, kuten esimerkiksi Enterprise Java Bean (EJB). 2) EJB muuntaa viestin IMS-muotoon ja kutsuu IMS Connector for Java komponenttia, joka välittää pyynnön IMS Connectille. 3) IMS Connect vastaanottaa viestin, tekee tarvittavat muunnokset, tarkistaa pyynnön laillisuuden (tietoturva) ja valmistelee viestin sopivaan muotoon. 4) IMS välittää viestin OTMA:n kautta sitä käsittelevälle sovellukselle. 5) Vastaus välitetään OTMA:n ja IMS:n kautta IMS Connectille. IMS Connect valmistelee vastauksen sopivaan muotoon. 6) EJB muuntaa viestin takaisin SOAP-muotoon (kuvassa virheellisesti numero 3) ja 7) palauttaa vastauksen asiakasohjelmalle.



Kuva 11: IMS Connect integrointi [KCL+07].

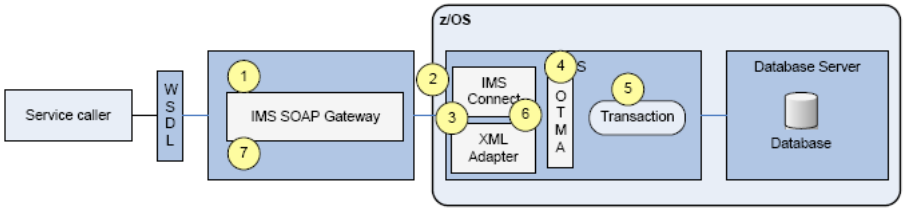
Kuvan esimerkissä WebSphere sovelluspalvelin on asennettuna z/OS-järjestelmään. IMS Connect mahdollistaa sen, että sovelluspalvelin on jokin muu kuin WebSphere ja että se sijaitsee suurkoneen ulkopuolella.

5.3.2 IMS SOAP Gateway

IMS SOAP Gateway on erillinen ohjelmisto, jolla IMS-järjestelmän sovellukset voidaan helposti tarjota web-palveluina SOAP-protokollaa käyttäen. IMS SOAP Gateway -ohjelmisto asennetaan z/OS-ympäristön ulkopuolelle Unix- tai Windows-järjestelmään. Se tarjoaa IMS-sovellusten palvelut SOAP-protokollan avulla asiakassovelluksille käyttäen omaa sisäänrakennettua Http-palvelintaan.

SOAP Gateway on yhteydessä IMS-järjestelmään tässäkin tapauksessa IMS Connectin kautta ja se lisää viestiin IMS-järjestelmää varten tapahtumakoodin. Asiakasohjelmiston XML-muotoiset SOAP-viestit muunnetaan IMS-muotoon joko IMS Connectin toimesta hyödyntäen ohjelmakuvauksesta generoituja XML-muunnosohjelmia (XML Adapter) tai IMS-ohjelmassa (vaatii muutoksia ohjelmaan). [IGW]

Web-palvelun WSDL-kuvaus voidaan muodostaa esimerkiksi generoimalla se IMS-ohjelman Cobol-kielisestä copybook-kuvauksesta. Palvelun käyttö etenee järjestelmässä seuraavien vaiheiden (kuva 12) mukaisesti: 1) IMS SOAP Gateway vastaaottaa asiakasohjelman SOAP-viestin, prosessoi sen XML-otsaketiedot ja selvittää tarvittavat IMS-yhteyden tiedot. 2) Viestiin lisätään tarvittavat IMS Connect -otsaketiedot ja se lähetetään IMS Connect-järjestelmään. 3) IMS Connect muuntaa XML-muotoisen SOAP-viestin IMS-muotoon, käyttäen XML Adapter -muunnosta hyödykseen. 4) IMS Connect kutsuu sovellusta OTMA-rajapinnan kautta. 5) Tapahtuma käsitellään normaalin IMS-tapahtuman mukaisesti ja vastaussanoma muodostetaan. 6) IMS Connect hyödyntää jälleen XML Adapteria muuntamaan vastaussanoman IMS-muodosta XML-muotoon ja lähettää sitten vastauksen takaisin SOAP Gatewaylle. 7) IMS SOAP Gateway lisää viestiin XML-otsaketiedot ja palauttaa vastauksen asiakasohjelmalle.



Kuva 12: IMS SOAP Gateway integrointi [KCL+07].

XML Adapter ei ole valmisohjelmisto, vaan se on Cobol-muunnoksen tuloksena generoitu muunnosohjelmisto, jota IMS Connect kutsuu. Viestin XML-muunnokset voidaan tehdä XML Adapterin lisäksi myös IMS-ohjelmassa. Tällöin IMS Connect ei kutsu XML-muunnosta kummassakaan tapauksessa, vaan IMS-ohjelma on vastuussa siitä, että se osaa käsitellä XML-muotoista tietoa, ja että se ohjelman suorituksen jälkeen muuntaa vastaussanoman sopivaan XML-muotoon. Tämä tapa vaatii yleensä muutoksia IMS-ohjelmaan, joten ratkaisun toteutus vaikeutuu huomattavasti.

