Lauri Nevala

Web Services -tekniikoiden hyödyntäminen Java Platform, Micro Edition -ympäristössä

Tietojärjestelmätieteen
pro gradu -tutkielma
9.5.2008

Jyväskylän yliopisto
Tietojenkäsittelytieteiden laitos
Jyväskylä

TIIVISTELMÄ

Nevala, Lauri Tapani
Web Services -tekniikoiden hyödyntäminen Java Platform, Micro Edition -ympäristössä / Lauri Nevala
Jyväskylä: Jyväskylän yliopisto, 2008.
96 s.
Pro gradu -tutkielma

Mobiilisovellukset voidaan integroida Web Services -tekniikoiden avulla osaksi yritysten tietojärjestelmiä. Tässä tutkielmassa selvitetään kuinka laajasti näitä tekniikoita voidaan hyödyntää erityisesti Java ME -ympäristössä ja mitä vaikutuksia tekniikoiden käytöstä aiheutuu.

Web Services -tekniikoiden hyödyntämisen mahdollisuutta on selvitetty kirjallisuuskatsauksen avulla. Lopputuloksena on taulukko Web Services -tekniikoista, joita on mahdollista tällä hetkellä käyttää Java ME -ympäristössä. Web Services -tekniikoiden käytön vaikutuksia arvioidaan tutkielmaa varten toteutettujen konstruktioiden avulla.

Web Services -tekniikoiden hyödyntäminen tekee sovelluksen toiminnallisuudesta tehokkaamman kuin vastaavan toiminnallisuuden suorittaminen laitteen Web-selaimen avulla. Java ME:n Web Services -tekniikat ovat vielä kehittymättömiä palvelukeskeisen integraation vaatimuksiin. Ydintekniikoista XML:n ja SOAP-protokollan hyödyntäminen on kuitenkin mahdollista. Tulevaisuudessa tulisikin keskittyä myös muiden Web Services -tekniikoiden tukemiseen, joiden avulla mobiilisovellukset voidaan liittää entistä turvallisemmin ja luotettavammin yritysjärjestelmiin.

AVAINSANAT: Java Platform Micro Edition, EAI, palvelukeskeinen ohjelmistojen integrointi, Web Services, liikkuva tietojenkäsittely

SISÄLLYSLUETTELO

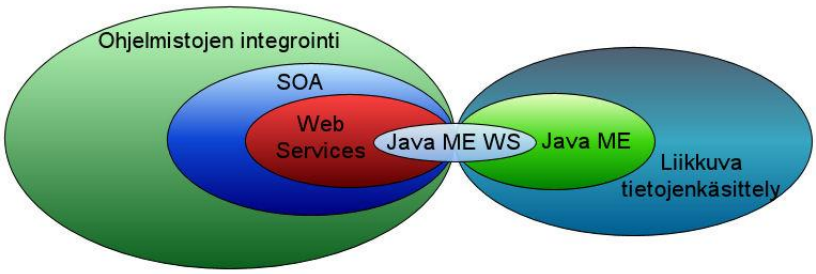
1 JOHDANTO

Matkapuhelimet ja kämmentietokoneet ovat yleistyneet huimasti viimeisen kymmenen vuoden aikana. Samaan aikaan laitteiden ja langattomien tietoverkkojen kehitys on mahdollistanut hyvin monipuolisten palveluiden tarjoamisen laitteiden käyttäjille. Teknisen kehityksen ansiosta on herännyt ajatus päästää käyttäjät yritysten tietojärjestelmiin myös mobiilien päätelaitteiden avulla. Järjestelmien integrointi on ollut arkipäivää perinteisissä ohjelmistoissa jo 1970-luvun alkupuolelta lähtien ja nykyiset yritysjärjestelmät ovat suunniteltu tehokkaille tietokoneille sekä kiinteille ja nopeille tietoverkkoyhteyksille. Tietojärjestelmien integrointiin on kehitetty lukuisia lähestymistapoja. Nämä lähestymistavat antavat suuntaviivat tekniikoille, joiden avulla integrointi on mahdollista toteuttaa. Mobiilit päätelaitteet ovat vielä resursseiltaan vajavaiset ja verkkoyhteydet katkonaiset. Sekä laitteiden vajavaiset resurssit että integroinnissa käytettävien tekniikoiden kehittymättömyys liikkuvan tietojenkäsittelyn tarpeisiin asettavat haasteita mobiilisovellusten integrointiin.

Yritysten organisaatiot on tavallisesti jaoteltu yrityksen sisällä eri osastoihin, joilla jokaisella on omat itsenäiset tietojärjestelmänsä. Erilliset tietojärjestelmät johtavat usein saman informaation ylläpitämiseen useassa eri paikassa. Ylläpidettävät tiedot voivat myös olla ristiriitaiset keskenään. Perinteisesti tietojärjestelmät on yhdistetty yrityksen sisällä keskenään, jolloin lopputuloksena on valtava yhteyksien määrä eri järjestelmien välillä. Ratkaisuna tähän 1990-luvulla alettiin puhua Enterprise Application Integration:sta (EAI). EAI:n mukaisesti nämä erilliset tietojärjestelmät yhdistetään yhdeksi järjestelmäksi erillisen väliohjelmiston avulla, joka huolehtii yhteydet eri sovellusten kesken.

EAI ei ota kuitenkaan kantaa, kuinka tällainen integraatio toteutetaan ja mitä tekniikoita tulee käyttää järjestelmien integroinnissa. Onkin kehitetty lukuisia integrointilähestymistapoja, joiden avulla tällaiseen lopputulokseen voidaan päästä. Palvelukeskeinen ohjelmistojen integrointi ja Web Services -tekniikat ovat osoittautuneet lupaaviksi lähestymistavoiksi, joiden avulla myös mobiilit päätelaitteet ja sovellukset voivat päästä yritysten tietojärjestelmien sisältämiin tietoihin standardilla tavalla. Jotta mobiilisovellukset voidaan integroida osaksi muuta palvelukeskeisesti integroitua järjestelmää, on mobiilisovellusten tuettava integroinnissa käytettyjä tekniikoita.

Tämän tutkielman tarkoituksena onkin selvittää, kuinka laajasti Web Services -tekniikoita voidaan hyödyntää mobiilisovelluksissa erityisesti Java ME -ympäristössä ja mitä vaikutuksia tekniikoiden käytöllä on. Tutkielmassa yhdistyy siis kaksi hyvin erilaista kokonaisuutta: ohjelmistojen integrointi, palvelukeskeinen ohjelmistojen integrointi ja Web Services -tekniikat sekä liikkuva tietojenkäsittely ja erityisesti Java ME -ympäristö (KUVIO 1).



KUVIO 1. Tutkielman aihealueiden limittyminen.

Web Services -tekniikoiden hyödyntämisen mahdollisuutta ja laajuutta Java ME -ympäristössä arvioidaan kirjallisuuden perusteella kvalitatiivisesti. Lopputuloksena on taulukko, josta käy ilmi, mitä Web Services -tekniikoita tämän hetkiset Java ME -sovellusliittymät tukevat.

Web Services -tekniikoiden käytön vaikutuksia arvioidaan tutkielmaa varten toteutettujen konstruktioiden avulla. Konstruktioiden avulla verrataan langattoman laajennetun Internetin mallia ja langattoman portaalin mallia. Langattoman laajennetun Internetin mallissa Web Services -palveluita kutsutaan suoraan asiakaskerroksessa. Langattoman portaalin mallissa sen sijaan Web Services -palveluita kutsutaan palvelinkerroksessa ja tulokset esitetään asiakkaalle Internet-selaimen avulla. (Yuan ja Long 2002.) Web Services -tekniikoiden käytön vaikutuksia arvioidaan kvantitatiivisella tutkimusmenetelmällä.

Järjestelmäintegraatio ja *ohjelmistojen integrointi* on tapa ja tekniikka, jolla automatisoidaan erityyppisten tietojärjestelmien ja ohjelmistojen vuoropuhelua.

Palvelukeskeinen ohjelmistojen integrointi on integrointilähestymistapa, jossa järjestelmän eri osat tarjoavat resurssejaan muiden järjestelmän osien käyttöön palveluiden avulla.

Web Services -tekniikat mahdollistavat tietojärjestelmien integroinnin palvelukeskeisesti tarjoten tekniikat palveluiden määrittämiseen, kuvaamiseen ja löytämiseen sekä tiedon siirron järjestelmien välillä standardilla ja turvallisella tavalla.

Liikkuva tietojenkäsittely (*mobile computing*) tarkoittaa yksinkertaisesti tietojenkäsittelyä, joka tapahtuu mobiilissa päätelaitteessa. Satyanarayanan (1996) määrittelee liikkuvalle tietojenkäsittelylle neljä tunnuksenomaista rajoitetta. Ensiksi mobiilien laitteiden resurssit ovat pieniä. Siksi ne ovat riippuvaisia staattisista tekijöistä. Toiseksi mobiilit päätelaitteet ovat luonnostaan alttiita varkauksille ja rikkoontumisille. Kolmanneksi laitteiden verkkoyhteydet ovat erittäin muuttuvia nopeudeltaan ja luotettavuudeltaan. Neljänneksi mobiilit päätelaitteet perustuvat rajalliseen energialähteeseen. Nämä muodostavat liikkuvan tietojenkäsittelyn erityispiirteet ja aiheuttavat mobiilisovellusten suunnittelun monimutkaisuuden. Yleisesti, jos langaton laite pystyy hajautettuun tiedonkäsittelyyn verkkoviestinnän lisäksi, voidaan puhua liikkuvasta tietojenkäsittelystä.

Mobiili päätelaite on laite, joka on kykenevä liikkuvaan tietojenkäsittelyyn. Mobiili päätelaite on langaton ja kannettava. Laitteen käyttäjällä on pääsy samaan tai samanlaisiin telekommunikaatiopalveluihin eri paikoissa. Laitteen kannettavuus tarkoittaa, että tietoliikennelaite liikkuu käyttäjän kanssa tai ilman, jolloin verkon ja laitteen ominaisuudet varmistavat, että tietoliikenne on mahdollista laitteen liikkuessa. (Schiller 2001, 1.) Laitteen on myös mahduttava taskuun ja sitä on pystyttävä käyttämään yhdellä kädellä tai kokonaan ilman käsiä (Hjelm 2000, 3).

Mobiilisovelluksella tarkoitetaan tässä tutkielmassa sovellusta, joka kokonaan tai osittain sijaitsee mobiilissa päätelaitteessa.

Java Platform, Micro Edition (Java ME) on Java-alusta, joka on tarkoitettu sulautettujen järjestelmien ja pienten, resursseiltaan rajoittuneiden laitteiden ohjelmistojen toteutukseen. Muut Java-alustat ovat palvelinten ja laajojen yritysovellusten ohjelmointiin tarkoitettu Java Platform, Enterprise Edition (Java EE) ja työasemien ja pöytätietokoneiden sovellusten ohjelmointiin tarkoitettu Java Platform, Standard Edition (Java SE).

Kirjallisuudessa on käsitelty paljon järjestelmien ja ohjelmistojen integrointia, mutta yleensä niiden lähtökohtana ovat kiinteät verkot sekä tietokoneet ja työ-

asemat. Web Services -tekniikoiden käytön mahdollisuutta mobiileissa päätelaitteissa ja niiden sovelluksissa on käsitelty jonkin verran, mutta pääpaino on ollut niissä lähinnä SOAP-protokollan käytön ja XML-kielen tuomissa eduissa ja haitoissa sekä niiden minimoinnissa. Java ME:a käsittelevässä kirjallisuudessa pääpaino on sen sisältämien sovellusliittymien ominaisuuksien esittelyssä. Tässä tutkielmassa selvitetäänkin laajemmin mobiilisovellusten integrointimahdollisuuksia palvelukeskeisesti Web Services -tekniikoiden avulla. Tutkielmassa arviointi rajataan Web Services -tekniikoiden hyödyntämiseen Java ME -ympäristön MIDP-profiilissa, mutta tulokset ovat sovellettavissa myös muissa mobiilialustoissa (mm. Symbian, Windows Mobile sekä Android).

Luvussa 2 selvitetään yleisellä tasolla, mitä erityispiirteitä ja lähestymistapoja ohjelmistojen integrointiin on olemassa. Tämä katsaus taustoittaa, mistä järjestelmäintegraatiosta on kysymys. Lisäksi tarkastellaan tarkemmin palvelukeskeistä ohjelmistojen integrointilähestymistapaa. Tarkastelun avulla saadaan käsitys, mihin haasteisiin Web Services -tekniikoiden tulee vastata, jotta ohjelmistoja voidaan integroida palvelukeskeisesti. Web Services -tekniikoita sekä niiden käytöstä aiheutuvia hyötyjä ja haittoja tarkastellaan lähemmin luvussa 3. Luvussa käsitellyt tekniikat toimivat myöhemmin pohjana, kun arvioidaan niiden hyödyntämisen mahdollisuutta mobiilisovelluksissa. Luvussa 4 selvitetään strategioita, joilla Web Services -tekniikoita voidaan hyödyntää liikkuvassa tietojenkäsittelyssä. Lisäksi luvussa käsitellään yleisiä hyötyjä ja haittoja, joita tekniikoitten käytöstä aiheutuu liikkuvan tietojenkäsittelyn kontekstissa.

Luvussa 5 luodaan tarkempi katsaus Java ME -ympäristöään. Luvussa selvitetään, mitä ominaisuuksia ympäristö sisältää yleisellä tasolla. Katsauksen perusteella selviää tarkempi kuva, mitä mobiilisovelluksilla tässä tutkielmassa käsitetään ja mitä ominaisuuksia sovellukset voivat sisältää. Luvussa selvitetään myös tarkemmin, miten Web Services -palveluiden kutsuminen on mahdollista Java ME -ympäristössä ja mitä Web Services -tekniikoita on mahdollista hyödyntää.

Kun on saatu selville, mitä Web Services -tekniikoita Java ME -ympäristössä on mahdollista käyttää, on kiinnostavaa tietää, mitä vaikutuksia tekniikoitten käytöstä konkreettisesti aiheutuu. Luvussa 6 kuvataan tässä tutkielmassa käytetyt konstruktiot ja metriikat, joiden avulla vaikutuksia pystytään arvioimaan. Luvussa 7 esitetään arvioinnin tulokset, joita analysoidaan tutkielman päättävässä luvussa 8.

2 JÄRJESTELMÄINTEGRAATIO

Järjestelmäintegraatiossa automatisoidaan erityyppisten tietojärjestelmien vuoropuhelua. Itse informaation jakaminen järjestelmien välillä ei kuitenkaan ole itseisarvo, vaan integraatiosta tulee olla jotain konkreettista hyötyä. (Tähtinen 2005, 22.)

Järjestelmien integroinnilla haetaan samanlaista hyötyä kuin mistä tahansa tietojenkäsittelysovelluksista – tehokkuutta. Informaation jakamisen automatisointi nopeuttaa prosesseja ja vähentää virheitä. Prosessien nopeutuminen ja virheiden väheneminen puolestaan johtaa kustannussäästöihin ja tätä kautta kilpailukyvyn paranemiseen. Lisäksi automatisoinnilla vapautetaan manuaalisten prosessien suoritukseen sitoutunutta työvoimaa. Prosessien nopeutuminen voi johtaa asiakastyytyväisyyden kasvuun ja sitä kautta pidempiin ja kannattavampiin asiakassuhteisiin. (Tähtinen 2005, 23-25.)

Järjestelmäintegraatio helpottaa myös informaation synkronointia eri järjestelmien välillä (Tähtinen 2005, 25). Kerran järjestelmään syötetty tieto on useiden eri järjestelmien käytettävissä sen sijaan, että se pitäisi syöttää jokaiseen järjestelmään erikseen. Tämä itse tiedon syöttämiseen kuluvan ajan säästämisen lisäksi, tiedon virheellisyyden todennäköisyys pienenee ja tiedon yhtenäisyys paranee.

Integroitavan järjestelmän kokonaisarvo kasvaa mitä enemmän siihen liitetään organisaatiossa olevia tietojärjestelmiä. Järjestelmäintegraatio kannattaakin suunnitella huolellisesti ja kaukonäköisesti, koska tulevaisuudessa järjestelmien verkostoa saatetaan kasvattaa huomattavasti alkuperäistä laajemmaksi. Huonosti ja kevyesti rakennettu integraatioratkaisu saattaa pahimmassa tapauksessa kasvattaa ylläpitokustannukset suuremmiksi kuin informaation jakamisesta saatu hyöty. (Tähtinen 2005, 22-23.)

2.1 Yritysten sisäinen integraatio - Enterprise Application Integration (EAI)

Viime vuosien teknologinen kehitys on mahdollistanut organisaatioiden muuttumisen globaaleiksi verkostoiksi. Tällaiset globaalit organisaatiot eivät voisi toimia ilman yritysten toimintojen integrointia. Yrityksen toimien integroiminen vaatii sekä organisatorista että teknologista integraatiota (Lee, Siau ja Hong 2003, 57).