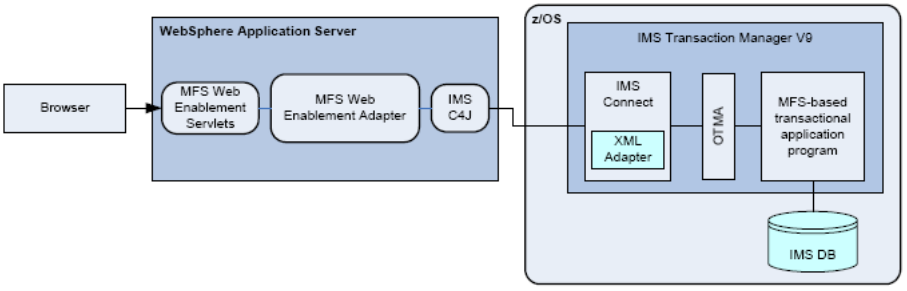
IMS SOAP Gateway on parhaimmillaan nopea tapa toteuttaa IMS-sovellusten palveluita web-palveluina, koska sen käyttö ei vaadi erillisen sovelluspalvelimen tai keskitetyn palveluväylän käyttöönottoa. Toisaalta sen ajoympäristön arkkitehtuuri ei välttämättä sovellu suoraan kriittiseksi tuotantoympäristöksi ilman erityisratkaisuja.

5.3.3 IMS MFS Web Support

IMS MFS Web Support -ohjelmistoja käyttämällä voidaan määritellä palvelu Message Format Services (MFS) muodossa olevasta sovelluksen kuvauksesta. Muodostettu palvelu voidaan sen jälkeen suorittaa WebSphere-sovelluspalvelimella joko EJB-luokan tai Javaproxy-luokkien kautta. Tämä toteutustapa käyttää IMS Connector for Java -asiakasohjelmistoa WebSphere-sovelluspalvelimella sekä IMS Connect -palvelua IMS-järjestelmässä. Palvelun toteutus muodostetaan generoimalla MFS tiedostostosta tarvittavat WSDL kuvaukset, Java-luokat sekä tietomuunnoksia varten määrittelytiedostot. Määrittelytiedostot kuvaavat muun muassa syöttö- ja tulostietojen sisältöä. [JiPS+06]

Kooijmans & al. [KCL+07] mainitsevat kaksi eri tapaa toteuttaa palveluita MFS Web Support -ohjelmistoilla. Ensimmäinen vaihtoehto (IMS MFS Web Services Support), toteuttaa MFS-pohjaisen sovelluksen toiminnallisuutta web-palveluna esimerkiksi EJB-komponentin kautta. Toinen vaihtoehto (IMS MFS Web Enablement), käyttää Javabean- ja Servlet-luokkia tarjotakseen MFS-pohjaisen sovelluksen käyttöliittymän selainpohjaiselle asiakasohjelmistolle (esimerkiksi Mozilla Firefox) (kuva 13). Tämä käyttöliittymä on 3270-terminaalisovelluksen näköinen. Kuvasta 13 poiketen, XML-muunnoksia ei tehdä IMS Connectin XML Adapter -toteutuksessa, vaan WebSphere-sovelluspalvelimen MFS Adapter -toteutuksessa [JiPS+06].

IMS 3270/MFS-pohjaiset sovellukset eivät välttämättä sovellu kovin hyvin palveluina tarjottavaksi (MFS Web Services Support), koska niiden hienojakoisuus on yleensä liian karkea palveluksi [KCL+07]. MFS Web Enablement sen sijaan sopii hyvin tilanteisiin, joissa tarvitaan nopeasti selainkäyttöinen pääsy IMS 3270-sovelluksiin. Toinen tapa tarjota selainkäyttöliittymä 3270-tyyppisille sovelluksille on IBM Host Access Transformation Services (HATS). HATS on yleiskäyttöisempi ja sen avulla voidaan muodostaa selainkäyttöliittymät IMS-sovellusten lisäksi



Kuva 13: IMS MFS Web Enablement integrointi [KCL+07].

CICS- ja TSO-sovelluksille.

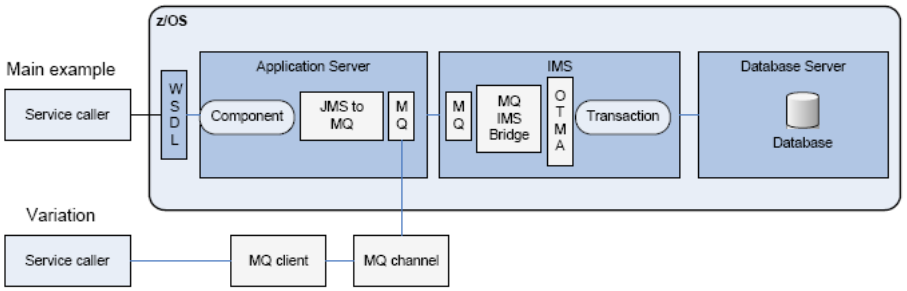
5.3.4 MQ-IMS Bridge

IMS Connect with Java Client, IMS SOAP Gateway ja IMS MFS Web Support käyttävät IMS-järjestelmän integrointiin IMS Connect -ohjelmistoa sekä IMS Connect -asiakasohjelmistoa, kuten edellisissä alakohdissa todettiin. MQ-IMS Bridge -ohjelmisto poikkeaa edellisistä siinä, että se muodostaa yhteyden suoraan IMS-järjestelmän OTMA-rajapintaan eli MQ Bridge toimii OTMA-asiakasohjelmistona [JLG+02]. Varsinaisen liiketoimintapalvelun asiakasohjelmana voi olla mikä tahansa ohjelmisto, joka pystyy kirjoittamaan MQ-jonoon.