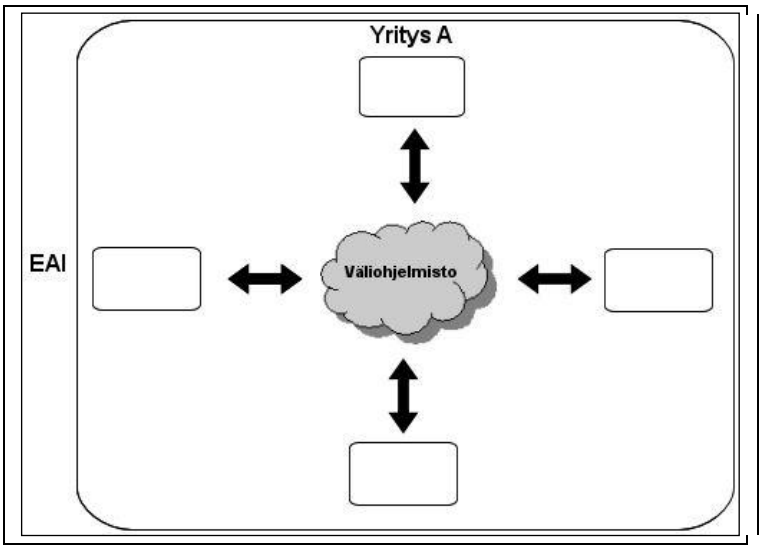
Yritysten erillisten osastojen tietojärjestelmien integroinnissa on kuitenkin monta haastetta ja ongelmaa. Järjestelmät on mahdollisesti kehitetty aikaan, jolloin järjestelmät toimivat keskitetyssä tietojenkäsittely-ympäristössä ja tieto, prosessit sekä vasteet olivat homogeenisiä (Linthicum 2000, 6).

1990-luvun alussa kehittyi kaksi erillistä integraatiolähestymistapaa: toiminnanohjaus (*Enterprise Resource Planning, ERP*) ja tietovarastointi (*Data Warehousing*). Toiminnanohjausjärjestelmät keskittyvät operationaaliseen integrointiin ja sitä kautta tukemaan päivittäisiä prosesseja, kun taas tietovarastointi keskittyy informaation integrointiin tukemaan päätöksen tekoa.

Enterprise Application Integration (EAI) termiä alettiin käyttää 1990-luvun puolivälissä. EAI käsitteellä tarkoitetaan suunnitelmia, menetelmiä ja työkaluja, joilla uudistetaan, yhdistetään ja koordinoidaan yrityksen koko tietojenkäsittely (Lee, Siau ja Hong 2003, 57). EAI on tiedon ja liiketoimintaprosessien rajatonta jakamista kaikkien yrityksen yhteydellisten sovellusten välillä (Linthicum 2000, 3). Yksinkertaisimmillaan se on varsin teknistä informaation siirtämistä eri järjestelmien välillä (Tähtinen 2005, 33). Tyypillisesti yrityksen perinnejärjestelmät yhdistetään uusien ohjelmistojen kanssa, jotka hyödyntävät Internetiä, elektronista liiketoimintaa, ekstranettejä ja muita uusia teknologioita. Nämä toimenpiteet ovat aikaa vieviä ja siten kalliita. EAI:n perusajatus onkin integroida yrityksen tietojärjestelmät pienemmin kustannuksin, vähemmällä ohjelmoinnin määrällä. (Linthicum 2000, 3).

EAI voi johtaa myös täysin uuden yrityksen liiketoiminnan näkemysten ja sen sovellusten muodostamiseen. Tällöin selvitetään, kuinka vanhat järjestelmät soveltuvat uuteen ajattelutapaan ja suunnitellaan niiden tehokkaat uudelleenkäyttötavat samalla kun luodaan uusia järjestelmiä.

Verrattuna perinteisiin kalliisiin ja aikaa vieviin järjestelmäintegraatiomalleihin, joissa joudutaan kirjoittamaan uudelleen koodia yhdistettäviin järjestelmiin, EAI:ssa käytetään erityistä väliohjelmistoa (*middleware*), joka toimii siltana eri järjestelmien välillä (KUVIO 2). Tällöin sovellukset voivat kommunikoida yhteisen rajapinnan kautta ennemmin kuin kaksipisteyhteydellä (*point-to-point connection*), jolloin ohjelmoinnin tarve vähenee (Lee, Siau ja Hong 2003, 57-58).



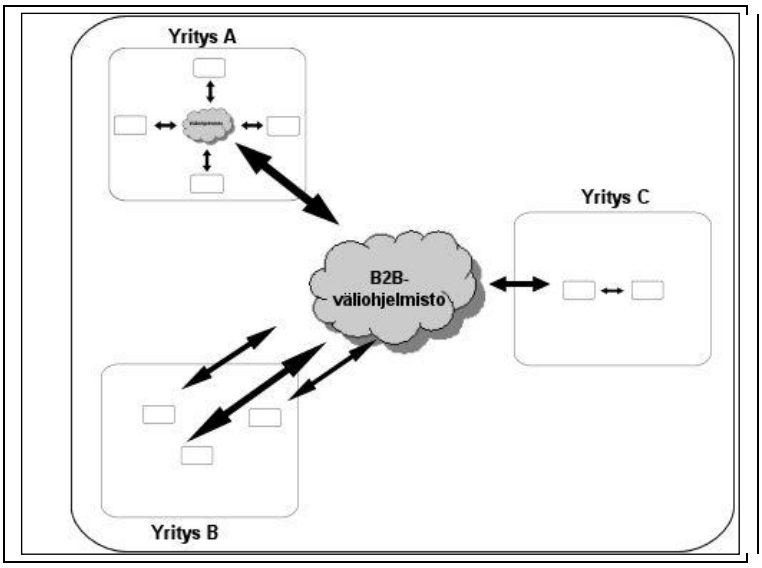
KUVIO 2. Järjestelmien integrointi erillisen väliohjelmiston avulla EAI:ssa.

EAI-toteutuksissa on havaittu myös joitakin puutteita ja rajoitteita. Ensinnäkin EAI vaatii suuren panostuksen suunnitteluvaiheeseen. Toiseksi onnistunut integroinnin toteutus vaatii vahvan viestinnän, koordinoinnin ja yhteistyön yrityksen henkilöstön välillä. Kolmanneksi lähestymistapa vaatii liiketoimintaprosessien etukäteiskartoituksen. (Lee, Siau ja Hong, 2003. 58.)

2.2 Yritysten välinen integrointi - Business-to-business integration (B2B AI)

Nykyään harva tuote on täysin yhden valmistajan alusta loppuun saakka kokonaan itse valmistama, vaan yleensä tuotteen valmistukseen osallistuu yritysten muodostama tuotantoketju alkaen raakamateriaalien tuottajista lopputuotteen valmistajiin ja tuotteen oston jälkeiseen huolto- ja ylläpitopalveluiden tarjoajiin. Tapahtumaperustaisessa liiketoiminnassa tehokkaan tuotantoketjun mahdollistamiseksi on tieto ostotapahtumasta kuljettava mahdollisimman nopeasti myyjältä valmistajalle, valmistajalta osakomponenttien valmistajille ja alihankkijoilta raakamateriaalin tuottajille. Paperityönä tehtynä tiedon kulkuun menee kuukausia, kun automatisoituna siihen saisi kulua yksi päivä. Osittain tämä tiedonvälitys saadaan hoidettua organisaatioiden välisellä tiedonsiirrolla (OVT) (*Electronic Data Interchange, EDI*), mutta tämä ei ole täysin riittävä. Tuotantoketjun eri osapuolilla on käytössä lukuisia eri tietojärjestelmiä, joiden tehokas integroiminen takaa tehokkaan tuotantoketjun. Tehokas tuotantoketju nopeuttaa liiketoimintaprosesseja, alentaa kustannuksia ja kasvattaa kilpailuetua. (Linthicum 2001, 3-9.)

Yrityksillä on lukuisia ohjelmistoja, jotka on saatettu integroida EAI:lla tehokkaaksi yhdeksi järjestelmäksi. Näistä integroiduista tiedoista ja prosesseista saatu hyöty ei kuitenkaan hyödytä yrityksen yhteistyökumppaneita, vaan jokainen yhteistyöverkoston yritys varastoivat osittain samat tiedot omin järjestelmiin. Tällöin on selkeä tarve B2B-ohjelmistojen integroinnille, jolloin tiedot ja prosessit ovat yhteneväiset ja käytettävissä samanlaisina koko tuotantoketjussa. B2B-ohjelmistojen integrointi laajentaa EAI:n tarjoamalla ulkopuolisille toimijoille pääsyn yrityksen järjestelmiin erillisen väliohjelmiston avulla (KUVIO 3).



KUVIO 3. B2B-ohjelmistojen integroinnissa yritysten tietojärjestelmät yhdistetään erillisen B2B-väliohjelmiston avulla.

2.3 Integraation periaatteet

Edellä käsiteltiin yleisellä tasolla yritysten liiketoiminnan tarpeita järjestelmille, järjestelmäintegraation tuomia hyötyjä liiketoiminnalla ja luotiin pintapuolinen katsaus käsitteellisellä tasolla, mitä järjestelmäintegraatio on.

Tässä luvussa luodaan konkreettisempi kuva, mitkä ovat ne peruskomponentit ja toiminnalliset kokonaisuudet, joista järjestelmäintegraatioratkaisut koostuvat. Luku perustuu pääasiassa Tähtisen (2005) käsityksiin järjestelmäintegraation periaatteista.

2.3.1 Mitä on järjestelmäintegraatio?

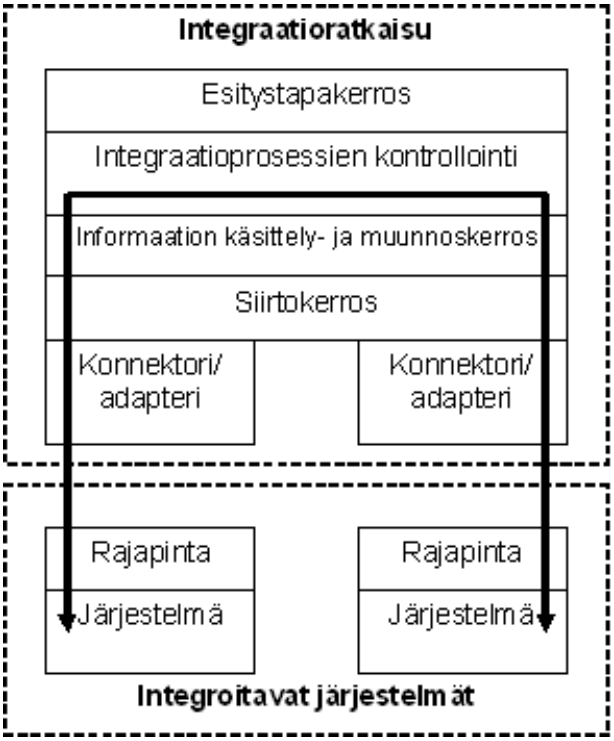
Järjestelmäintegraatio määriteltiin tässä tutkielmassa aiemmin tavaksi ja tekniikaksi, jolla automatisoidaan erityyppisten tietojärjestelmien vuoropuhelua. Tämä suppea määritelmä riittää kertomaan integraation perustarpeesta, mutta se ei kerro, miksi integraatiota tehdään ja mitä hyötyjä siitä on.

Tähtinen (2005) on määrittänyt järjestelmäintegraation toimintamalleiksi ja tekniikoiksi, joiden avulla voidaan saattaa vähintään kaksi eri toiminnallisuutta tarjoavaa tietojärjestelmää jakamaan informaatiota siten, että informaation siirto ja muunnokset ovat kontrolloitavissa ja monitoroitavissa yhdestä tai useammasta keskitetystä pisteestä.

Järjestelmäintegraatiossa on yksinkertaisimmillaan kysymys: (Tähtinen 2005, 48)

- informaation siirtämisestä integroitavien järjestelmien välillä
- tiedonmuunnoksista näiden järjestelmien sisäisten esitysmuotojen välillä
- kokonaisprosessin kontrolloinnista sekä tähän liittyvästä valvonnasta ja raportoinnista.

Tähtinen (2005) on esittänyt kuvion 4 mukaisen arkkitehtuurimallin, jossa arkkitehtuuri on jaettu integroitaviin järjestelmiin sekä integraatioratkaisuun. Seuraavaksi käydään läpi, mitä nämä eri osa-alueet tarkoittavat.



KUVIO 4. Integraatioarkkitehtuurimalli (Tähtinen 2005, 72).

2.3.2 Rajapinnat ja siirtokerros

Integroitavien järjestelmien on tarjottava jonkinlaiset rajapinnat, joiden kautta järjestelmistä voidaan hakea ja järjestelmiin voidaan syöttää informaatiota, jotta informaation siirto järjestelmien välillä on mahdollista. Nykyaikaisissa sovelluksissa nämä rajapinnat ja niiden käyttämät tekniikat vaihtelevat yksinkertaisista siirtotiedostoista moderneihin sanomapohjaisiin rajapintoihin. (Tähtinen 2005, 5.)

Tyypillisessä hajautetussa järjestelmässä eri osakomponenttien rajapinnat ovat vakioituja, integraatioratkaisussa rajapinnat muuttuvat jokaisen eri järjestelmän ja jokaisen eri yrityksen välillä. Ajateltaessa integraatioratkaisua yhtenä toiminnallisena kokonaisuutena, integroitavien järjestelmien rajapintoja vastaan asettuvat erityiset integraatioratkaisun rajapintakomponentit, joista käytetään nimityksiä liitin, sovitin tai agentti. (Tähtinen 2005, 71.)

Järjestelmien välillä täytyy olla jokin fyysinen siirtotie, jonka avulla informaatiota kuljetetaan. Mediana voivat toimia optiset tai magneettiset tallennusvälineet, mutta yleisemmin käytetään hyväksi tietoverkkoja, jotka mahdollistavat

nopeatempoisen tiedonsiirron automatisoinnin. Informaation siirtoon käytetään usein jotain TCP/ IP-pohjaista siirtotapaa ja sen päällä toimivaa etäkutsu- tai sanomansiirtoarkkitehtuuria. (Tähtinen 2005, 51-52.)

2.3.3 Informaation tulkinta ja muunnokset

Sovellusten keskeinen tehtävä on käsitellä, jalostaa ja esittää informaatiota. Näiden tehtävien toteutus vaihtelee sovelluksittain riippuen käytettävästä teknologiasta, teknisistä rajoitteista ja määräyksistä, ohjelmointitottumuksista ja muista lukuisista seikoista. Sama informaatio voidaan esittää erilaisten tietorakenteiden avulla.

Järjestelmän informaation sisäinen rakenne heijastuu ohjelmistoja integroitaessa myös ulospäin (Tähtinen 2005, 55). Mikäli tieto esitetään järjestelmässä puurakenteena, todennäköisesti myös ulkoinen rajapinta heijastelee tätä rakennetta. Automatisoidulle tiedonsiirrolle sovellusten väliset informaation esitystapojen eroavaisuudet voivat olla ongelmallisia, mikäli informaatiota ei välillä muunneta sovellusten ymmärtämästä esitysmuodosta toiseen.

Järjestelmäintegraation tavoitteena on yleiskäyttöinen ja yhtenäinen integraatioratkaisu, joka mahdollistaa periaatteessa minkä tahansa ohjelmistoparin välisen katkottoman kommunikaation. Tällöin tarvitaan informaation käsittely- ja muuntokerros, jonka kautta järjestelmien välinen informaatio kulkee. Tämän käsittely- ja muuntokerroksen tehtävä on kahtalainen. Ensiksi sen tulee tulkita lähettävän järjestelmän muodostamaa informaatiota ja toiseksi muodostaa tästä informaatiosta vastaanottavan järjestelmän ymmärtämä kokonaisuus. Tämä edellyttää, että kerros ymmärtää molempien järjestelmien tavat käsitellä informaatiota. (Tähtinen 2005, 57.)

2.3.4 Informaation siirron kontrollointi

Integraatioratkaisuissa tärkein asia on tiedonsiirron ja tietomuunnosten tehokas hallinta. Tiedonsiirto ja tietomuunnokset eivät tapahdu spontaanisti, vaan käytettävissä tulee olla hallinnointiympäristö, jossa voidaan keskitetysti sekä ymmärtää että hallita informaation siirtymistä järjestelmien välillä. (Tähtinen 2005, 59).

Integraatiotapahtumat ovat prosessimaisia, jotka alkavat tietystä herätteestä, koostuvat toisiaan seuraavista ennalta määrätyistä tapahtumista ja lopulta päättyvät. Näitä prosesseja hallitaan integroitavien järjestelmien kontrollointikerroksessa. Toisin sanoen yksittäiset integraatiotapahtumat ovat kontrolloivan kerroksen välittämiä kutsuja alla oleviin kerroksiin – siirtokerrokseen ja käsittely- ja muunnoskerrokseen. (Tähtinen 2005, 59).

Tähtinen (2005) esittää liiketoimintaprosessin rinnalle erillisen integraatioprosessin käsitteen. Liiketoimintaprosessi kuvaa tavat, joiden mukaan yritys toimii, ja integraatioprosessi kuvaa, kuinka liiketoimintaprosessit käytännössä toteutetaan. Toisin sanoen integraatioprosessi on liiketoimintaprosessin tai sen osan tekninen ilmentymä.

Tarkemmin määriteltynä integraatioprosessi on seuraavanlainen (Tähtinen 2005, 62-63):

1. Integraatioprosessi on sarja toimintoja, joiden tarkoituksena on siirtää informaatiota järjestelmien välillä ja tarvittaessa tulkita tätä informaatiota ja tehdä tietomuunnoksia.

2. Integraatioprosessit alkavat spontaanisti tai ulkoisesta herätteestä.

3. Integraatioprosessi toimii normaalitilanteessa automaattisesti ilman käyttäjän aktiivista puuttumista prosessin suoritukseen.

2.3.5 Esitystapakerros

Esitystapakerros on integraatioratkaisun ylin kerros, jonka avulla yrityksen johto ja työntekijät saavat tietoa integraatioprosessin ja tätä kautta koko yrityksen liiketoiminnan tilasta. Nämä käyttöliittymät voivat olla yksinkertaisimmillaan sähköpostitse lähetettäviä raportteja, mutta myös erilaisia portaaleja, joiden avulla saadaan tietoa prosessien tilasta ja voidaan kontrolloida alla olevia järjestelmiä. (Tähtinen 2005, 71.)

2.4 Ohjelmistojen integrointilähestymistavat

Ohjelmistojen integroinnissa on selkeä trendi siirtyä informaatiokeskeisestä integroinnista palvelukeskeiseen integrointiin. Informaatiokeskeinen integrointi on halpa lähestymistapa, koska lähestymistapaa käytettäessä harvoin tarvitsee

muuttaa ohjelmistoja. Pitkällä aikavälillä kuitenkin ohjelmistojen palveluiden ja metodien integrointi tarjoaa enemmän arvoa. (Linthicum 2004, 4-5.)

Ohjelmistojen integrointi on erilaisten ongelmakenttien yhdistelemistä. Jokaisella osapuolella on omat tarpeensa ja erityisalueensa, jotka tulee ottaa huomioon integroinnissa. Tästä johtuen lähestymistavat voivat vaihdella hyvinkin paljon. Linthicum (2004) on kategorisoinut neljä erilaista lähestymistapaa:

- informaatiokeskeinen lähestymistapa
- liiketoimintaprosessikeskeinen lähestymistapa
- portaalikeskeinen lähestymistapa
- palvelukeskeinen lähestymistapa.

Eri järjestelmien valmiudet integrointiin voivat vaihdella hyvinkin laajasti, joten yksittäinen lähestymistapa ei välttämättä riitä tehokkaan integraatioratkaisun saamiseksi. Juric, Basha, Leander ja Nagappan (2001) näkevätkin, että näitä kaikkia lähestymistapoja tarvitaan kerroksittain tuottamaan tietoa ylemmäntason integraatiolle.

2.4.1 Informaatiokeskeinen lähestymistapa

Informaatiokeskeisessä integrointilähestymistavassa tietokannat ja informaatiota tuottavat sovellusliittymät nähdään integroinnin pääkohtina. Lähestymistapa keskittyy tiedon siirtoon eri järjestelmien välillä. Tavoitteena on, että järjestelmät jakavat ja käyttävät samaa tietoa. (Juric ym. 2001, 80.) Integraatioratkaisut voidaan jakaa kolmeen kategoriaan: tiedon toisintamiseen (*replication*), tiedon liittämiseen, sovellusliittymien (*Application Programming Interface, API*) hyödyntämiseen (Linthicum 2004, 6).

Tietokantojen toisintaminen on yksinkertaisesti tiedon siirtoa kahden tai useamman tietokannan välillä. Perusvaatimus toisintamisessa on, että siinä otetaan huomioon mallien ja kaavojen eroavaisuudet tietokantojen välillä tarjoamalla perusrakenteet tietojen vaihdolle. Perusrakenne tarjotaan usein tietokantaorientoituneella väliohjelmistolla, joka sijoittuu tietokantojen väliin ja huolehtii tiedon muunnoksen ja välityksen eri tietokantojen välillä. (Linthicum 2004, 7.)

Tietokantojen toisintaminen on halpa ja yksinkertainen integrointitapa. Nämä edut katoavat kuitenkin nopeasti, mikäli tiedon lisäksi on tarve hyödyntää toiminnallisuutta tiedon rajaamiseen tai jakaa toiminnallisuuksia ohjelmistojen kesken. (Linthicum 2004, 7.)

Tietokantojen yhteenliittäminen tarkoittaa lukuisten tietokantojen ja tietokantamallien integrointia yhdeksi virtuaaliseksi näkymäksi fyysisistä tietokannoista. Tietokantojen liitos mahdollistaa pääsyn jokaiseen liitettyyn tietokantaan yhden hyvin määritellyn rajapinnan kautta. Tämä rajapinta on väliohjelmisto, joka sijoitetaan tietokantojen ja niiden tietoja käyttävän ohjelmiston väliin. (Linthicum 2004, 7.)

Toisin kuin tiedon toisintamisessa, tietokantojen yhteen liittämisessä ei tarvitse tehdä muutoksia lähdetietokantoihin. Muutokset tulee tehdä ainoastaan tietokantoja hyödyntäviin ohjelmistoihin tukemaan tätä yhtä virtuaalista tietokantaa. (Linthicum 2004, 7.)

Tietokantojen toisintamisen ja tiedon liittämisen haittana on, ettei niissä juurikaan hyödynnetä integroitavien ohjelmistojen liiketoimintalogiikkaa ja toimintoja, jotka voivat hyvinkin olla relevantteja integrointitoimenpiteissä. (Linthicum 2004, 10.)

Sovellusten rajapintoja ja etäproseduurikutsuja hyödyntävät ratkaisut käyttävät hyvin määriteltyä sovellusliittymää integroimaan paketoidut ohjelmistotuotteet ja tilaustyönä tehdyt ohjelmistot. Sovellusliittymä toimii sovittimena ohjelmistojen välillä muuntaen välitettävät tiedot oikeaan muotoon. Juric ym. (2001) puhuvat alemman tason virtuaalikomponenteista, jotka piilottavat järjestelmissä käytettävien eri tekniikoitten eroavaisuudet. Lähestymistavan etuna on sen tehokkuus yhdistää useita erityyppisiä ohjelmistoja. Sovellusliittymien käytöllä voidaan hyödyntää pelkän tiedon lisäksi myös eri järjestelmien toiminnallisuuksia (Juric ym. 2001, 90).

2.4.2 Liiketoimintaprosessikeskeinen integrointi

Yksinkertaistettuna liiketoimintaprosessikeskeinen ohjelmistojen integrointi (*Business Process Integration-Oriented Application Integration, BPIOAI*) tuottaa olemassa olevien yrityssovellusten liiketoimintaprosessien päälle kerroksen, joka sisältää keskitetysti hallittavan ja hyvin määritellyn liiketoimintaprosessien

joukon, jonka kautta on pääsy etäsovellusten tietoihin ja prosesseihin. Liiketoimintaprosessien integrointi (*Business Process Integration, BPI*) on mekanismi tiedon liikkumisen hallintaan, hallinnan tueksi prosessien herättäminen oikeassa ja tarkoituksenmukaisessa järjestyksessä ja prosessien suorittaminen sovellusten välillä ja sisällä. (Linthicum 2004, 10-11.)

Integraatioprosessit voivat vaatia lukuisten eri järjestelmän osien panosta, jolloin tarvitaan mekanismi, joka koordinoi näitä integraatioprosesseja. Hohpe ja Woolf (2004) näkevät jaettujen liiketoimintaprosessien suorittavan tällaiset tehtävät. Varsinaiset integraatioprosessit voidaan suorittaa toisintamalla tietokantoja tai toteuttamalla liiketoimintaprosessit palveluina.

Liiketoimintaprosessien integrointi tarjoaa tyypillisesti kolmenlaisia palveluja (Linthicum 2004, 13):

- liiketoimintaprosessien visualisoinnin
- rajapintojen abstrahoinnin
- reaaliaikaisen liiketoimintaprosessien suoritusten arvioinnin.

Lähestymistavassa tavoitteena on saada yhteen yrityksen tai yritysten relevantit prosessit sekä tukea informaatio- ja hallintalogiikkavirtaa näiden prosessien välillä. Tällöin välisovellus nähdään hyödykkeenä, joka tarjoaa helppokäyttöisen visuaalisen rajapinnan luomaan prosessiketjun, jolla automatisoidaan aiemmin manuaalisesti suoritetut tehtävät.

Konkreettisena tuloksena syntyy integrointiratkaisuun liiketoimintatason virtuaalisia komponentteja, jotka tarjoavat rajapinnat liiketoimintametodien ja -palveluiden suorittamiseen (Juric ym. 2001, 91). Tyypillisesti yksi tällainen korkeamman tason virtuaalinen komponentti kutsuu alemman tason virtuaalisia komponentteja, jotka ovat yhteydessä sovellusliittymien ja muiden rajapintojen kautta yrityksen eri järjestelmiin. Koko prosessiketjun toimivuuden takaamiseksi liiketoimintatason komponentit suorittavat myös tiedon prosessointia ja tietotyyppien muunnoksia, jotta tiedon siirto eri osajärjestelmien välillä olisi mahdollista.

Pelkkään informaatiokeskeiseen integraatioon verrattuna liiketoimintaprosessikeskeinen integrointi tarjoaa (Juric ym. 2001, 96)

- yhtenäisen liiketoimintaprosessin
- kaksisuuntaisen viestinnän
- välittömän operaatioiden käynnistämisen
- fyysisesti ja loogisesti yhdistetyn järjestelmän.

2.4.3 Portaalikeskeinen integraatio

Portaalikeskeinen tai esitystapakeskeinen lähestymistapa tarkoittaa yksinkertaisesti luotavaa käyttöliittymää, jonka kautta integroitua järjestelmää voidaan yhdestä paikkaa käyttää. Vaikka liiketoimintalogiikka ja tiedot olisivatkin yhteisessä käytössä eri osajärjestelmien välillä, näitä sovelluksia käytetään kuitenkin erillisinä sovelluksina tarpeen mukaan sovellusta vaihtaen. Tätä varten tarvitaan yhtenäinen käyttöliittymä, joka piilottaa käyttäjältä eri osajärjestelmien käytön.

Linthicum (2004) näkee portaalikeskeisen integraation vain koostettuna käyttöliittymänä eri sovelluksiin, ilman suoraa integraatiota eri osajärjestelmien välillä. Tällainen integraatioratkaisu ei yksistään ole kuitenkaan riittävä tehokkaan integraatioratkaisun aikaansaamiseksi. Juric ym. (2001) näkevätkin esitystapakeskeisen integraation välineenä yhden ja yhtenäisen käyttöliittymän toteuttamiseen integroituun järjestelmään. Esitystapakerroksena voivat toimia asiakassovellukset, joissa on graafinen käyttöliittymä, Web-asiakkaat tai universaalit asiakkaat, joissa käytetään erityyppisiä asiakasteknologioita (nettipohjaiset, mobiilit ym. käyttöliittymät) (Juric ym. 2001, 99).

2.4.4 Palvelukeskeinen integraatio

Tietokantojen toisintaminen ja yhdistäminen jakavat järjestelmien tietoa, muttei sovelluslogiikkaa. Sovellusliittymien ja etäproseduurikutsujen kautta voidaan jakaa järjestelmien sovelluslogiikkaa, mutta nämä ovat sidottu usein tiettyyn teknologiaan ja käyttöjärjestelmään. Jakamalla järjestelmien liiketoimintaprosessit etukäteen kuvattujen palveluiden ja järjestelmäriippumattomien viestien avulla, voidaan järjestelmän osat integroida toisistaan riippumattomasti.

Palvelukeskeinen sovellusten integrointi on terminä hyvin monitahoinen. Usein puhutaan myös palvelukeskeisestä arkkitehtuurista (*Service Oriented Architecture, SOA*). Käsitteet liittyvätkin hyvin kiinteästi toisiinsa, sillä palvelukeskeisen ohjelmistojen integroinnin lopputuloksena syntyy palvelukeskeinen arkkitehtuuri.

OASIS (2006) määrittelee palvelukeskeisen arkkitehtuurin paradigmana organisoida ja hyödyntää eri toimialueiden hajautettuja resursseja. Ort (2005) on määritellyt palvelukeskeisen ohjelmistojen integroinnin tavaksi jakaa organisaation sisäisiä (EAI) ja organisaatioiden välisiä (B2B AI) toimintoja (tyypillisesti liiketoiminnallisia toimintoja) laajasti sekä joustavasti.

W3C Working Group (2004) näkee palvelukeskeisen arkkitehtuurin hajautetun arkkitehtuurin muotona, jossa palveluja karakterisoivat seuraavat ominaisuudet:

- *Looginen näkymä*. Palvelu on käsitteellinen, looginen näkymä todellisista sovelluksista, tietokannoista, liiketoimintaprosesseista jne. ja se määritetään sen mukaan, mitä se tekee (tyypillisesti toteuttaa jonkin liiketoimintaprosessin).

- *Viestikeskeisyys*. Palvelu on muodollisesti määritetty palvelun tarjoajan ja pyytäjän välillä vaihdettavien viestien suhteen, ei osapuolten ominaisuuksien suhteen. Osapuolten sisäinen rakenne ja toiminnallisuus, kuten esimerkiksi toteutuskieli ja tietokantarakenne, on tarkoituksellisesti jätetty merkityksettömiksi seikoiksi.

- *Kuvauskeskeisyys*. Koneellisesti prosessoitava metatieto kuvaa palvelun. Kuvauksen tulee sisältää vain ne tiedot, jotka ovat julkisia ja palvelun käytön kannalta tärkeitä. Palvelun semantiikka tulee dokumentoida joko suorasti tai epäsuorasti kuvaukseen.

- *Rakeisuus*. Palveluissa on tyypillisesti vähän operaatioita, mutta joiden viestit ovat kuitenkin suhteellisen suuria ja monimutkaisia.

- *Verkkokeskeisyys*. Palveluita suoritetaan verkon välityksellä, joskin tämä ei ole ehdoton vaatimus.

- *Alustariippumattomuus*. Viestit lähetetään alustariippumattomassa, standardoidussa muodossa rajapintojen kautta. XML on sopivin muoto vastaamaan näihin vaatimuksiin.

Tähtinen (2005) näkee, että palvelukeskeisen ohjelmistojen integroinnin tarkoituksena on mahdollistaa eri ohjelmistojen tarjoamien palveluiden ja informaatiosisältöjen mahdollisimman joustava ja monipuolinen hyväksikäyttö yrityksen liiketoiminnan apuna.

Palvelukeskeinen ohjelmistojen integrointi ei ole uusi keksintö, vaan Sun määritteli termin ”palvelukeskeisen arkkitehtuuri” 1990-luvun lopussa kuvaamaan JINI-ympäristöä, jonka avulla voidaan dynaamisesti tarjota, löytää ja käyttää palveluita verkon välityksellä (Mahmoud 2003). JINI-ympäristön tavoitteena on joustava ja helposti hallittava palvelukeskeinen verkko, jossa palveluita voivat tarjota verkkoon kytketyt laitteet, sovellukset tai näiden yhdistelmät (Sun Microsystems 1999).

Palvelukeskeiseen arkkitehtuuriin pohjautuvat järjestelmät perustuvat nimensä mukaisesti palveluihin. *Palvelulla* tarkoitetaan johonkin liiketoimintaprosessiin kuuluvaa toistettavaa tehtävää. *Palvelukeskeisyys* tarkoittaa lähestymistapaa, jossa järjestelmän osat integroidaan toisiinsa yhdistettyjen palveluiden avulla. Lähestymistavassa järjestelmän osa voi olla joko palvelun tuottaja, käyttäjä tai molempia. Näiden määritysten pohjalta voidaan määrittää *palvelukeskeinen arkkitehtuuri* tyyliksi rakentaa yrityksen IT-arkkehtuuri palvelukeskeisen lähestymistavan periaatteiden mukaisesti. (High, Kinder & Graham 2005.)

Verrattuna aiempiin tapoihin lähestymistavassa on uutta järjestelmän eri osien löyhä kytkentä (*loose coupling*) (High, Kinder & Graham 2005, 8). Löyhä kytkentä tarkoittaa, että järjestelmän eri osat ovat itsenäisiä toisista järjestelmän osista, eivätkä ne ole riippuvaisia muiden järjestelmäosien toteutuksesta. Ydinajatuksena on vähentää osapuolten etukäteisoletuksia toisistaan (Hohpe ja Woolf 2004, 10). Palvelua käytetään ennalta määritetyn rajapinnan avulla ja palvelun kuluttajan ei tarvitse tietää, millä ohjelmointikielellä palvelu on toteutettu ja mitä sisäisiä toiminnallisuuksia palvelun toteuttamiseen tarvitaan. Tiukasti kytketyssä arkkitehtuurissa eri ohjelmistokomponentit on sidottu toisiinsa jaettujen kirjastojen, semantiikan ja usein myös tilan kautta. Tällöin on vaikeaa kehittää järjestelmää alituisesti vaihtuvien vaatimusten mukaisiksi. Tiukasti kytkettyjen

järjestelmien viestintä on tehokkaampaa, mutta löyhästi kytketyt järjestelmät ovat joustavampia ja herkempiä muutoksille.