Kooijmans & al. [KCL+07] ovat esittäneet esimerkin MQ-IMS Bridge toteutuksesta (kuva 14). Tässä kuvassa WebSphere-sovelluspalvelin tarjoaa web-palvelut asiakasohjelmille SOAP-rajapinnalla. Sovelluksen liiketoimintalogiikka on EJB-komponentissa, joka suorittaa myös viestin XML-käsittelyn sekä IMS-sanoman muodostamisen. EJB kirjoittaa viestin WebSpheren JMS-palvelun kautta WebSphere MQ:lle, joka reitittää viestin vastaanottavalle MQ-ohjelmistolle. MQ-IMS Bridge kirjoittaa viestin OTMA-rajapinnan kautta IMS-sovellukselle ja toimittaa mahdollisen vastaussanoman MQ-palvelimen vastausjonoon. Vastaussanoman lukemista varten täytyy WebSphere-palvelimella olla Message Driven Bean (MDB) lukemassa vastausjonoa.

5.3.5 IMS Database Access

WebSphere-sovelluspalvelin z/OS-järjestelmään asennettuna pystyy käsittelemään IMS-tietokantoja suoraan ODBA-rajapinnan kautta, kunhan WebSphere ja

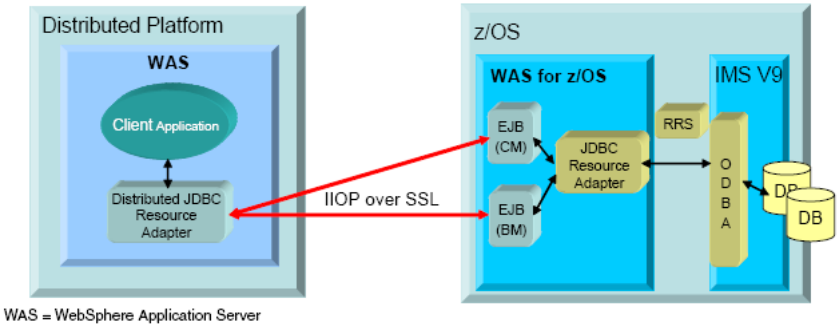


Kuva 14: MQ-IMS Bridge integrointi [KCL+07].

IMS ovat asennettuna samaan z/OS koneeseen (looginen osio). Tietokannan käsittely tapahtuu WebSphere-sovelluspalvelimelle asennettujen IMS Java -luokkien avulla. Nämä luokat toteuttavat J2EE Connector -arkkitehtuurin mukaisen resurssiadapterin nimeltään IMS JDBC Resource Adapter, Adapterin kautta voidaan käsitellä sekä IMS Full Function- että Data Entry -tietokantoja. [JiPS+06]

IMS Remote Database Services (RDS) ei ole varsinaisesti erillinen ohjelmisto, vaan toteutusmalli, joka koostuu sekä asiakasympäristöön että palvelinympäristöön asennettavista J2EE arkkitehtuurin mukaisista komponenteista. RDS mahdollistaa sen, että myös hajautetulla alueella (Unix, Windows) oleva WebSphere-sovelluspalvelin (asiakas) pystyy suorittamaan JDBC-kutsuja IMS-tietokantoihin. Sovelluksen tietokantakyselyt välitetään taustalla tietoliikenneyhteyttä hyödyntäen etäpalvelimelle suoritettavaksi. Asiakaspäähän on asennettava IMS Distributed JDBC Resource Adapter -resurssiadapteri ja z/OS-järjestelmän sovelluspalvelimelle (palvelin) IMS JDBC Resource Adapter -ohjelmisto. Tietoa käsittelevät EJB-komponentit toteutetaan z/OS-järjestelmän sovelluspalvelimelle ja asiakasosuuden sovelluspalvelin käyttää niitä Internet Inter-Orb Protocol (IIOP) yhteyden kautta. Kuvassa 15 on esimerkki RDS-palvelua käyttävästä ohjelmistokokonaisuudesta. [JiPS+06]

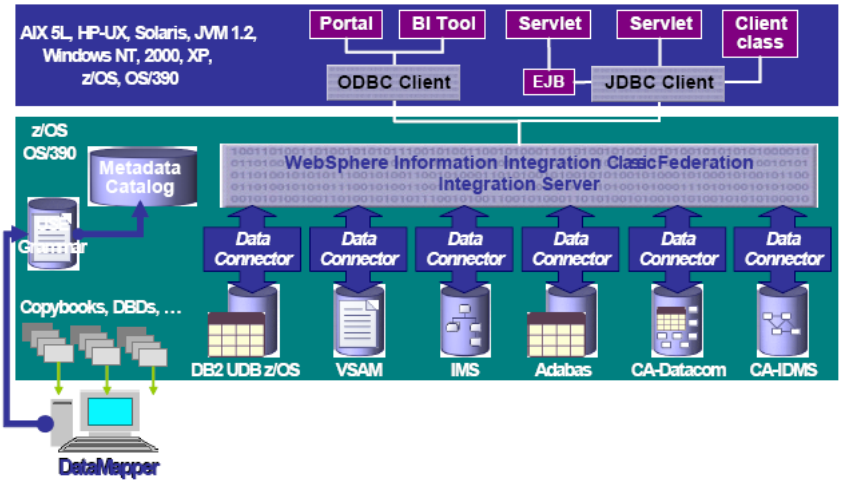
Edellä esitetyissä malleissa sovellusten tekeminen ja liiketoimintapalveluiden toteutus on lähes samanlaista. Ensimmäisessä vaihtoehdossa koko sovellus liiketoimintapalveluineen voidaan toteuttaa z/OS alueen sovelluspalvelimelle. Jälkimmäisessä tapauksessa ainoastaan tietokantojen käsittelyssä tarvittavat EJB-komponentit asennetaan z/OS alueen palvelimelle ja muu sovelluksen logiikka sekä tarjottavat palvelut sijaitsevat hajautetun alueen palvelimella. Valinta vaihtoehtojen välillä on lähinnä tekninen, mutta osittain myös kustannuspohjainen.



Kuva 15: IMS Remote Database Access integrointi [JiPS+06].

5.3.6 WebSphere Information Integrator

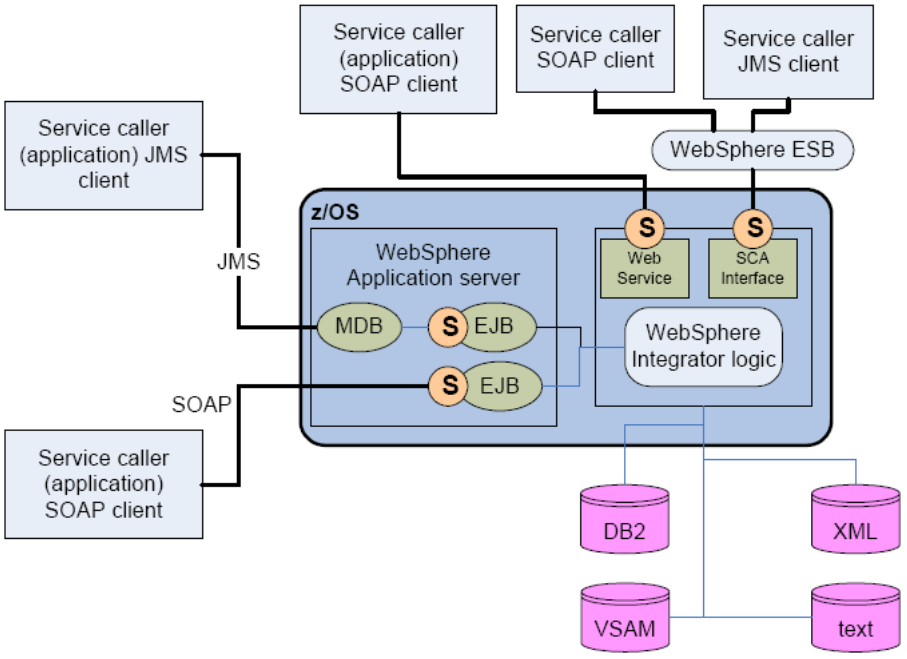
IMS-tietokantojen integrointiin voidaan IMS JDBC -yhteyden lisäksi käyttää integrointi-ohjelmistoja. WebSphere Information Integrator (WebSphere II) ohjelmiston avulla voidaan yhtenäistää näkymät eri tietolähteisiin. Näitä tietolähteitä voivat IMS-tietokantojen lisäksi olla muun muuassa DB2-tietokannat ja VSAM-tiedostojärjestelmän tiedostot. Yhteydet eri järjestelmiin tapahtuvat yhteyskomponenttien (Connector) avulla. WebSphere II tarjoaa kaikkiin tietolähteisiin relaatiopohjaisen näkymän ja se tallettaa eri tietolähteiden metatietoa omaan sisäiseen hakemistoonsa (Metadata Catalog). Sovellukset voivat relaatiopohjaisen tiedon käsittelyn lisäksi hakea tietoa hakemistosta. Kuvassa 16 on havainnollistettu monimutkaista WebSphere II järjestelmää, jossa on käytössä useita eri tietolähteitä. [JiPS+06]



Kuva 16: WebSphere II järjestelmäarkkitehtuuri [JiPS+06].