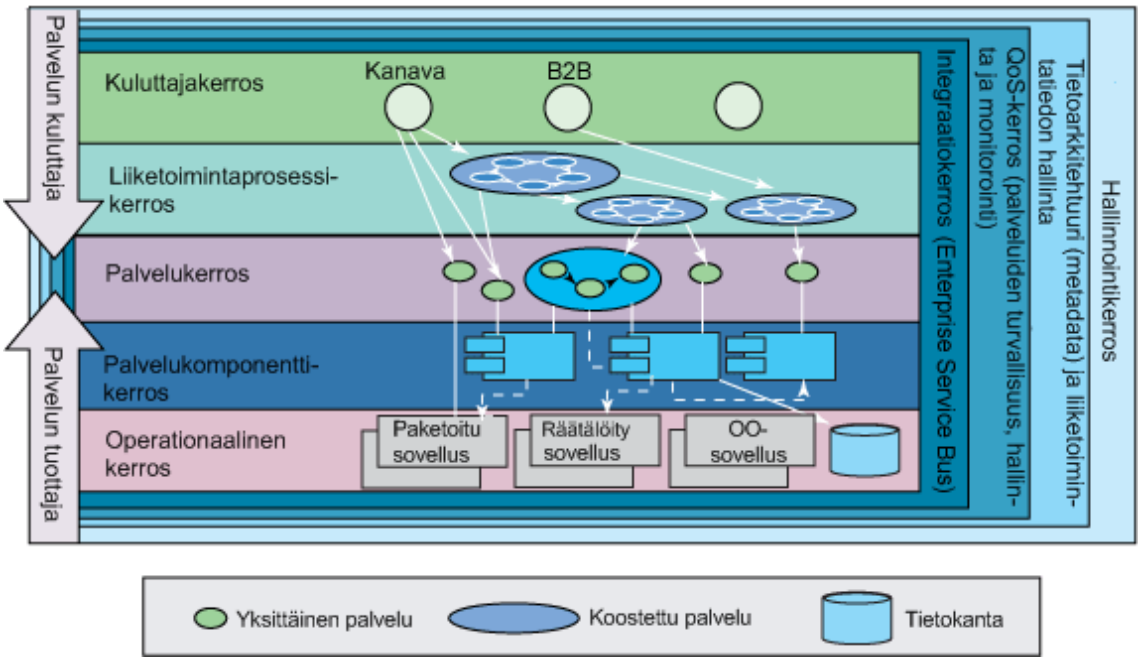
Palvelukeskeinen arkkitehtuuri ei kuitenkaan ole pelkästään teknologia, vaan se on ennen kaikkea menetelmien ja työkalujen yhdistelmä, joka auttaa suunnittelemaan yrityksen liiketoimintoja ja parantaa niiden toteutusta. Se tarjoaa menetelmät ja ohjelmointimallit, joiden avulla näitä liiketoimintamalleja voidaan toteuttaa yrityksen tietojärjestelmissä. Lisäksi se määrittää, kuka on vastuussa ja valtuutettu liiketoimintamallien suunnittelusta ja niiden toteutuksesta tietojärjestelmissä. (High, Kinder & Graham 2005, 10). Toisin sanoen palvelukeskeisen arkkitehtuurin avulla yritys voi kartoittaa, suunnitella ja toteuttaa liiketoimintamalleja, integroida tietojärjestelmät siten, että ne tukevat näitä liiketoimintamalleja ja hallita liiketoimintaprosessien muutoksia.

Kuviossa 5 on esitetty tyypillinen palvelukeskeisen arkkitehtuurin ratkaisumalli. Integraatiokerroksessa yrityksen liiketoimintaprosessit tunnistetaan ja niistä luodaan liiketoimintapalveluita (*Liiketoimintaprosessikerros*), joita kuluttajat (*Kuluttajakerros*) käyttävät. Palvelukerroksessa sijaitsevat palvelut hyödyntävät erityisiä palvelukomponentteja, jotka tarjoavat integroitavien järjestelmien resurssit laitteistoriippumattomassa muodossa. Järjestelmät voivat tarjota myös suoraan itse resurssejaan palveluina (*Operationaalinen kerros*) (vrt. luku 2.3.2 ja 2.3.3).

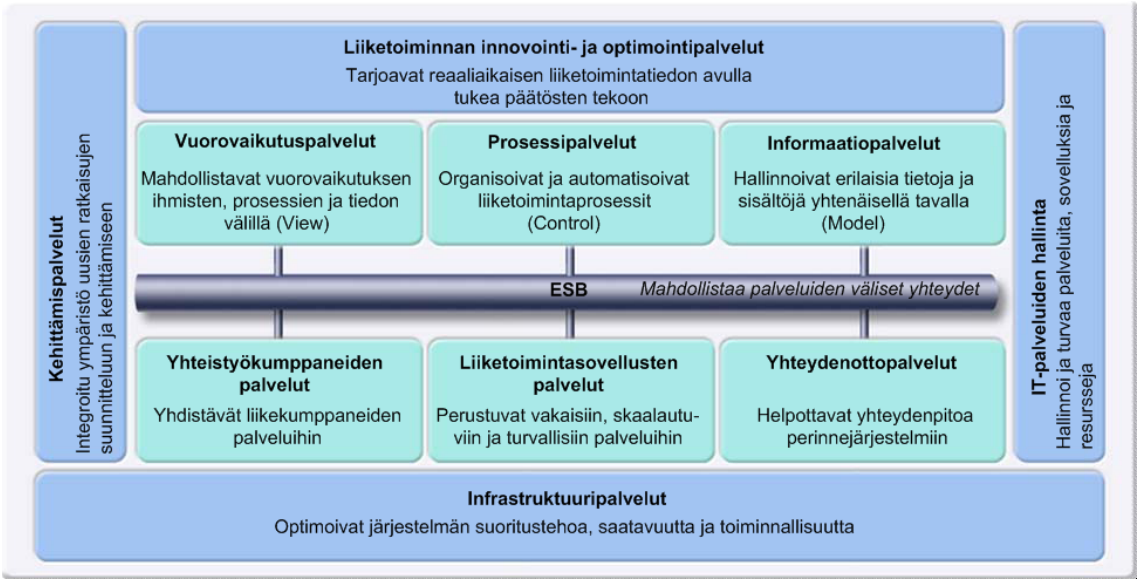
Integraatioratkaisussa tulee ottaa huomioon järjestelmän ei-toiminnalliset vaatimukset ja turvallisuustekijät sekä määrittää, mitkä ovat järjestelmän monitoroitavia tietovirtoja (*QoS-kerros*) (vrt. luku 2.3.4). Lisäksi tulee ottaa huomioon tietojärjestelmän tietoarkkitehtuuriset ja liiketoimintatiedon keräämisen vaatimukset (*Tietoarkkitehtuuri- ja liiketoimintatiedon hallinta*). Järjestelmän tulee lisäksi tukea yrityksen hallinnointimallia ja tarjota raportteja integraatioprosesseista (H*allinnointikerros*) (vrt. luku 2.3.5). Lopputuloksena on järjestelmän looginen arkkitehtuuri (KUVIO 6), jossa koko integraatioratkaisu käsitetään palveluiden kautta. (High, Kinder & Graham 2005, 25).

Kuvioissa 5 ja 6 on esillä myös keskeinen palvelukeskeisen arkkitehtuurin osa, Enterprise Service Bus (ESB). Sen sijaan, että ohjelmistot pyytäisivät palveluita toisiltaan, voivat ne pyytää palveluita keskitetysti ESB:lta. ESB piilottaa järjestelmän fyysisen arkkitehtuurin ja huolehtii pyydetyn toiminnon suorittamiseen tarvittavien palveluiden kutsumisesta. Vaikka markkinoilla on Enterprise Ser-

vice Bus -nimisiä tuotteita, ESB on ennen kaikkea arkkitehtuurinen malli (High, Kinder & Graham 2005, 30).



KUVIO 5. Palvelukeskeisen arkkitehtuurin ratkaisupino (Ibrahim & Long 2007).



KUVIO 6. Palvelukeskeisesti integroitavan järjestelmän looginen arkkitehtuuri (High, Kinder & Graham 2005, 25).

Palvelukeskeisen arkkitehtuurin tavoitteena on, että järjestelmät ja ohjelmistot voivat kommunikoida keskenään riippumatta laitealustasta, käyttöjärjestelmäs-

tä ja ohjelmointikielestä. Web Services -tekniikat tarjoavat tähän ratkaisun kypsien ja laajasti käytettyjen protokollien ja tekniikoiden avulla.

3 WEB SERVICES

Luvussa 2 sivuttiin lyhyesti Web Services -tekniikoita, mutta termiä ei määritelty sen enempää. Tässä luvussa selvitetään tarkemmin mitä Web Services -termillä tarkoittaan, mitä hyötyjä Web Services -tekniikoista on ja mitä eri osaalueita Web Services -tekniikkoihin kuuluu.

3.1 Web Services -tekniikat

Kuten edellisessä luvussa mainittiin, Web Services -tekniikat ovat palvelukeskeisen integraation toteutusvälineitä. W3C Web Services Architecture Working Group (2004) määrittelee Web Services -palvelun olevan

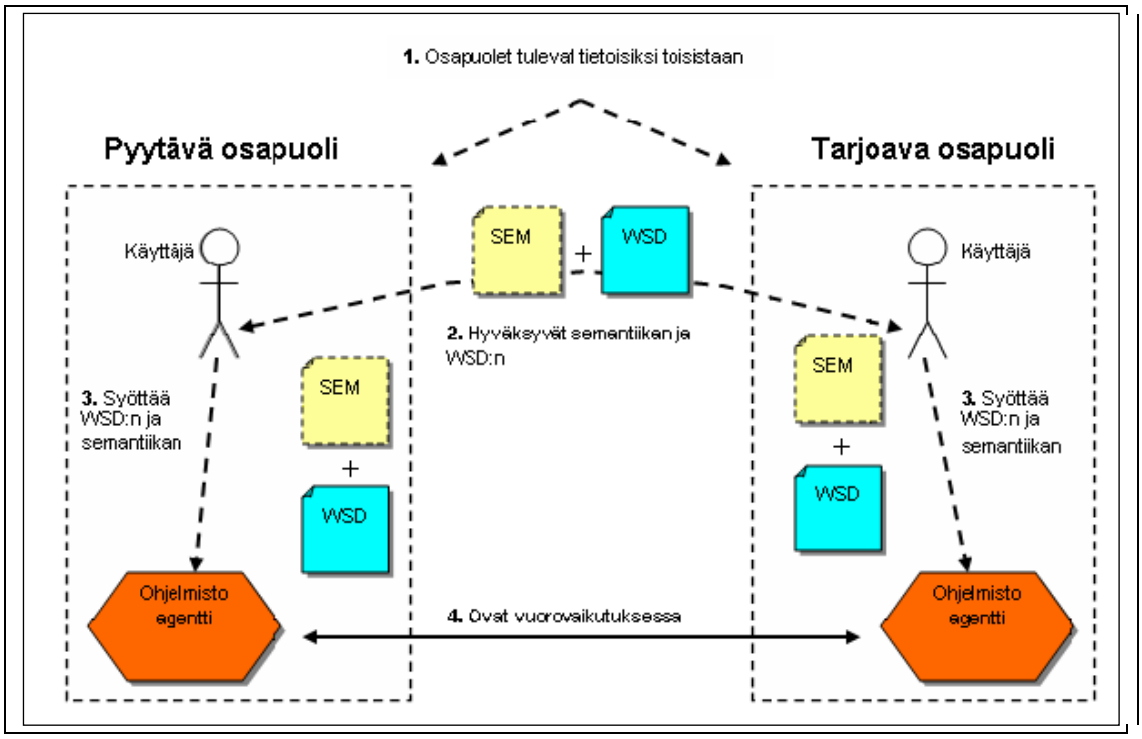
”URI:n yksilöimä sovellus, jonka julkiset rajapinnat ja sidonnaisuudet kuvataan ja määritetään XML:a käyttäen. Näiden määritysten puitteissa sovellukset kommunikoivat XML-pohjaisten viestein Internet-pohjaisia protokollia käyttäen.”.

Kirjallisuudessa on tiukennettu määrittelyä vaatimuksilla, että palvelun kuvaamisen käytetään WSDL-kuvauskieltä (Web Services Description Language) ja protokollana käytetään SOAP-protokollaa (Simple Object Access Protocol). (Ferris & Farrel 2003, 31). Internet-pohjaiset protokollat tarkoittavat minimissään TCP/ IP- tai UDP-protokollan käyttöä, mutta yleensä käytetään ja suositellaan käytettäväksi HTTP-protokollaa, joka toimii TCP-protokollan päällä (Burner 2003, 30; W3C Working Group 2007a). Kommunikointi sovellusten välillä tapahtuu viestien välityksellä. Viestit ovat riippumattomia, joten niiden tulee sisältää kaikki tarvittavat tiedot, jotka ovat tarpeellisia viestin sisällön ymmärtämiseksi (Burner 2003, 30).

Web Services -tekniikoiden käyttö edellyttää, että osapuolet ovat tietoisia toisistaan, sopivat vuorovaikutussäännöistä ja -muodoista sekä ovat kykeneviä olemaan vuorovaikutuksessa keskenään (KUVIO 7). Tätä menetelmää kutsutaan myös find-bind-execute-paradigmaksi (Mahmoud 2005). Paradigmassa palvelun tarjoajat rekisteröivät palvelunsa julkiseen rekisteriin, josta palvelun kuluttajat hakevat palveluita eri kriteerein. Mikäli haluttu palvelu löytyy, rekisteri tarjoaa kuluttajalle palvelun käyttöehdot ja palvelun osoitteen (*service endpoint*).

Web Services Interoperability Organisation edistää Web Service -tekniikoiden yhteensopivuutta. Organisaatio on julkaissut WS-I Basic Profile -suosituksen,

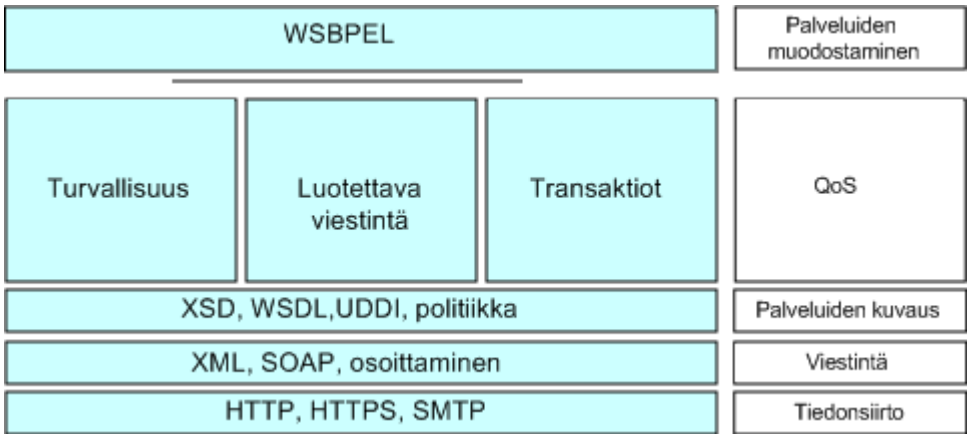
joka helpottaa entisestään eri tekniikoiden ja ohjelmointikielien yhteensopivuutta selventämällä, jalostamalla, tulkitsemalla ja vahvistamalla tiettyjä Web Services -spesifikaatioissa määritettyjä seikkoja (Web Services Interoperability Organization 2004).



KUVIO 7. Web Services -tekniikoiden käytön prosessi (W3C Working Group, 2004).

3.2 Ydinkomponentit

Web Services -tekniikoiden ydinkomponentit muodostavat W3C- ja OASIS-konsortioiden standardoimat XML, WSDL, SOAP ja UDDI. Nämä tekniikat mahdollistavat palveluiden kuvauksen, viestinnän osapuolten välillä sekä varsinaisen tiedonsiirron (KUVIO 8).



KUVIO 8. Palvelukeskeisen arkkitehtuurin mukainen Web Services –tekniikkapino (Ferguson, Storey, Lovering ja Swewchuck, 2003).

3.2.1 XML (eXtensible Markup Language)

XML-kieli on kuvauskieli, jota käytetään tiedon esittämiseen laitteistoriippumattomana ja itsensä kuvaavana tekstimuodossa. Kielen avulla esitetään dokumentin tietosisältö sekä tietosisällön rakenne käyttämällä XML-elementtejä ja -attribuutteja. Lisäksi täytyy olla jotkin sovitut säännöt, jotka vasta antavat merkityksen eri elementeille. Esimerkiksi mikäli ihminen (1) ymmärtää listauksen 1 <book>-elementin tarkoittavan kirjaa, <title>-, <author>-, ja <price>-elementtien tarkoittavan otsikkoa, tekijää ja hintaa sekä (2) käyttää näitä tunnuksia (*tags*) johdonmukaisesti, on tietojen vaihto mahdollista. Yleensä XML-dokumenttiin on liitetty erillinen skeema, jossa on määritetty, mitä tunnuksia dokumentissa on sallittu käyttää sekä näiden tunnusten rakenteet ja niihin liittyvät säännöt (Ort 2005).

LISTAUS 1. Yksinkertainen XML-dokumentti (Ort 2005).

```
<bookshelf>
   <book>
      <title>My Life and Times</title>
      <author>Felix Harrison</author>
```

```
      <price>39.95</price>
   </book>
</bookshelf>
```

Web Services -tekniikoissa elementit identifioivat tietotyyppinsä XML Schema Languagen (XSD) avulla. Tämän perusteella sovellus- ja laitealustat osaavat muuntaa tiedot alustakohtaisiksi tietotyypeiksi. (Burner 2003, 30.)

3.2.2 WSDL (Web Services Description Language)

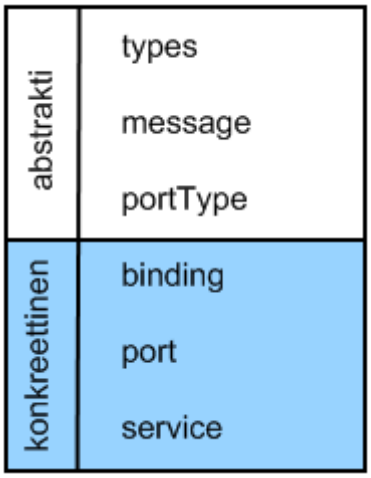
WSDL:n avulla voidaan määrittää palvelulle ne viestit, joita se ottaa vastaan ja joita se lähettää (Burner 2003, 30). Tosin sanoen siinä määritetään rajapinnat, joiden kautta palvelua voidaan käyttää. WSDL-muotoisessa WSD-dokumentissa (Web Servicess Description) (tai yleisemmin WSDL-dokumentissa) määritetään myös, kuinka viestit tulee lähettää ja missä verkko-osoitteessa palvelu sijaitsee (Ferris & Farrel 2003, 31).