IMS-tietokantojen kohdalla yhteyskomponentti muuntaa saamansa SQL-kieliset lauseet IMS-järjestelmän DL/1-kutsuiksi. Käyttötarpeesta riippuen on valittavissa kolme erilaista yhteyskomponenttia. *IMS BMP/DBB/DLI interface* ei ole kovin skaalautuva ja se soveltuu lähinnä kehitysaikaiseen käyttöön. *IMS CCTL DRA interface* rajapinta on suositeltava mikäli sovellukset eivät käytä kaksivaiheista komennon hyväksyntää (Two-Phase-Commit) ja *IMS ODBA DRA interface* mikäli suorittavat. [JiPS+06]

WebSphere II tarjoaa sovellusten ohjelmoitiin JDBC-rajapinnan. Sovellukset voivat siis toteuttaa tietointegroinnin tämän rajapinnan avulla sekä toteuttaa liiketoimintalogiikan ja palveluiden toteutuksen sisäisesti haluamallaan tavalla. Kooijmans & al. [KCL+07] havainnollistavat kuvassa 17 tilanteen, jossa palveluita tarjotaan sovelluspalvelimen kautta web-palveluina sekä SCA-komponentteina palveluväylän kautta.



Kuva 17: WebSphere II palveluiden tarjoajana [KCL+07].

5.4 Toteutusvaihtoehtojen vertailu

IMS-järjestelmän palveluiden toteuttamiseksi palvelukeskeiseen arkkitehtuuriin soveltuviksi on useita tapoja. Kuhunkin tilanteeseen parhaiten sopiva toteutus-

tapa riippuu ainakin kolmesta seikasta: SOA-aloitustilanteesta, tavoitellusta perinnejärjestelmän muutosstrategiasta ja palvelun toteutusmallista.

Alakohdissa 5.3.1 - 5.3.6 tarkasteltiin kuutta eri tapaa toteuttaa palveluita IMS-järjestelmään. Näistä toteutustavoista neljä vastasi SOA-aloitustilannetta, jota Kooijmans & al. [KCL+07] kuvasivat terminaalisovellus-aloitustilanteeksi ja kaksi vastasi aloitustilannetta, jota he kuvasivat tiedon integrointi -aloitustavaksi. Taulukossa 1 esitellään nämä toteutustavat ja niihin sovellettavissa olevat SOA-muutosstrategiat sekä mahdolliset palvelukeskeisen palvelun toteutustavat [KCL+07]. Taulukossa on merkitty palvelun toteutustapa symbolilla (T), mikäli sitä on käytetty SOA:n terminaalisovellus-aloitustilanteessa sekä symbolilla (I) mikäli kyseessä on ollut tiedon integrointi -tyyppinen aloitustilanne.

Taulukko 1: IMS-järjestelmän palveluiden toteutustavat

Palvelun toteutustapa	*SOA-muutosstrategia*	*Palvelun toteutusmalli*
IMS Connect With Java Client (T)	mukauttaminen	yhdistämismalli
IMS SOAP Gateway (T)	mukauttaminen	yhdistämismalli
IMS MFS Web Support (T)	kehittäminen	alhaalta-ylös -malli
MQ-IMS Bridge (T)	mukauttaminen	yhdistämismalli
IMS Database Access (I)	mukauttaminen ja kehittäminen	yhdistämismalli ja alhaalta-ylös -malli
WebSphere Information Integrator (I)	mukauttaminen	yhdistämismalli

Tarkastelluista palvelun toteutustavoista neljä soveltuu SOA-muutosstrategiaan *mukauttaminen*. Näille toteutustavoille sopiva palvelun toteutusmalli on yhdistämismalli. IMS MFS Web Support -toteutustapa on muutosstrategiaan *kehittäminen* soveltuva ja sille soveltuva palvelun toteutusmalli on alhaalta-ylös -malli. IMS Database Access -toteutustapa puolestaan soveltuu hieman eri variaationa kumpaankin edellä esitetyistä muutosstrategioista ja toteutusmalleista.

Kooijmans & al. [KCL+07] esittelivät yhteensä 29 erilaista palvelun toteutustapaa. Kokonaisuutta laajemmin tarkastellen huomataan, että lähes kaikkiin SOA-aloitustilanteisiin löytyy vaihtoehtoja jokaisesta SOA-muutosstrategiasta. Toteutustavat poikkeavat joiltakin osilta toisistaan vain hiukan, joten lisää variaatioita

voi kehittää helposti. Tästä voidaan päätellä, että palvelun toteutustavan valinnassa on enemmän kyse tietoisesta valinnasta kuin valintakaavan noudattamisesta. Valinta ohjautuu tietyn toteutustavan suuntaan, kunhan tietyt reunaehdot (aloitustilanne ja muutosstrategia) on täytetty.

Toteutustavan valintaan vaikuttavat myös niin sanotut toissijaiset tekijät. Perinnejärjestelmien kehittämiseen saattaa olla käytettävissä vain niukasti osaajia, jolloin strategia siirtyy helposti lähemmäksi kehittämistä kuin perinnejärjestelmän uudistamista. Tällöin myös toteutusmallit ovat lähempänä alhaalta-ylös -mallia. Mikäli toteutustavan valintaa tarkastellaan vain taloudellisesti, päädytään yleensä samaan lopputulokseen, sillä kehittämis-strategia on pääsääntöisesti myös halvempi kuin uudistamisstrategia. Paras ja puhdasverisin SOA-lopputulos saavutettaisiin uudistamisstragialla ja ylhäältä-alas toteutusmalleilla. Nämä ratkaisut taas ovat usein taloudellisesti mahdottomia. Ehkä kaikista edellä mainituista syistä johtuen monesti päädytään juuri mukauttamisstrategian ja yhdistämismallien noudattamiseen.

6 Yhteenveto

Palvelukeskeinen arkkitehtuuri on tietojärjestelmien suunnitteluun ja toteutukseen liittyvä ajattelumalli. Sen periaatteita voidaan soveltaa myös pienessä mittakaavassa, jopa yksittäisessä sovelluksessa. Tyypillisimmin sitä kuitenkin käytetään suurehkon yrityksen tietojärjestelmäkokonaisuudessa. Usein tällaiseen kokonaisuuteen liittyy monia erilaisia tietojärjestelmiä ja tarve palvelukeskeiseen arkkitehtuurin tulee ensisijaisesti tarpeesta saada nämä järjestelmät toimimaan yhdessä.

Palvelukeskeisessä arkkitehtuurissa ydinajatus on, että tietojärjestelmien palvelut saadaan toteutettua riittävän yleisiksi ja sopivan tietosisällön omaaviksi, että niiden uudelleenkäyttö olisi helppoa. Toteutettavien palveluiden tulisi olla mahdollisimman löyhästi kytköksissä toisiina, myös saman tietojärjestelmän muihin palveluihin. Palvelukeskeiselle arkkitehtuurille on tyypillistä eri teknologioilla toteutettujen järjestelmien yhteistoiminta. Tämä saadaan aikaiseksi käyttämällä avoimiin standardeihin pohjautuvia tuotteita.

IBM-suurkone tehokkaana ja skaalautuvana laitteistona soveltuu hyvin SOA-järjestelmien ajoympäristöksi. IMS-järjestelmän tehokas tapahtumankäsittelyjärjestelmä ja suorituskykyiset tietokannat ovat teknisesti hyvin soveltuvia palvelukeskeisen arkkitehtuurin toteutukseen. Ongelmia saattaa esiintyä olemassa olevien sovellusten ja tietokantojen sopeuttamisessa palvelukeskeisen mallin mukaiseksi. Vanhoja tietojärjestelmiä voi olla pitkältä ajalta ja ne voivat poiketa myös toisistaan. Järjestelmän osaajien löytäminen on toinen siihen liittyvä mahdollinen ongelma. IBM-suurkoneelle voi ennustaa vahvaa tulevaisuutta SOA-ympäristöihin liittyen, sillä IBM:n panostus laitteiden, käyttöjärjestelmien ja ohjelmistojen kehittämiseen on viime vuosina ollut voimakasta. Toisaalta se on luopunut hiljattain pc-laitetuotannostaan, mikä varmasti osaltaan ohjaa panostusta suurkoneiden suuntaan.

Tietojärjestelmää mukautettaessa palvelukeskeiseen arkkitehtuuriin sopivaksi, on ensin arvioitava miten siihen suhtaudutaan strategisesti. Tällöin harkitaan, onko järjestelmä myös tulevaisuudessa aktiivisen kehitystyön kohteena, säilyykö sen toiminnallisuus entisellään vai tullaanko siitä kenties luopumaan jollakin aikavälillä. Strategisen tarkastelun jälkeen tiedetään mitä palveluiden toteutustapaa valit-

taessa on otettava huomioon. Palveluita toteutetaan joko ylhäältä-alas, alhaalta-ylös tai niistä muodostetun yhdistetyn toteutusmallin mukaisesti. Ylhäältä-alas toteutusmallin valinta tarkoittaa, että on päädytty strategiaan, jossa tietojärjestelmää kehitetään aktiivisesti kohti palvelukeskeistä arkkitehtuuria. Tällöin tietojärjestelmän rooli myös tulevaisuudessa nähdään vahvana. Ylhäältä-alas toteutusmalli tarkoittaa myös sitä, että palveluiden kehittäminen tehdään vahvasti SOA-periaatteiden pohjalta.

IMS-järjestelmän integroinnissa palvelukeskeiseen arkkitehtuuriin löytyy paljon erilaisia vaihtoehtoja. Vahvimmassa roolissa ovat IMS Connect -pohjaiset ratkaisut, kuten IMS Connector for Java ja IMS SOAP Gateway. MQ-Bridge -pohjainen integrointi on toimiva asynkronisessa viestien välityksessä ja tietopohjainen integointi sopii hyvin esimerkiksi eräohjelmalla käsiteltäviin tapahtumajonoihin. Se voisi soveltua myös tilanteisiin, joissa tehdään kokonaan uusia palveluita tieto palveluna -mallin mukaisesti.

Palveluiden toteutustavoista suuri osa pohjautuu malliin, jossa olemassa olevan sovelluksen tai tietokannan kuvauksesta muodostetaan ohjelmallisesti generoimalla palvelukeskeiseen arkkitehtuuriin sopiva palvelun kuvaus sekä apuohjelmia tiedon käsittelyyn. Näissä toteutustavoissa on se heikkous, että ne eivät suoraan mukauta entisen sovelluksen toiminnallisuutta tai tietokannan tietosisältöä palvelukeskeisen arkkitehtuurin mallien mukaiseksi.