Yksinkertaistettuna WSDL-dokumentti määrittää XML-skeeman, jolla Web Service -palvelu kuvataan. Dokumentissa määritetään palvelut päätepisteiden (*endpoints*) ja porttien (*ports*) joukkona. WSDL:ssa päätepisteiden ja viestien abstraktit määritykset erotetaan niiden todellisista käyttöönotoista verkossa ja tietotyyppien sidonnoista (KUVIO 9). Tämän johdosta abstrakteja määrityksiä voidaan käyttää uudelleen. WSDL-dokumentti käyttää seuraavia elementtejä palveluiden määrittämiseen (Hirsch, Kemp ja Ilkka 2006, 37):

- types-elementti määrittää viesteissä käytettävät XML-tietotyypit
- message-elementti määrittää yksisuuntaisen operaation (kutsu- tai vastausviestin) ja sen mukana kuljetettavan types-elementin
- portType-elementti määrittää abstraktilla tasolla operaatioissa vaihdettavat viestit.
- binding-elementti määrittää yksityiskohdat (tiedonsiirtoprotokollan ja viestien muodostamistyylin) portType-elementissä kuvattujen viestien vaihtoon

- port-elementti määrittää palvelun konkreettisen päätepisteen (*endpoint*), sen osoitteen ja binding-elementin, jossa on määritetty kutsuttavat operaatiot.

- service-elementti määrittää päätepisteitten joukon, joiden kautta port- ja portType-elementeissä kuvatut operaatiot ovat käytettävissä.



KUVIO 9. WSDL-dokumentin elementit (Hirsch, Kemp ja Ilkka 2006, 38).

Esimerkin mukainen WSDL-dokumentti (LISTAUS 2) määrittää pörssikurssipalvelun. Palvelulle on määritetty yksi operaatio GetLastTradePrice. Syöteviesti, GetLastTradeInput, on TradePriceRequest-muotoa ja se pitää sisällään merkkijonon tickerSymbol. StockQuoteSoapBinding määrittää viestinnässä käytettävän tyylin. Esimerkissä käytetään Document-tyylistä viestintää. Vaihtoehtona olisi käyttää RPC-tyyliä. Document-tyylisessä viestinnässä vaihdetaan operaatiokutsun yhteydessä koko XML-dokumentti, jolloin todentaminen ja liiketoimintasääntöjen hyväksyminen on helpompaa (Ort 2005).

LISTAUS 2. Esimerkki WSDL-dokumentti (W3C Working Group 2001).

```
<?xml version="1.0"?>
<definitions name="StockQuote"
targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd1="http://example.com/stockquote.xsd"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

    <types>
       <schema targetNamespace="http://example.com/stockquote.xsd"
              xmlns="http://www.w3.org/2000/10/XMLSchema">
           <element name="TradePriceRequest">
              <complexType>
```

```
                <all>
                    <element name="tickerSymbol" type="string"/>
                </all>
            </complexType>
        </element>
        <element name="TradePrice">
            <complexType>
                <all>
                    <element name="price" type="float"/>
                </all>
            </complexType>
        </element>
    </schema>
</types>

<message name="GetLastTradePriceInput">
    <part name="body" element="xsd1:TradePriceRequest"/>
</message>

<message name="GetLastTradePriceOutput">
    <part name="body" element="xsd1:TradePrice"/>
</message>

<portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
        <input message="tns:GetLastTradePriceInput"/>
        <output message="tns:GetLastTradePriceOutput"/>
    </operation>
</portType>

<binding name="StockQuoteSoapBinding"
type="tns:StockQuotePortType">
    <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetLastTradePrice">
        <soap:operation
soapAction="http://example.com/GetLastTradePrice"/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
</binding>

<service name="StockQuoteService">
    <documentation>My first service</documentation>
    <port name="StockQuotePort" binding="tns:StockQuoteBinding">
        <soap:address location="http://example.com/stockquote"/>
    </port>
</service>
</definitions>
```

3.2.3 SOAP (Simple Object Access Protocol)

Luvussa 3.2.1 kävi ilmi, että XML on sopiva väline verkossa välitettävän tiedon kuvaamiseen. Tehokkaan tiedonvaihdon takaamiseksi tarvitaan lisäksi joukko sääntöjä ja protokollia, jotta osapuolet ymmärtävät viestien sisällön samalla tavoin. SOAP on yksinkertainen etäkutsuprotokolla, jossa tiedon sisäinen esitysmuoto on paketoitu XML-muotoon. SOAP koostuu kolmesta pääkomponentista: pakollisesta SOAP-kirjekuoresta, valinnaisista SOAP-otsikkotiedoista sekä pakollisesta SOAP-viestistä (Ort 2005).

SOAP-kirjekuori sisältää tietoa käytettävästä nimiavaruudesta ja koodauksesta. Nimiavaruuden määrittämisellä ehkäistään konfliktit, joissa eri tietoja käsitellään samalla nimellä. Tietotyyppien koodauksella XML-skeemoja käyttäen voidaan määritellä uusia tietotyyppejä ja käyttää niitä SOAP-viestin osina. (McLaughlin 2002, 361; Juric ym. 2001,845; Ort 2005.)

SOAP-otsikkotiedot (*header*) laajentavat SOAP-viestiä modulaarisesti. Viestiä välittävät tahot voivat matkan varrella otsikkotietojen perusteella tehdä eri toimenpiteitä, esimerkiksi muuntaa tietoja tai suorittaa tietoturvaan liittyviä proseduureja. Viestin runko-osa (*body*) sisältää viestin tärkeimmän osan ja se on tarkoitettu ainoastaan viestin vastaanottajalle.

Listauksessa 3 viestin otsikkotiedoissa on yksi standardiin kuulumaton lokaali elementti *alertcontrol*, jossa määritetään prioriteetti ja kelpoisuusaika. Tämän tiedon perusteella viestin eri välityskerrokset voivat esimerkiksi priorisoida viestin jatkolähetystä. Viestin runko-osassa on myös yksi lokaali elementti *alert*, joka sisältää varsinaisen muistutusviestin.

SOAP Messages with Attachments -spesifikaation mukaisen SOAP-viestin mukana voi olla myös liitetiedostoja, kuten esimerkiksi kuvia (W3C Working Group 2000).

LISTAUS 3. Yksinkertainen SOAP-viesti (W3C Working Group 2007b).

```
<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>
```

3.2.4 UDDI (Universal Description, Discovery and Integration)

UDDI on Web Services -palveluita luetteloiva rekisteri, josta voi etsiä palveluita tarjoavia organisaatioita, niiden tarjoamia palveluita sekä teknisiä rajapintoja, joiden kautta näitä palveluita voidaan käyttää (OASIS 2004). Palvelun etsijä saa UDDI:n kautta WSD:n, jonka perusteella asiakassovellus toteutetaan tai konfiguroidaan (Ferris ja Farrel 2003, 31). Kiteytettynä nämä perusmääritelmät toisiinsa: UDDI auttaa löytämään tietyntyyppisen palvelun, WSDL kuvaa tämän palvelun ja SOAP mahdollistaa palvelun kutsumisen (Tähtinen 2005, 119).

3.3 Lisäkomponentit

Pelkästään Web Services -tekniikoiden ydinkomponentteja käyttämällä palvelukeskeisen arkkitehtuurin toteutus ei ole mahdollista palveluiden muodostamisen, viestinnän ja palveluiden laadunvarmistuksen (*Quality of Services, QoS*) osalta (Keen, Acharaya, Bishop ym. 2004, 86) (ks. KUVIO 8). Eri osapuolet ovat kehittäneet tekniikoita kattamaan nämä alueet. Näitä tekniikoita kutsutaan yleisesti WS-*-spesifikaatioiksi. Palveluiden yhteensopivuuden parantamiseksi W3C- ja OASIS-organisaatiot ovat muodostaneet näistä tekniikoista standardeja ja suosituksia. Lisäksi Liberty Alliance on kehittänyt tekniikoita identiteettien varmistamiseen ja hallintaan. Seuraavaksi käsitellään joitakin näistä tekniikoista tarkemmin.

3.3.1 WS-BPEL

Liiketoimintaprosessien toteutukset saattavat usein sisältää useita eri askelia ennen kuin prosessi on suoritettu. Nämä askeleet toteutetaan erillisten palveluiden avulla. Tällaisissa tapauksissa tarvitaan jokin keino, jolla liiketoimintaprosessi ja sen aikana suoritettavat palvelut kuvataan. WS-BPEL on XML-pohjainen kieli, jonka avulla kuvataan liiketoimintaprosessit ja niiden suorittamiseen tarvittavat Web Services -palvelut (OASIS 2007a; Ort 2005).

3.3.2 WS-Security

WS-Security tarjoaa laajennoksen SOAP-protokollaan, jonka avulla voidaan toteuttaa viestien sisällön koskemattomuus ja luottamuksellisuus tietoturvallisten Web Services -palveluiden toteuttamisen yhteydessä (OASIS 2006) (KUVIO 10). Koskemattomuudella tarkoitetaan, ettei viestin sisältöä ole muutettu matkan varrelta asiakkaalta palvelimelle. Luottamuksellisuudella tarkoitetaan, että viestin näkevät vain sille määritetyt vastaanottajat. (Ort 2005.)

WS-Security ei varsinaisesti määritä uusia turvallisuusmalleja, vaan se mahdollistaa olemassa olevien mallien ja tekniikoiden, esimerkiksi PKI-arkkitehtuuri (Public Key Infrastructure), Kerberos sekä SSL/ TLS-protokollat, yhteensopivan käytön Web Services -palveluissa (Ort 2005; Ferguson ym. 2003).

3.3.3 WS-Trust

WS-Trust tarjoaa laajennoksen WS-Security-spesifikaatioon. Laajennos määrittää menetelmät turvallisuusvaltuuksien (*security tokens*) jakamiseen, uusimiseen ja kelpoisuuden tutkimiseen sekä tavat luoda ja varmentaa luottamuksellinen yhteys (OASIS 2007b).

3.3.4 WS-ReliableMessaging

Tietoverkot eivät koskaan ole täysin luotettavia, vaan yhteydet saattavat katketa ja viestit kadota matkan varrella. WS-RelibleMessaging määrittää mekanismin, jonka avulla Web Services -palvelut voivat varmistua viestien perille menosta (Ferguson ym. 2003) (ks. KUVIO 10). Spesifikaatio määrittää viestitysprotokollan, jolla tunnistetaan, jäljitetään ja hallitaan viestien luotettavaa välitystä lähde- ja kohdesijainnin välillä (OASIS 2007c).

3.3.5 WS-Addressing

Käytettäessä Web Services -palveluissa HTTP-protokollaa, viestin lähettäjän ja vastaanottajan tiedot kulkevat HTTP-protokollan otsikkotiedoissa. Varsinaisessa SOAP-viestissä näitä tietoja ei ole lainkaan, jolloin käytettäessä välityspalvelimia tai yhteyden katkeamisen yhteydessä tämä tieto voi muuttua tai puuttua kokonaan. On myös mahdollista, että vastausviestin vastaanottajaksi halutaan

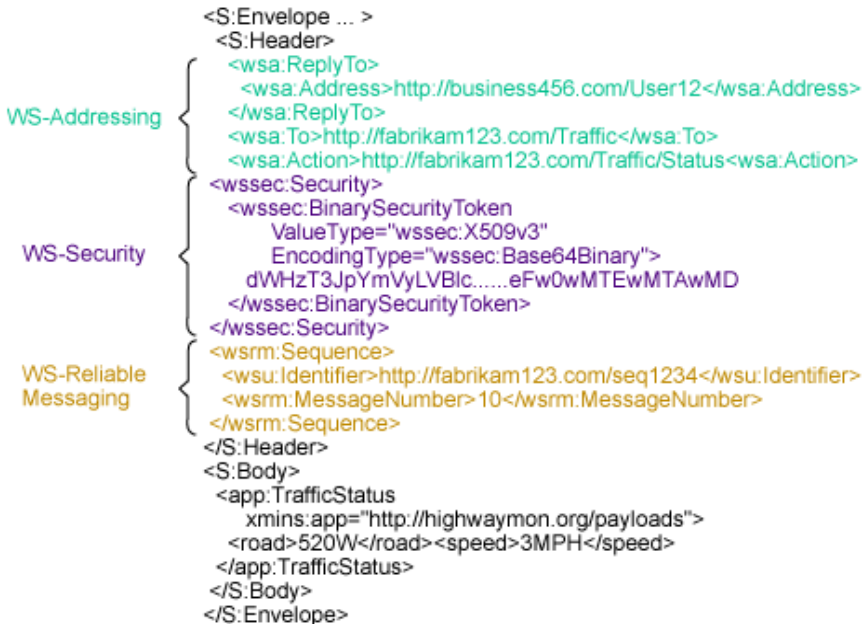
täysin toinen taho kuin alkuperäinen viestin lähettäjä. WS-Addressing tarjoaa toimivan ja riippumattoman tavan identifioida viestin lähettäjä ja vastaanottaja (W3C Working Group 2006; Ferguson ym. 2003) (KUVIO 10).

3.3.6 WS-Coordination

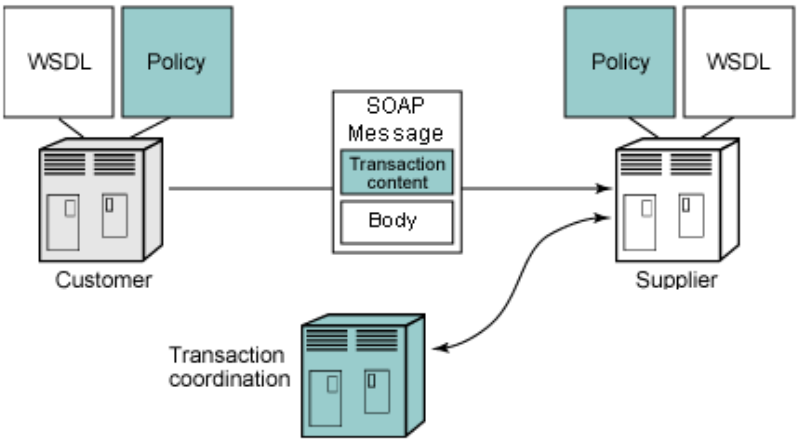
Web Services -palvelut yhä enenevissä määrin sitovat yhteen useita osapuolia ja ovat rakenteeltaan monimutkaisia. Palveluiden suorittaminen saattaa vaatia useiden viestien välitystä osapuolten kesken. WS-Coordination määrittää laajennettavan kehyksen näiden aktiviteettien suorittamiseen erityisen koordinoijan ja koordinointi protokollien avulla (OASIS 2007d) (KUVIO 11).

3.3.7 WS-Policy

WSDL-dokumentti määrittää palvelun rajapinnan syntaksin, mutta siitä ei käy ilmi, kuinka palvelu jakaa rajapinnan ja mitä palvelu odottaa kutsujalta, esimerkiksi edellyttääkö palvelu jotain turvallisuusmallia tai tukeeko se transaktioita. WS-Policy mahdollistaa tällaisten tietojen määrittämisen palveluille (KUVIO 11). (Ferguson ym. 2003).



KUVIO 10. Turvallinen ja luotettava SOAP-viesti (Ferguson ym. 2003). Vrt. LISTAUS 3.



KUVIO 11. Transaktioiden koordinointi ja politiikka Web Services -palvelussa (Ferguson 2003).

3.3.8 Identiteetin hallinta

Eri osapuolten identiteetistä varmistuminen on kriittistä Web Services -palveluille. Palvelukohtaiset identiteetinhallinta- ja tietoturvaratkaisut palvelut sitovat tiettyyn teknologiaan ja palveluntarjoajaan, mikä heikentää palveluiden yhteensopivuutta ja käyttäjäkokemusta. (Hirsch, Kemp ja Ilkka 2006, 103.) Liberty Alliance -organisaatio tarjoaa spesifikaatiot identiteetin hallintaan eri järjestelmien välillä (*federated identity*) sekä erityisten identiteettipalveluiden (*identity services*) hyödyntämiseen (Liberty Alliance Project 2008).

Liberty Identity Federation Framework (ID-FF) määrittää arkkitehtuurit identiteettien jakamiseen eri järjestelmien kesken, kertakirjautumiseen (*single sign-on)* ja -uloskirjautumiseen (*single logout)*. Liberty Identity Web Services Framework (ID-WSF) määrittää järjestelmän, joka mahdollistaa identiteettipalveluiden (*Identity Web Services*) käytön. Liberty Identity Services Interface Specifications (ID-SIS) kuvaa useita käyttötapauksia identiteettipalveluiden hyödyntämiseen. (Hirsch, Kemp ja Ilkka 2006, 104).

3.4 Web Services -tekniikoiden hyödyt

Web Services -tekniikoiden hyötyjä voidaan tarkastella sekä integroitavien järjestelmien että integraatioprosessin näkökulmasta.

Integroitavien järjestelmien näkökulmasta tarkasteltaessa suurin hyöty on, että Web Services -palvelut perustuvat itsensä kuvaaviin XML-sanomiin, joten teknologian käyttö on alustariippumatonta (Tähtinen 2005, 119). Tosin sanoen palveluntarjoaja ja asiakassovellus voivat olla ohjelmoitu eri ohjelmointikielillä (rakenteellisella ja olio-ohjelmointikielellä), ne voivat toimia eri käyttöjärjestelmissä sekä sovelluksen suorittavat laitteet voivat olla aivan erilaisia, esimerkiksi matkapuhelin ja järeä sovelluspalvelin.

SOAP-viestit välitetään yleisimmin HTTP-protokollan välityksellä, jolloin tieto reititetään saman tietoliikenneportin lävitse kuin verkkoselaintenkin liikenne. Tällöin vältytään yrityksen tietoturvapolitiikan kannalta ongelmallisten TCP/ IP-porttien käytöltä. Lisäksi SOAP-protokolla on yksinkertainen ja kevyt käyttää, eikä sen pitäisi vaatia suuria prosessointitehoja laitteilta. (Tähtien 2005, 119; McLaughlin 2002, 360-362.)

Tarkasteltaessa koko integraatioprosessia Hogg, Chilcott, Nolan ja Srinivasan (2004) ovat luetelleet Web Services -tekniikoiden käytön hyödyiksi seuraavat asiat:

- Tekniikat mahdollistavat yrityksen sisäisen (EAI) ja yritysten välisen (B2BAI) ohjelmistojen integroinnin.
- Järjestelmien tarjoamat resurssit ovat paremmin saatavilla (*access enablement*).
- Tekniikoiden avulla voidaan luoda joustavia ja uudelleenkäytettäviä ohjelmistokomponentteja.
- Tekniikat mahdollistavat ohjelmistojen evolutionaariseen ja inkrementaaliseen käyttöönottoon, jolloin ei tarvita muutoksia olemassa oleviin järjestelmiin.
- Laitteistoriippumattomuuden ansiosta perinnejärjestelmien resurssien hyödyntäminen on mahdollista.
- Järjestelmäintegraation vaatima työmäärä vähenee.
- Tekniikat mahdollistavat myös olemassa olevien toiminnallisuuksien laajentamisen.

3.5 Web Services -tekniikoiden heikkoudet

HTTP-protokollan käytössä on omat hyötynsä, mutta tietyiltä osin se ei ole paras mahdollinen vaihtoehto. Alun perin protokolla suunniteltiin tilattomaan synkroniseen dokumenttien hakemiseen verkkoselaimella. Protokollana HTTP ei ole kovin luotettava sovellusten välisessä viestinnässä. (Endrei, Ang, Arsanjani ym. 2004, 111; Hohpe ja Woolf 2004, 373.) Tästä johtuen Web Services -palveluissa viestien asynkronisuutta sekä luotettavaa perille menoa ei voida toteuttaa tiedonsiirtotasolla, vaan nämä tulee huolehtia jollain toisella tavalla.

Lukuisia Web Services -spesifikaatioita[1] on kehitetty eri osapuolten toimesta osittain päällekkäinkin. Tämä on osaltaan hidastanut tekniikoiden kypsymistä ja laajamittaista hyväksyntää. Jotta Web Services -tekniikat lunastavat odotukset palvelukeskeisen ohjelmistojen integroinnin välineenä, on näitä tekniikoita tuettava integraatiototeutuksissa (Endrei ym. 2004, 128). Kuitenkaan mitään selkeää ohjenuoraa ei ole olemassa, mitä spesifikaatioita tulisi noudattaa palvelukeskeisen arkkitehtuurin toteuttamiseksi. Tämän vuoksi eri Web Services -sovelluskehykset tukevat vaihtelevasti eri spesifikaatioita hieman riippuen siitä, kenen osapuolen toimesta ne on kehitetty.

---

[1] Esimerkiksi IBM (2007) listaa WWW-sivuillaan noin 40 eri Web Services -spesifikaatiota.

4 MOBIILIT WEB SERVICES -PALVELUT

Edellisessä luvussa luotiin katsaus Web Services -tekniikoihin. Seuraavaksi käsitellään, mitä keinoja tekniikoiden käyttöön on mobiileisssa päätelaitteissa sekä mitä etuja ja haittatekijöitä käytöstä seuraa.

Farley ja Capp (2005) esittävät Web Services -termin rinnalle erityisen Mobile Web Services -termin. Tällä he tarkoittavat Web Services -tekniikoita hyödyntäviä sovelluksia mobiilissa ympäristössä. Tällaiset sovellukset eroavat kuitenkin perinteisistä Web Services -sovelluksista seuraavien piirteiden johdosta (Farley ja Capp 2005, 203):

- laitteen kannettavuus
- laitteen yhdistettävyys ja yksilöitävyys laitetta käyttävään henkilöön
- sovelluksen personointi käyttäjälle
- laitteen ominaispiirteiden aiheuttamat rajoitukset sovelluksille.

Mobile Web Services -palveluita hyödyntävien sovellusten tavoitteena on tarjota käyttäjän tietoihin ja sijaintiin perustuvia personoituja palveluita. Tämä tavoite ei välttämättä toteudu, jos käytetään vain perinteisiä Web Services -tekniikoita. Tässä tutkielmassa mobiileilla Web Services -palveluilla tarkoitetaan yleisesti Web Services -tekniikoilla toteutettujen palveluiden hyödyntämistä mobiileissa päätelaitteissa.

4.1 Strategiat Web Services -palvelujen hyödyntämiseen mobiileissa päätelaitteissa

Mobiilit päätelaitteet voivat käyttää Web Services -palveluita kahdella tavalla. Palvelua voidaan kutsua suoraan päätelaitteesta tai voidaan käyttää välikerrosta, joka kutsuu palvelua ja palauttaa tuloksen päätelaitteelle sen ymmärtämässä muodossa.

Yuan ja Long (2002) ovat esittäneet kolme arkkitehtuurimallia Web Services -palveluiden hyödyntämiseen mobiileissa päätelaitteissa:

- langaton portaali (*Wireless portal network*)

- langaton laajennettu Internet (*Wireless extended Internet*)
- vertaisverkot (*Wireless ad hoc network*).

Langattomassa portaalissa tiedon siirto mobiilin päätelaitteen ja palvelimen välillä kulkee WAP-yhdyskäytävän kautta WML-muodossa. Yhdyskäytävä muuntaa asiakkaalta tulevat pyynnöt SOAP-kutsuiksi ja lähettää SOAP-vastaukset takaisin päätelaitteelle WML-muodossa. Malli voidaan laajentaa käsittämään myös HTTP-protokollan päällä siirtyviä HTML-sivuja. Langattomalla laajennetulla Internetillä Yuan ja Long tarkoittavat Internetiä, joka on käytettävissä mobiileissa päätelaitteissa. Tämän edellytyksenä päätelaitteilla täytyy olla IP-osoite sekä täydet verkko-ominaisuudet. Langattomassa laajennetussa Internetissä mobiili päätelaite voi toimia suoraan Web Services -asiakkaana, jolloin kaikki tieto siirretään XML-muodossa HTTP- ja TCP/ IP-protokollia käyttäen. Vertaisverkkoarkkitehtuurissa myös mobiilit päätelaitteet voivat toimia palveluntarjoajina.

Langattoman portaalin mallissa sovellusta käytetään laitteen verkkoselaimella, joka toimii sovelluksen esityskerroksena. Kaikki sovelluksen toimintalogiikka sijaitsee välikerroksessa, portaalipalvelimella, ja Web Services -palveluita tarjoavalla palvelimella. Kyseessä on ns. laiha asiakas -malli. Mallin etuna on tekniikoiden kypsyys palvelinohjelmistoissa ja vähäisen tiedonprosessoinnin tarve päätelaitteessa. Mallin suuri haitta on portaalin hallitseva rooli. Portaali määrittää tarjottavat palvelut sekä yhteydet asiakkaisiin, joten uusien palveluiden kutsuminen vaatii muutoksia aina itse portaalin toiminnallisuuteen. Portaalin mahdollinen vikatila pysäyttää koko palveluiden ketjun toiminnan. Lisäksi päätelaitteen selain tarjoaa ainoastaan rajoitetut käyttöliittymämahdollisuudet. Selain ei myöskään voi tallentaa suoritettavan toiminnon tilaa, vaan käyttäjä joutuu jokaisessa istunnossa aloittamaan toiminnon suorittamisen alusta saadakseen tehtävän suoritetuksi. Tästä aiheutuu sekä ajan hukkaa että enemmän kustannuksia siirrettävän tiedon määrän kasvamisen seurauksena. Käyttäjän sijaintiin perustuvia palveluita ei myöskään voida hyödyntää. (Yuan ja Long 2002)

Mobiileilla päätelaitteilla pääsy suoraan Web Services -palveluihin sallii laitteen käyttäjien laajentaa laitteidensa toiminnallisuutta. Verkkosivuja selatessa käyttäjä joutuu kirjoittamaan verkko-osoitteita ja täyttämään sivuilla olevia lomakkeita ennen kuin hän pääsee haluamaansa tietoon käsiksi. Sen sijaan Web Services -palveluita hyödyntämällä voidaan minimoida käyttäjän tiedonsyöttö, yk-

sinkertaistaa prosesseja, helpottaa sovelluksen käyttöä sekä pienentää tiedonsiirrosta aiheutuvaa odotusaikaa. Päätelaitteiden kehityksen johdosta, osa tiedon prosessoinnista voidaan suorittaa päätelaitteessa. Mobiileissa sovelluksissa voidaan käyttää sofistikoituja käyttöliittymiä sekä tietoa voidaan tallentaa päätelaitteen muistiin ja käsitellä yhteydettömässä tilassa. Langattomaan portaaliin verrattuna tiedon prosessoinnin tapahtuessa päätelaitteessa, prosessorin käyttö ja muistin käyttö kasvaa aiheuttaen suuremman virran kulutuksen. (Zahreddine ja Mahmoud 2005; Yuan ja Long 2002)

Mobiilit päätelaitteet sisältävät runsaasti käyttäjän henkilökohtaisia tietoja, kuten esimerkiksi kalenterimerkintöjä ja yhteystietoja. Näiden tietojen sekä laitteen sijaintiin perustuvan informaation hyödyntäminen on mahdollista vertaisverkko-mallin mukaisesti. Tällaisiin henkilökohtaisiin tietoihin käsiksi pääsy vaatii palveluilta identiteetin varmistamisen, esimerkiksi identiteettipalveluiden avulla (Hirsch, Kemp ja Ilkka 2006).

4.2 Web Services -tekniikoiden käytön hyödyt

Luvussa 3.4 lueteltiin Web Services -tekniikoiden käytön yleisiä hyötyjä ohjelmistokehityksessä. Tässä kappaleessa tarkastellaan, mitä hyötynäkökohtia tekniikoiden käyttö tarjoaa erityisesti liikkuvaan tietojenkäsittelyyn.

Web Services -tekniikoiden on nähty soveltuvan käytettäväksi mobiileissa päätelaitteissa seuraavien ominaisuuksien ansioista (Zahreddine ja Mahmoud 2005; Steele 2003):

- Web Services -tekniikat ja XML-kieli ovat laitteistoriippumattomia ja sen vuoksi ne soveltuvat hyvin erilaisten mobiilien päätelaitteiden ja liityntäpisteiden (*access point*) yhdistelmille.

- Web Services -tekniikat mahdollistavat tiedon prosessoinnin palvelimella ja ainoastaan tulosten esittämisen mobiilissa päätelaitteessa.

- Mobiilien päätelaitteiden muistikapasiteetti on rajallinen, joten päätelaitteiden lukumuistissa ei ole mahdollista pitää suuria sovelluksia. Web Services -tekniikoiden ansiosta mobiilit päätelaitteet voivat tarvittaessa kutsua palvelimella sijaitsevien sovellusten palveluita.

- Tekniikat mahdollistavat omien mukautettujen palveluiden muodostamisen mobiileja päätelaitteita varten. Tällaiset palvelut voivat olla koostettu useasta eri palvelusta, jolloin käyttäjän tarvitsee kutsua vain yhtä palvelua usean eri palvelun sijaan.

4.3 Web Services -tekniikoiden käytön heikkoudet

Seuraavat piirteet on nähty heikkoutena Web Services -tekniikoiden käytölle mobiileissa päätelaitteissa (Sánchez-Nielsen, Martín-Ruiz ja Rodríquez-Pedrianes 2006; Apte, Deutsch ja Jain 2005; Hanslo ja MacGregor 2004; Arsanjani, Hailpern, Martin ja Tarr 2003, 54; Steele 2003).

- Tieto siirtyy sovellusten välillä XML-muodossa. XML-viestit tulee muuntaa sovellusten sisäisiksi tietorakenteiksi ja tämä vaatii tiedon prosessointia. Tästä tiedon käsittelystä aiheutuu virrankulutuksen kasvua.

- XML-esitysmuodossa varsinainen tietosisällön lisäksi on useita ylimääräisiä merkkejä kuvaamassa sisältöä. Tällöin siirrettävän tiedon määrä kasvaa esimerkiksi binäärimuotoiseen RMI:hin verrattuna 3-10-kertaiseksi.

- Osa Web Services -standardeista on kypsymättömiä ja joitain standardeja ei vielä ole olemassa.

- Mobiilisovellukset eivät pysty ottamaan yhteyttä UDDI-rekistereihin. Tämän johdosta palveluiden dynaaminen muodostaminen suorituksen aikana ei ole mahdollista. Palveluita voidaan kutsua ainoastaan sovelluksen staattisen tyngän kautta, joka muodostetaan sovelluksen suunnitteluaikana.

- Protokollat eivät tue transaktioiden käyttöä. Sovelluksen suunnittelijan tulee ottaa huomioon mahdolliset virhetilanteet ja niistä toipuminen.

Web Services -tekniikoiden käytöstä aiheutuneita haittoja on yritetty minimoida monella eri tapaa. XML-viestien kokoa on yritetty pakkaamalla niitä, mutta tämä aiheuttaa tiedonprosessoinnin kasvua, koska tieto pitää purkaa päätelaitteessa (Hanslo ja MacGregor 2004, 281).

Yhdeksi strategiaksi on esitetty, että palveluiden tuottajat voivat ottaa mobiilit asiakkaat huomioon tarjoamalla palveluita useassa eri muodossa (Kleijnen ja Raju 2003, 46). Esimerkiksi mobiiliasiakkaille voidaan tarjota räätälöityjä palveluja, joissa tiedon määrä ja esitysmuoto on karsittu minimiin.

Yksi kirjallisuudessa esitetty malli on käyttää hyväksi mobiileja ohjelmistoagentteja. Ohjelmistoagentti on objekti, joka kulkee verkossa ja suorittaa tehtäviä ja toimintoja käyttäjän puolesta (Zahreddine ja Mahmoud 2005; Campadello 2003).

5 JAVA PLATFORM, MICRO EDITION (JAVA ME)

Tähän mennessä on käsitelty tietojärjestelmien ja ohjelmistojen integrointia ja luotu katsaus, miten Web Services -tekniikoita voidaan hyödyntää mobiilisovelluksissa. Nyt luodaan tarkempi katsaus Java ME -alustaan, joka tarjoaa yhden sovellusympäristön mobiilisovelluksille.

Java ME on Sunin kehittämä Java-alusta verkkoyhteydellisille, sulautetuille ja pienille, resursseiltaan rajoittuneille laitteille. Tässä luvussa perehdytään Java ME:n arkkitehtuurimalliin, CLDC-konfiguraatioon, MIDP-profiiliin sekä MIDP-profiilin tueksi julkaistuihin erityistoiminnallisuuksia tarjoaviin sovellusliittymiin. Näiden avulla saadaan käsitys, mitä ominaisuuksia Java ME –sovellukset voivat sisältää ja missä ympäristössä ne toimivat. Lopuksi selvitetään, mitä keinoja Java ME -ympäristö tarjoaa Web Services -tekniikoiden hyödyntämiseen ja kuinka hyvin tekniikoita tuetaan.

5.1 Taustaa

Java ME on sulautettujen ja pienten, resursseiltaan rajoittuneiden laitteiden sovellusalusta ja ohjelmointiympäristö. Se ei ole kuitenkaan ensimmäinen pienille laitteille tarkoitettu Java-teknologia, vaan se syrjäytti lukuisat JDK 1.1 perusteiset tekniikat yhtenäisemmällä Java 2 -alustaan pohjautuvalla ratkaisulla. Java ME on ennemminkin aiempien tekniikoiden jälkeläinen. (Ortiz 2002; Topley 2002, 4.) Java ME:a edeltäneet teknologiat olivat Oak, Java Card, PersonalJava, EmbeddedJava, KVM ja Spotless System.

Kirjallisuudessa Java ME:a on käsitelty jonkin verran. Pääasiassa teoksissa selvitetään spesifikaatioiden määrittämät ominaisuudet selventävin esimerkein, mutta sen syvällisempää asioiden käsittelyä ei kirjallisuudessa ole tehty. Pääpaino kirjallisuudessa on MIDP-profiilissa ja CLDC-konfiguraatiossa. CDC-konfiguraatiota ja sen päälle tehtyjä profiileja ei kirjallisuudessa ole juuri tarkemmin käsitelty.

5.2 Arkkitehtuuri

Oikeastaan Java ME on spesifikaatioiden joukko, joka tarjoaa Java-teknologian hyödyntämisen pienissä, resursseiltaan rajoittuneissa laitteissa (Helal 2002, 1;

Topley 2002, 10). Java Commmunity Process (JCP) ohjaa spesifikaatioiden (Java Spesification Request, JSR) ja niiden referenssitoteutusten kehitystä.

Java ME koostuu Java-virtuaalikoneesta sekä sovellusliittymistä. Osa sovellusliittymistä on supistettu Java SE -alustasta ja osa on lisäyksiä Java ME -alustaan. Tärkeimmät sovellusliittymistä ovat konfiguraatiot ja profiilit. Konfiguraatio on tarkoitettu ominaisuuksiltaan ja resursseiltaan saman laitekategorian laitteille, esimerkiksi laitteille, joilla on kiinteä verkkoyhteys. Profiilit tarkentavat konfiguraation laitejoukkoa yksityiskohtaisemmilla ominaisuusvaatimuksilla, esimerkiksi näytön vähimmäisresoluutiolla ja muistiresurssien määrällä. Kuvion 12 mukaisesti Java ME:n arkkitehtuuri on jaettu kolmeen kerrokseen. Alin kerros koostuu laitteen käyttöjärjestelmästä, keskimmäisessä kerroksessa ovat konfiguraatio sekä Java-virtuaalikone ja ylimmässä kerroksessa profiilit.