Useimmissa palveluiden toteutustavoissa voidaan uuden palvelun sisältöön ja toiminnallisuuteen vaikuttaa muokkaamalla generoituja kuvauksia ja apuohjelmia. Tämä työ on pääsääntöisesti manuaalista työtä, joten se on virhealtista ja työlästä. Vaihtoehtona voisi olla esimerkiksi yritys- tai järjestelmäkohtaisen kielen (Domain Specific Language, DSL) kehittäminen. Tämän DSL-kielen avulla kuvattaisiin muunnosvaiheessa tarvittavia muutoksia generoitaviin tiedostoihin. DSL-kielinen kuvaus voitaisiin muodostaa vaikkapa graafisella työkalulla palvelukohtaisesti.

Viitteet

[BB07] Wesal Al Belushi and Youcef Baghdadi. An approach to wrap legacy applications into web services. In *Service Systems and Service Management, 2007 International Conference on*, pages 1–6, 9-11 June 2007.

[CFNS05] Francisco Curbera, Donald Ferguson, Martin Nally, and Marcia L. Stockton. *Service-Oriented Computing - ICSOC 2005*, chapter Toward a Programming Model for Service-Oriented Computing, pages 33–47. Springer Berlin / Heidelberg, 2005.

[DW07] Patricia Derler and Rainer Weinreich. *Trends in Enterprise Application Architecture*, chapter Models and Tools for SOA Governance, pages 112–126. Springer Berlin / Heidelberg, 2007.

[EAK06] Abdelkarim Erradi, Sriram Anand, and Naveen Kulkarni. Evaluation of strategies for integrating legacy applications as services in a service oriented architecture. *Services Computing, 2006. SCC '06. IEEE International Conference on*, pages 257–260, Sept 2006.

[EBA+06] Mike Ebbers, Frank Byrne, Pilar Gonzalez Adrados, Rodney Martin, and Jon Veilleux. *Introduction to the New Mainframe: Large-Scale Commercial Computing*. IBM International Technical Support Organization, 2006.

[EOO06] Mike Ebbers, Wayne O'Brien, and Bill Ogden. *Introduction to the New Mainframe: Z/OS Basics*. IBM, International Technical Support Organization, 2006.

[Erl05] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[Erl07] Thomas Erl. *SOA Principles of Service Design*. Prentice Hall PTR, 1 edition, 7 2007.

[HEHB08] Thorsten Hau, Nico Ebert, Axel Hochstein, and Walter Brenner. Where to start with soa: Criteria for selecting soa projects. In *Hawaii*

International Conference on System Sciences, Proceedings of the 41st Annual, pages 314–314, 7-10 Jan. 2008.

[Hes05] Howard M. Hess. Aligning technology and business: Applying patterns for legacy transformation. *IBM Systems Journal*, 44:25–45, 2005.

[HKB06] H. Harrer, G.A. Katopis, and W. Becker. Feature - from chips to systems via packaging - a comparison of ibm's mainframe servers. *Circuits and Systems Magazine, IEEE*, 6(4):32–41, Fourth Quarter 2006.

[Hun06] P. Hunter. The elephant that learnt to dance [zseries mainframes]. *IEE Review*, 52(3):36–40, March 2006.

[IGW] Ims soap gateway version 9.2. Technical report, IBM.

[Ioc07] P. Iocola. When legacy meets soa: Achieving business agility by integrating new technology with existing software asset. *Systems Conference, 2007 1st Annual IEEE*, pages 1–8, 9-13 April 2007.

[IPB] Ibm patterns for e-business. Technical report, IBM.

[JAK+04] Jouko Jäntti, Yukari Andoh, Knut Kubein, Rich Lewis, Geoff Nicholls, Dave Viguers, and Suzie Wendler. *IMS Version 9 Implementation Guide: A Technical Overview.* IBM, International Technical Support Organization, 2004.

[JiPS+06] Jouko Jäntti, Jordi Guillaumes i Pons, Gen Sasaki, Egide Van Aerschot, and Andres Wolf Andreoni. *IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity.* IBM International Technical Support Organization, 2006.

[JLG+02] Jouko Jäntti, Rick Long, Shannon Gordon, Gen Sasaki, Varadarajan Velamoor, and Suzie Wendler. *IMS e-business Connectors: A Guide to IMS Connectivity.* IBM International Technical Support Organization, 2002.

[Jos07] Nicolai M. Josuttis. *SOA in Practice - The Art of Distributed System Design.* O'Reilly Media, 2007.

[KBCG06] Martin Keen, Niall Betteridge, Diego Cardalliaguet, and Andreas Groeschl. z/os technical overview - websphere process server and websphere enterprise service bus. Technical report, IBM Redbooks Redpaper, 2006.

[KCL+07] Alex Louwe Kooijmans, Raymond Chiang, Irin Litman, Mats Pettersson, Bill Seubert, and Jens Erik Wendelboe. *SOA Transition Scenarios for the IBM z/OS Platform*. IBM, International Technical Support Organization, 2007.

[KdGRY06] Alex Louwe Kooijmans, Niek de Greef, Daniel Raisch, and Eran Yona. The value of ibm system z and z/os in service-oriented architecture. Technical report, IBM Redbooks Redpaper, 2006.

[LHHN00] Rick Long, Mark Harrington, Robert Hain, and Geoff Nicholls. *IMS Primer*. IBM, International Technical Support Organization, 2000.