KUVIO 12. J2ME:n arkkitehtuuri (Kontio 2002, 5)

5.2.1 Java-virtuaalikone

Java ME tarvitsee Java SE:n ja Java EE:n tavoin virtuaalikoneen ohjelmien suorittamiseksi. Java ME:a varten Sun on kehittänyt kaksi virtuaalikonetta, CVM:n ja KVM:n. KVM on suunniteltu Java-alustaksi muistirajoittuneille ja resursseiltaan pienille laitteille, kuten älypuhelimet, PDA:lle ja sulautetuille järjestelmille. (Kontio 2002, 29). CVM on virtuaalikone, joka on kirjoitettu C:llä. Se on tarkoitettu KVM:n kohdelaitteita tehokkaammille laitteille. Se tarjoaa tuen myös mm. sarjallistamiselle ja RIM:lle. (Ashri, Atkinson, Ayers, Hagling ym. 2001, 3).

5.2.2 Konfiguraatiot

Konfiguraatio määrittelee minimialustan laitteille, jotka voivat käyttää Java ME:a. Tarkemmin sanottuna se määrittelee, mitkä Java-kielen osat, Java-virtuaalikoneen ominaisuudet sekä minimikirjastot ja sovellusliittymät laitteen valmistajan tulee toteuttaa laitteessaan tukeakseen Java ME -teknologiaa (Arokoski, Jääskeläinen, Kontio, Köykkä ym. 2002, 139; Nokia 2002, 236; Riggs, Taivalsaari ja VandenBrink 2001, 14). Yleensä tietyn konfiguraation laitteilla on samankaltaiset muistiresurssit, verkkoyhteyksien nopeudet, sähkövirran tarpeet ja käyttöliittymäominaisuudet (White 2001, 725).

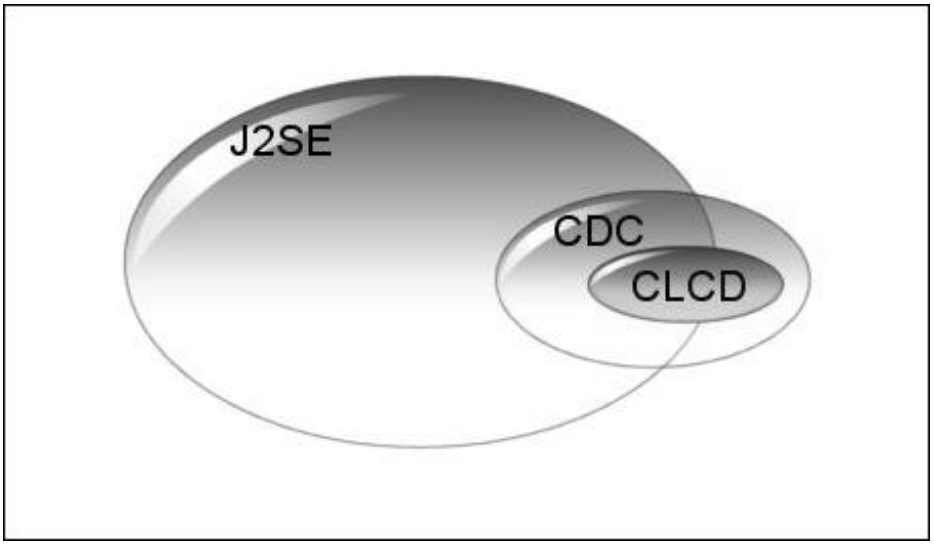
Tällä hetkellä on olemassa kaksi eri konfiguraatiota, jotka ovat molemmat tarkoitettu verkkoyhteydellisille laitteille. Verkkoyhteys voi olla joko kiinteä tai katkonainen. Kiinteän verkkoyhteyden laitteille tarkoitettu konfiguraatio on Connected Device Configuration (CDC) ja katkonaisen verkkoyhteyden mahdollistaville laitteille on Connection Limited, Device Configuration (CLDC).

CLDC-konfiguraatio keskittyy laitteisiin, joilla on katkonainen verkkoyhteys. Lisäksi laitteilla on yleensä yksinkertainen käyttöliittymä, minimaaliset muistiresurssit alkaen 128 kilotavusta sekä rajallinen virran saanti. Tyypillisiä laitteita ovat matkapuhelimet, hakulaitteet ja muistikirjamikrot. (Riggs ym 2001, 9-14) Konfiguraatioon on lisätty myös Java SE:iin kuulumattomia sovellusliittymiä, jotka soveltuvat erityisesti pienten laitteiden tarpeisiin. CLDC-konfiguraatio toimii KVM-virtuaalikoneen päällä. CLCD-konfiguraatiota käsitellään tarkemmin luvussa 5.3.

CDC-konfiguraatio keskittyy laitteisiin, joilla on kiinteä ja nopea verkkoyhteys. Laitteilla on yleensä myös korkeatasoiset käyttöliittymämahdollisuudet ja muistiresurssit ovat suuremmat kuin CLDC-konfiguraation laitteilla. Tyypillisiä tämän konfiguraation laitteita ovat esimerkiksi korkeatasoiset kommunikaattorit, Internet-yhteydelliset televisiot sekä autojen viihde- ja navigointilaitteet. Konfiguraation sisältämät kirjastot ja sovellusliittymät ovat paljon kattavammat ja laajemmat kuin CLDC-konfiguraatiossa. CDC-konfiguraatio toimii yleensä CVM-virtuaalikoneen päällä.

Konfiguraation välisiä yhteyksiä ja niiden suhdetta Java SE -ympäristöön voidaan havainnollistaa kuviolla 12. Valtaosa CDC:n ja CLCD:n toiminnallisuudes-

ta on peritty Java SE-ympäristöstä, mutta niihin on lisätty myös paremmin pienille laitteille soveltuvia luokkia.



KUVIO 13. Konfiguraatioiden ja Java SE:n väliset suhteet (Kontio 2002, 7).

5.2.3 Profiilit

Profiili määrittelee Java-alustan tietylle vertikaaliselle laitekategorialle (Riggs ym. 2001, 13). Profiili määrittelee minimijoukon sovellusliittymiä saman konfiguraation laitejoukolle. Nämä sovellusliittymät tarjoavat mahdollisuuden käyttää hyväksi kohdelaitteen resursseja ja ominaisuuksia. Profiili palvelee kahta tarkoitusta: laitteiden erikoistamista ja laitteiden siirrettävyyttä (Helal 2002, 83).

CLDC-konfiguraatiolle on olemassa kaksi profiilia: MIDP-profiili (Mobile Information Device Profile) ja IMP-profiili (Information Module Profile). MIDP-profiili on suunniteltu resursseiltaan rajoittuneille laitteille, kuten matkapuhelimille ja PDA:lle. MIDP-profiilia käsitellään tarkemmin luvussa 5.4. IMP-profiili on tarkoitettu erityisesti sulautetuille laitteille, joilla ei ole lainkaan graafista käyttöliittymää. Käytännössä profiili vastaa MIDP-profiilin 1.0 versiota, josta on jätetty pois käyttöliittymään liittyvät sovellusliittymät (Sun Microsystems 2003a). MIDP-profiilin laajennokseksi oli suunnitteilla erityinen PDA-profiili, joka olisi sisältänyt tuen AWT-käyttöliittymäkomponenteille ja henkilökohtaisentiedon hallintaan (Sun Microsystems 2002). Profiilista kuitenkin luovuttiin ja henkilökohtaisen tiedonhallinnan sovellusliittymät julkaistiin valinnaisina MIDP-profiilia täydentävinä sovellusliittyminä.

Personal Profile, PersonalJavan seuraaja, tarjoaa Java ME-ympäristön laitteille, joilla on kiinteä verkkoyhteys ja joiden käyttäjillä ei ole paljoa kokemusta tietokoneista. Se on suunniteltu CDC-konfiguraation laitteille, esimerkiksi satelliitti- ja digitaalitelevisiolle. (Ashri ym. 2001, 122.)

Toinen profiili CDC-konfiguraatiolle on Foundation Profile, joka on tarkoitettu laitteille, joilla on jonkinlainen verkkoyhteys, mutta jotka eivät tarvitse graafista näyttöä (Nokia 2002, 237).

5.3 CLDC-konfiguraatio

CLDC on Java ME:n konfiguraatioista erityisesti mobiileille päätelaitteille tarkoitettu konfiguraatio. Konfiguraatio tarjoaa sovellusliittymät resursseiltaan rajoittuneille laitteille, joilla on katkonainen verkkoyhteys. Tässä luvussa perehdytään tarkemmin CLDC:n kohdelaitteisiin ja ominaisuuksiin.

5.3.1 Kohdelaitteet

CLDC olettaa kohdelaitteilta tiettyjä ominaisuuksia. Ensinnäkin se olettaa, että muistia laitteessa on Javaa varten 192 kilotavua ja se on jaettu luku- ja käyttömuistialueisiin. Virtuaalikonetta ja CLDC-kirjastoja varten laitteessa tulee olla lukumuistia vähintään 160 kilotavua ja virtuaalikonetta varten tulee olla käyttömuistia 32 kilotavua. Toiseksi laitteilla on rajallinen energian saanti ja vähäinen virran kulutus. Kolmanneksi laitteilla on rajoitettu ja katkonainen verkkoyhteys. Viimeiseksi konfiguraatio olettaa, että laitteella on rajoitettu käyttöliittymä tai ei käyttöliittymää lainkaan. (Sun Microsystems 2000a, 17-80.; Sun Microsystems 2003b, 22-21).

5.3.2 Ominaisuudet

CLDC määrittelee seuraavat Javan toiminnallisuudet: Javan ydinkirjastot (java.lang.*, java.util.*), syöttö- ja tulostevirrat, verkkoliikenteen, turvallisuuden ja kansainvälistämisen. Se jättää kuitenkin käyttöliittymän ja tapahtumien hallinnan, korkeantason sovellusmallin (*high-end appliction model*) eli sovelluksen ja käyttäjän välisen vuorovaikutuksen ja pysyvän muistinhallinnan profiilien määritettäväksi. (Ashri ym. 2001, 20.)

Tässä yhteydessä ei käydä yksityiskohtaisesti CLDC:n sisältämiä luokkakirjastoja ja luokkia. Aiheesta ovat kirjoittaneet kattavasti mm. Sun Microsystems (2003b) sekä Riggs ym. (2001).

5.3.3 Rajoitteet

CLDC tarjoaa tiettyjä poikkeuksia lukuun ottamatta täyden tuen Java-kielelle. CLDC:n versiossa 1.0 ei ole liukulukuja (ei float- ja double-tietotyyppejä ja Double- ja Float-luokkia), koska useimmat CLDC-laitteet eivät tue liukulukuja, vaikka ohjelmallisesti tuen toteuttaminen olisi ollutkin mahdollista (Ashri ym. 2001, 20; Arokoski ym. 2002, 138; Riggs ym. 2001, 47). Konfiguraation viimeisimmässä versiossa 1.1 tuki liukuluvuille sen sijaan on lisätty. Muita muutoksia päivitetyssä versiossa ovat mm. Calendar-, Date- ja Zone-luokkien uudelleenmääritykset paremmin Java SE -yhteensopiviksi, tuki heikolle viittaukselle (*weak reference*) ja yksityiskohtaisempi luokkavarmennin (Sun Microsystems 2003b; Mahmoud 2003).

CLDC:ssä ei ole myöskään määritelty finalize-metodia. Siten finalize-metodia ei voida kutsua ohjelmoijan toimesta, vaan sitä kutsutaan automaattisesti, kun virtuaalikone huomaa, ettei olioon ole enää yhtään viittausta (Ashri ym. 2001, 20; Riggs ym. 2001, 47). Myös poikkeusten määrässä on jouduttu tinkimään. CLDC-konfiguraatioon on muistiresurssien vuoksi määritelty suppeampi poikkeusten joukko kuin Java SE:ssa ja Java EE:ssa (Kontio 2002, 38).

Lisäksi CLDC:ssa on seuraavat poikkeukset normaaliin Java-kieleen verrattuna (Ashri ym. 2001, 20; Arokoski ym 2002, 138; Riggs ym. 2001, 51-52.):

- ei Java Native Interface:a (JNI)
- ei reflektioita
- ei säiejoukkoja ja taustaprosessisäikeitä (*daemon threads*)
- ei käyttäjän muodostamia luokkalataajia (*class loaders*)
- ei RMI:a (*Remote Method Invocation*)
- ei sarjallistamista.

5.3.4 Turvallisuusmalli

Tietoturva on erittäin tärkeä asia Java ME-ympäristössä, koska sovellukset ladataan dynaamisesti verkosta. Tällöin on tärkeätä varmistua, etteivät sovellukset pääse vahingoittamaan laitetta eivätkä verkkoa. Java SE tarjoaa hyvät turvallisuusominaisuudet, mutta ne ovat liian laajat Java ME:lle. CLDC-konfiguraatiossa määritetään Java ME:lle turvallisuus malli, joka on jaettu kahteen tasoon: matalaan tasoon ja korkeaan tasoon (Riggs ym. 2001, 38).

Matala taso eli virtuaalikonetaso huolehtii, ettei Java-sovellus pääse vahingoittamaan laitetta. Tämä tapahtuu niin sanotun luokkavarmentimen (*classfile verifier*) avulla. Se huolehtii, etteivät sovellukset voi viitata sallitun muistialueen ulkopuolelle ja suoritettavat luokkatiedostot ovat kunnollisia. Java-virtuaalikoneen on hylättävä epäkelpo tiedosto. (Riggs ym. 2001, 38; Arokoski ym. 2002, 140.)

Luokkavarmennin ei kuitenkaan pysty toteamaan muuta kuin, että sovellus on kelvollista Java-koodia. On kuitenkin useita muita turvallisuus uhkia, jotka pitää ottaa huomioon. Korkeammalla tasolla eli sovellustasolla huolehditaan, että sovellus voi päästä käsiksi vain luokkakirjastoihin, järjestelmän resursseihin ja muihin komponentteihin, joihin laite ja Java-ympäristö sen sallivat päästä. Tämä on toteutettu siten, että jokainen ohjelma suoritetaan niin sanotun hiekkalaatikon sisällä. Hiekkalaatikko takaa, ettei ohjelma pääse käsiksi hiekkalaatikon ulkopuolisiin kirjastoihin ja tietoihin. Myös systeemiluokat on suojattu, joten sovellus ei pysty kirjoittamaan niiden päälle tai tilalle mitään. Järjestelmä voi rajoittaa myös sovellusten "moniajoa" yhteen ajettavaan sovellukseen kerrallaan. (Riggs ym. 2001, 39; Arokoski ym. 2002, 140.)

Monet mobiilisovellukset ovat hajautettuja, ja sovellusten osat kommunikoivat verkon välityksellä keskenään. Tämä vaatii monia muita keinoja turvallisuuden takaamiseksi. Kuitenkin lukuisten erilaisten arkkitehtuurien vuoksi, tämä päästä-päähän (*end-to-end*) ulottuva tietoturvaratkaisu on jätetty konfiguraation ulkopuolelle (Riggs ym. 2001, 41).

5.4 MIDP-profiili

Kuten luvussa 5.2.3 tuli ilmi, MIDP on CLDC:n päällä toimiva profiili. Se on tarkoitettu pääasiassa älypuhelinten ja kaksisuuntaisten hakulaitteiden ohjelmointiin (Arokoski ym. 2002, 145).

MIDP laajentaa CLDC-konfiguraatiota verkkoyhteyksillä, tietojen hallinnalla, äänillä, ajastimilla sekä käyttöliittymäominaisuuksilla. Lisäksi profiili tuo lisäominaisuuksia sovellusten jakelun hallintaan, sovellusten eliniän kontrolloimiseen sekä sovellusten turvallisuusmalliin. (Sun Microsystems 2006a, 13-14.)

5.4.1 Kohdelaitteet

MIDP vaatii, että sen kohdelaitteet täyttävät CLDC:n tapaan joukon vaatimuksia. MIDP:n vaatimukset voidaan jakaa neljään eri luokkaan: muistiin, näyttöön, tiedon syöttöön ja verkkoyhteyksiin. (Sun Microsystems 2000b, 21).

Profiilin versiota 1.0 tukevilla laitteilla lukumuistia täytyy olla vähintään 128 kilotavua MIDP komponentteja varten sekä 8 kilotavua sovellusten tarpeisiin. Profiilin versiota 2.0 tukevien laitteissa profiilin tarvitsema lukumuistin tarve on kasvanut 256 kilotavuun ja virtuaalikoneen käyttömuistin tarve on lisääntynyt 128 kilotavuun. Näiden lisäksi tulee vielä CLDC:n komponentit, joita ei ole sisällytetty näihin vaatimuksiin. Näytön tulee olla vähintään 96x54 pikselin kokoinen, 1 bitin syvyinen (musta ja valkoinen) ja pikselin muodon (*aspect ratio*) tulee olla noin 1:1. Laitteessa tulee olla ainakin jokin seuraavista tiedonsyöttömekanismeista: yhdellä kädellä käytettävä näppäimistö, kahdella kädellä käytettävä näppäimistö tai kosketusnäyttö. Verkkoyhteyden tulee olla kaksisuuntainen ja langaton. Yhteys voi olla mahdollisesti katkonainen. Lisäksi laitteen tulee pystyä toistamaan ääniä. (Sun Microsystems 2000b, 21-22; Sun Microsystems 2006a, 16.)

5.4.2 MIDP-sovellukset, MIDletit

MIDP-profiilin mukaisia sovelluksia kutsutaan MIDleteiksi. MIDletit ladataan palvelimelta ja sen jälkeen ne suoritetaan itse laitteessa. Ne ovat kuin sovelmat (*applet*), mutta niitä ei tuhota muistista käytön jälkeen. MIDletit toimivat myös paljon rajoittuneemmassa ympäristössä. Sovellus voi olla täysin itsenäinen

MIDlet tai MIDlet Suite. MIDlet Suite on sovellusten joukko, jotka on kerätty saman jar-paketin sisälle. (Kontio 2002, 47.)