[Lie07] Philipp Liegl. The strategic impact of service oriented architectures. *Engineering of Computer-Based Systems, 2007. ECBS '07. 14th Annual IEEE International Conference and Workshops on the*, pages 475–484, 26-29 March 2007.

[LJH+99] Rick Long, Jouko Jäntti, Robert Hain, Niel Kenyon, Martin Owens, and André Schoeman. *IMS e-business Connect Using the IMS Connectors*. IBM, International Technical Support Organization, 1999.

[LMSW07] Grace A. Lewis, Edwin Morris, Soumya Simanta, and Lutz Wrage. Common misconceptions about service-oriented architecture. *Commercial-off-the-Shelf (COTS)-Based Software Systems, 2007. ICCBSS '07. Sixth International IEEE Conference on*, pages 123–130, Feb. 26-March 2 2007.

[MLM+06] C. Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F Brown, and Rebekah Metz. Reference model for service oriented architecture 1.0. Technical report, OASIS, 2006.

[Pap07] Michael P. Papazoglou. *Software Architecture*, chapter What's in a Service?, pages 11–28. Springer Berlin / Heidelberg, 2007.

[PvdH07] Mike P. Papazoglou and Willem-Jan van den Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal The International Journal on Very Large Data Bases*, 16(3):389–415, 2007.

[RAB+06] Paul Rogers, Theodore Antoff, Patrick Bruinsma, Paul-Robert Hering, Lutz Kühner, Neil O'Connor, and Lívio Sousa. *UNIX System Services z/OS Version 1 Release 7 Implementation*. IBM International Technical Support Organization, 2006.

[Res07] Luciano Resende. Handling heterogeneous data sources in a soa environment with service data objects (sdo). In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 895–897, New York, NY, USA, 2007. ACM.

[SBC+07] Peter Swithinbank, Srinivas Bandaru, Graham Crooks, Andrew Ferrier, Jenny He, Raghunath Krishnaswamy, Vijay Mann, and Muriel Viale. *Connecting Enterprise Applications to WebSphere Enterprise Service Bus*. IBM, International Technical Support Organization, 2007.

[Sne06] H.M. Sneed. Integrating legacy software into a service oriented architecture. *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*, 00:11 pp.–, 22-24 March 2006.

[SR06] Bill Seubert and Daniel Ralsch. The role of ibm system z in the design of a service-oriented architecture. Technical report, IBM Redbooks Redpaper, 2006.

[w3c07] Soap version 1.2 part 1: Messaging framework (second edition). Technical report, W3C. Www-sivusto, http://www.w3.org/TR/2007/REC-soap12-part1-20070427/ (28.5.2008), 2007.

Liite 1: WSDL-kuvaus

WSDL 2.0 kuvaus, pohjautuu SOAP 1.2 protokollaan [Jos07]

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    targetNamespace="http://soa-in-practice.com/wsdl"
    xmlns:tns="http://soa-in-practice.com/wsdl"
    xmlns:xsd1="http://soa-in-practice.com/xsd"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:wsoap="http://www.w3.org/2006/01/wsdl/soap"
    xmlns:wsdlx="http://www.w3.org/2006/01/wsdl-extensions"
    xmlns="http://www.w3.org/2006/01/wsdl">

  <types>
    <xsd:schema
        targetNamespace="http://soa-in-practice.com/xsd"
        xmlns="http://soa-in-practice.com/xsd">

      <xsd:element name="getCustomerAddress">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="customerID" type="xsd:long"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="getCustomerAddressResponse" type="Address"/>
      <xsd:complexType name="Address">
        <xsd:sequence>
          <xsd:element name="street" type="xsd:string"/>
          <xsd:element name="city" type="xsd:string"/>
          <xsd:element name="zipCode" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>

    </xsd:schema>
```

```
  </types>

  <interface name = "CustomerInterface" >

    <operation name="getCustomerAddress"
               pattern="http://www.w3.org/2006/01/wsdl/in-out"
               style="http://www.w3.org/2006/01/wsdl/style/iri"
               wsdlx:safe = "true">
      <input messageLabel="In"
             element="xsd1:getCustomerAddress" />
      <output messageLabel="Out"
              element="xsd1:getCustomerAddressResponse" />
    </operation>

  </interface>

  <binding name="CustomerSOAPBinding"
           interface="tns:CustomerInterface"
           type="http://www.w3.org/2006/01/wsdl/soap"
           wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP">

    <operation ref="tns:getCustomerAddress"
               wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>

  </binding>

  <service name="CustomerService"
           interface="tns:CustomerInterface">

    <endpoint name="CustomerEndpoint"
              binding="tns:CustomerSOAPBinding"
              address="http://soa-in-practice.com/customer20"/>

  </service>

</description>
```

Liite 2: SOAP-viesti

Yksinkertainen SOAP versio 1.2 viesti [w3c07]

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
   <n:priority>1</n:priority>
   <n:expires>2001-06-22T14:00:00-05:00</n:expires>
  </n:alertcontrol>
 </env:Header>
 <env:Body>
  <m:alert xmlns:m="http://example.org/alert">
   <m:msg>Pick up Mary at school at 2pm</m:msg>
  </m:alert>
 </env:Body>
</env:Envelope>
```