Jokaisen sovelluksen Main-luokan tulee olla peritty MIDlet-luokasta. MIDlet-luokassa ei ole toteutettu metodeja, mutta se määrittää kolme tärkeää metodia: startApp(), destroyApp() ja pauseApp(). Näitä metodeja käytetään muuttamaan sovelluksen tilaa. MIDletin mahdolliset tilat ovat aktiivinen (*Active*), pysäytetty (*Paused*) ja tuhottu (*Destoyed*). MIDP-spesifikaation mukaan on olemassa myös ladattu-tila (*Loaded*). Tilaa ei ole määritetty missään, koska ohjelmoija ei voi käyttää sitä mitenkään. (Kontio 2002, 49.)

5.4.3 Turvallisuusmalli

MIDP-profiilin ensimmäisessä versiossa sovellukset määrättiin ajettavan Javan omassa hiekkalaatikossa (*sandbox*), jolloin mahdollinen haittasovellus ei päässyt käsiksi laitteen arkaluontoisiin ominaisuuksiin ja toiminnallisuuksiin. Profiilin versiossa 2.0 esiteltiin luotettujen sovellusten (*trusted applications*) turvallisuusmalli, jossa tämä hiekkalaatikko osittain rikotaan ja sallitaan luotetuksi osoitettujen sovellusten päästä käsiksi tiettyihin laitteiden ominaisuuksiin riippuen, millä turvallisuustasolla (*domain*) sovellus toimii. Jokainen turvallisuustaso määrittää, mitkä erityistoiminnot ovat automaattisesti sallittuja suorittaa tai mitkä toiminnot vaativat käyttäjän hyväksynnän ennen toiminnon suorittamista. Mikäli sovellus ei ole luotettu, toimii se profiilin version 1.0 mukaisessa hiekkalaatikossa. (Sun Microsystems 2006a, 30.)

MIDletit ja MIDlet Suitet voivat tulla luotetuiksi erityisen varmennus- ja allekirjoituskäytännön kautta. Prosessissa tunnettu kolmas osapuoli allekirjoittaa MIDlet Suiten käyttäen X.509 PKI -järjestelmää (Public Key Infrastructure). Laite tunnistaa allekirjoituksen perusteella allekirjoittajatahon ja käsittelee MIDlet Suitea luotettavana. Sovellusten kehittäjien tulee määrittää etukäteen sovellukselle, mitä järjestelmän lupaa vaativia toimintoja he haluavat käyttää. (Sun Microsystems 2006a, 36.)

Tässä yhteydessä ei tarkastella tarkemmin itse allekirjoitusprosessia, vaan aihetta ovat käsitelleet tarkemmin esimerkiksi Knudsen (2003) ja Sun Microsystems (2006a).

5.4.4 Käyttöliittymäominaisuudet

Abstract Windowing Toolkit (AWT) katsottiin sopimattomaksi käyttöliittymäratkaisuksi Java ME -ympäristöön. AWT on suunniteltu pöytäkoneille ja sen vaatimukset ja oletukset eivät ole sopivia pienille näytöille, joten MIDP-profiiliin päätettiin suunnitella oma kaksitasoinen käyttöliittymäratkaisu: High-level API ja Low-level API. (Riggs ym. 2001, 102; Kontio 2002, 47.)

High-level API on suunniteltu erityisesti yrityssovellusten mobiiliasiakkaita varten. Tyypillisiä sovelluksia ovat esimerkiksi kalenterit, muistiot ja tiedonhallintasovellukset. Tunnuksenomaista sovelluksille on, ettei niiden tarvitse kontrolloida jatkuvasti käyttöliittymää. (Riggs ym. 2001, 104; Kontio 2002, 47.)

High-level API:ssa näytölle piirtämisestä, näytön vierittämisestä ja käyttöliittymäkomponenttien ulkoasusta huolehtii laitteen käyttöjärjestelmä. Sovellukset eivät pääse käsiksi tiedonsyöttömekanismeihin, kuten yksittäisten näppäinten painalluksiin. (Sun Microsystems 2006a, 125.)

Low-level API on suunniteltu sovelluksille, jotka vaativat tarkkaa sijoittelua ja graafisten osien hallintaa sekä tapahtumien käsittelyä kuten näppäinten painalluksia. Tällaisia sovelluksia ovat esimerkiksi pelit sekä kaavioita ja grafiikkaa sisältävät ohjelmat (Riggs ym. 2001, 104; Kontio 2002, 47; Arokoski ym. 2002, 148). Low-level API:a käyttämällä sovellus voi kontrolloida, mitä näytölle piirretään, ja käsitellä primitiivejä tapahtumia, kuten tiettyjen näppäinten painalluksia (Sun Microsystems 2006a, 125).

5.4.5 Pysyvän tiedon hallinta

MIDP ei salli sovelluksen päästä käsiksi laitteen tiedostojärjestelmään. Profiili määrittää kuitenkin oman sovellusliittymän Record Management System (RMS), joka mahdollistaa tiedon tallentamisen laitteen pysyvään muistiin. Sovellusliittymän ansiosta tieto voidaan tallentaa laitteen muistiin ja se on sovelluksen hyödynnettävissä vaikka laite suljettaisiinkin välillä.

Käytännössä tietokanta (*record store*) koostuu tietueista (*record*). Aiemmin vain saman MIDlet Suiten MIDleteillä oli oikeudet lukea ja kirjoittaa tietokantaan, mutta profiilin versio 2.0 sallii tietokannan luojan päättää muiden sovellusten pääsystä tietokantaan. MIDlet Suiten sisällä tietokannan nimen tulee olla yksi-

löllinen ja maksimissaan se saa sisältää 32 Unicode-merkkiä. (Sun Microsystems 2006a, 458; Kontio 2002, 137; Riggs ym. 2001, 157.)

RMS ei sisällä mitään lukitusmekanismeja, joten sovelluksen vastuulla on, ettei toinen MIDlet tai säie pääse ylikirjoittamaan käsiteltävää tietoa (Sun Microsystems 2006a, 458).

5.4.6 Verkkoyhteydet

Verkkoyhteys on yksi tärkeimmistä Java ME:n ominaisuuksista. MIDletit pystyvät kommunikoimaan verkon yli HTTP-protokollaa käyttäen. Profiili edellyttää, että HTTP-protokolla on ainoa protokolla, jota MIDP-profiilin laitteen tulee tukea (Kontio 2002, 123).

MIDP 2.0 –versiossa verkkoyhteyksiä laajennettiin tukemaan suojattuja HTTPS- ja SSL-protokollia. Profiiliin lisättiin myös tuki matalantason IP-pohjaisille yhteyksille, työntöperustaisille tapahtumille (*push technology*) sekä sarjaporttiyhteyksille. Matalan tason IP-yhteydet tarkoittavat TCP/ IP-vastakkeita (*sockets*) ja UDP/ IP-yhteyksiä. (Sun Microsystems 2006a, 58.)

Profiilin versioon 2.0 lisätyt yhteysprotokollat ovat kuitenkin vapaaehtoisia toteuttaa, joten profiilia tukevien laitteiden ei ole pakko tukea jokaista protokollaa. Kaikki verkkoyhteydet avataan *javax.microedition.io.Connector.open*-metodilla, joka päättelee parametrien perusteella tarvittavan yhteysmuodon ja avaa yhteyden, mikäli laite tukee yhteysmuotoa. (Sun Microsystems 2006a, 59.)

Käytettäessä verkkoyhteyksiä on riski, että sovellus käyttää niitä tietoisesti väärin ja aiheuttaa siten kustannuksia laitteen käyttäjälle. Profiilin version 2.0 turvallisuusmallin mukaisesti verkkoyhteyksien käyttö sallitaan vain niille sovelluksille, jotka ovat allekirjoitettu tai joille käyttäjä on antanut luvan verkkoyhteyden käyttöön (Sun Microsystems 2006a, 59).

5.4.7 Tulevaisuus

MIDP-profiilin kehitys ei ole jäänyt versioon 2.0, vaan profiilin kolmannen version kehitystyö on aloitettu. MIDP 3:n tavoitteena on kehittää ja ylläpitää MIDP:n roolia johtavana mobiilisovellusalustana, lisätä uusia toiminnallisuuksia vastaamaan ohjelmistokehittäjiltä tulleisiin pyyntöihin, vähentää profiilin

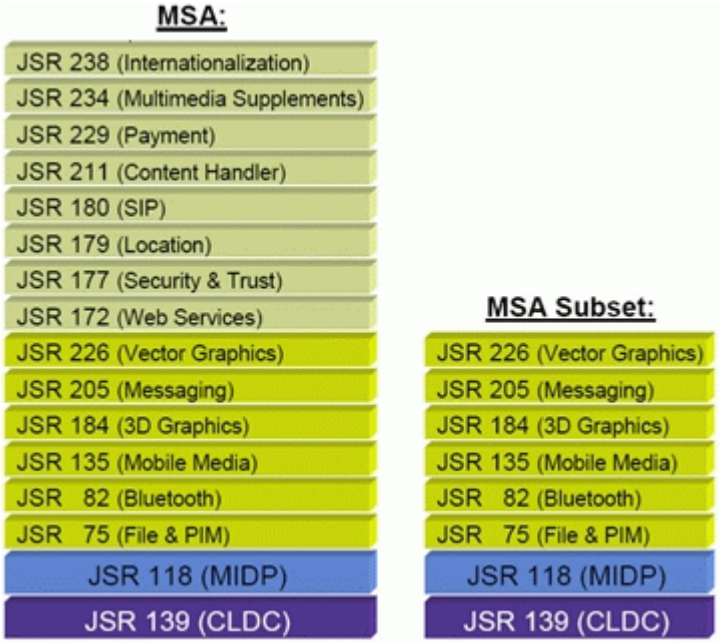
sekavuutta ja tiukentaa profiilin määrityksiä yhteentoimivuuden parantamiseksi sekä säilyttää profiilin taaksepäin yhteensopivuus (Sun Microsystems 2006b).

Profiiliin tullaan lisäämään tuet mm. yhtä aikaa suoritettaville MIDlet Suite:ille, taustalla toimiville MIDleteille sekä automaattisesti käynnistyville MIDleteille. Lisäksi MIDletit voivat käyttää hyväksi jaettuja kirjastoja. Käyttöliittymäominaisuuksia tullaan parantamaan mm. seuraavasti: parempi tuki suurille näytöille, tuki usealle näytölle, uusia käyttöliittymäkomponentteja sekä parempi tuki fonteille ja kuvien aliasestolle (*anti-aliasing*), alfakanavalle (*alpha blending*), skaalaamiselle, kääntämiselle sekä läpinäkyvyydelle. Tiedonhallintaan on tulossa tuki turvatuille, poistettaville ja siirrettäville RMS-tietokannoille. Verkkoyhteyksiä on kaavailtu parannettavaksi lisäämällä tuet mm. suojaverkolle (*Virtual Private Network, VPN*), IPv6:lle sekä IPSEC:lle (Sun Microsystems 2006b).

5.5 Mobile Services Architecture (MSA) (JSR 248)

Nykyiset mobiilit päätelaitteet sisältävät paljon sellaisia ominaisuuksia ja toiminnallisuuksia, jotka eivät kuulu MIDP-profiilin ja CLDC-konfiguraation vaatimuksiin, kuten esimerkiksi kamera, 3D-grafiikka ja Bluetooth-yhteys. MIDP-profiilin tueksi onkin julkaistu lukuisia profiilia täydentäviä spesifikaatioita, joiden avulla sovellukset voivat Java-kielen avulla käyttää tällaisia profiilin vaatimuksiin kuulumattomia ominaisuuksia. Spesifikaatiot rikkovat kuitenkin Javan ”kirjoita kerran, aja missä tahansa”-yhteensopivuuden, koska laitteet tukevat vaihtelevasti näitä spesifikaatioita. Tätä teknologian pirstaloitumista ehkäisemään on julkaistu Mobile Service Architecture (MSA) -spesifikaatio, joka määrittää konfiguraation, profiilin ja sovellusliittymien joukon standardoimaan tietyt laitteiden ominaisuudet ja Java-teknologiat niiden hyödyntämiseen (Sun Microsystems 2006c, 14). Lisäksi spesifikaatio vähentää ja poistaa joidenkin sovellusliittymien välisiä ristiriitaisuuksia (Ortiz 2006a).

CLDC-konfiguraatiolle tarkoitettu MSA määrittää kaksi teknologiapinoa: MSA ja MSA Subset (KUVIO 13). MSA Subset on nimensä mukaisesti MSA:n karsittu versio.



KUVIO 14. MSA-teknologiapinot (Sun Microsystems 2006c, 17).

MSA Subset-teknologiapinoon kuuluu CLDC-konfiguraation ja MIDP-profiilin lisäksi 6 sovellusliittymää (Ortiz 2006a):

- PDA Optional Packages for the J2ME platform (JSR 75) sisältää sovelluksliittymät, joilla pääsee käsiksi laitteen tiedostojärjestelmään sekä henkilökohtaisten tietojen hallintaan. Tällaisia tietoja ovat osoitekirjan yhteystiedot ja kalenterimerkinnät.

- Java APIS for Bluetooth (JSR 82) mahdollistaa Bluetooth-yhteyden sekä OBEX-protokollan käytön.

- Mobile Media API (JSR 135) tarjoaa pääsyn laitteen mediaominaisuuksiin, kuten esimerkiksi äänien ja videoiden toistamiseen.

- Mobile 3D Graphics API (JSR 184) tarjoaa tuen kolmiulotteisen grafiikan käsittelyyn.

- Wireless Messaging API (JSR 205) mahdollistaa SMS- ja MMS-viestien käsittelyn.

- Scalable 2D Vector Graphics API for J2ME (JSR 226) tarjoaa tuen kaksiulotteiseen skaalautuvaan vektorigrafiikkaan.

Täydellinen MSA-teknologiapino täydentää arkkitehtuuria vielä kahdeksalla sovellusliittymällä (Ortiz 2006b):

- J2ME Web Services (JSR 172) tarjoaa tuen XML:n prosessoinnille ja SOAP-viesteille.
- Security and Trust Services API (JSR 177) mahdollistaa kryptografisten palveluiden käytön, viestinnän älykorttien kanssa ja pääsyn PKI-palveluihin (Public Key Infrastructure services).
- Location API for J2ME (JSR 179) tukee paikkatietopalveluiden käyttöä.
- SIP API for J2ME (JSR 180) mahdollistaa SIP:n (*Session Initiation Protocol*) käytön.
- Content Handler API (JSR 211) mahdollistaa Java-sovellusten rekisteröidä tiettyjä sisältötyyppejä käsiteltäväkseen.
- Payment API (JSR 229) tarjoaa pääsyn maksamismekanismeihin.
- Advanced Multimedia Supplements (JSR 234) laajentaa JSR 135:ä tarjoamalla tuen kehittyneempien multimediaominaisuuksien hyödyntämiseen.
- Mobile Internationalization API (JSR 238) mahdollistaa sovellusten kansainvälistämisen ja kotoistamisen.

5.6 Java ME:n Web Services -sovellusliittymät

CLDC-konfiguraatiosta ja MIDP-profiilista puuttuu tuki XML:n prosessoinnille ja RPC-kutsuihin, jotka ovat edellytys Web Services -tekniikoiden käyttöön. Näitä tekniikoita tukemaan on julkaistu J2ME Web Services -spesifikaatio (JSR 172). Spesifikaatiossa määritettyjen sovellusliittymien toteuttaminen laitteissa on vapaaehtoista, joten tuki näille sovellusliittymille vaihtelee suuresti eri laitevalmistajien kesken. Spesifikaatio on kuitenkin otettu mukaan MSA-arkkitehtuuriin, minkä pitäisi standardoida tuen sovellusliittymille.

On olemassa myös kolmansien osapuolien XML-jäsentimiä ja SOAP-kirjastoja, joiden avulla Web Services -tekniikoiden käyttö on mahdollista Java ME-ympäristössä, joissa ei ole suoraa tukea JSR 172:n sovellusliittymille.

Tässä luvussa luodaan katsaus tekniikoihin, joiden avulla Web Services -palveluita on mahdollista käyttää MIDP-profiilin mukaisissa laitteissa. J2ME Web Services API:n lisäksi käsitellään kSOAP-kirjastot ja sekä Service Connection API for Java ME (JSR 279) -sovellusliittymä.

5.6.1 J2ME Web Services API

J2ME Web Services API:n päätavoitteena on mahdollistaa Web Services -tekniikoitten käyttö CLDC- ja CDC-konfiguraatioiden laitteissa. Spesifikaatio määrittää kaksi sovellusliittymää, joilla XML:n jäsentäminen ja SOAP- ja XML-pohjaisten Web Services -palveluiden kutsuminen on mahdollista.

SOAP-kutsujen lähettämisen mahdollistava JSR 172 JAX-RPC Subset API on karsittu versio Java SE:n Java API for XML-Based RPC:sta (JAX-RPC 1.1). Karsittuun versioon on otettu mukaan vain sellaiset ominaisuudet, jotka tarjoavat SOAP-kutsujen lähettämiseen tarvittavat ominaisuudet, ja jotka soveltuvat muistiresursseiltaan ja prosessointiteholtaan pienille laitteille.

Seuraavaksi käydään läpi nämä ominaisuudet ja rajoitteet (Sun Microsystems 2004; Ortiz 2006b):

- Kutsuttavien Web Services palveluiden tulee noudattaa WS-I Basic Profile Version 1.0:a. WS-I Basic Profile määrittelee lukuisia selvennyksiä ja suosituksia, jotka auttavat takaamaan paremman yhteensopivuuden erilaisten Web Services -toteutusten kesken.

- Sovellusliittymä tukee ainoastaan SOAP 1.1:a. SOAP 1.2:ssa on lukuisia uusia ominaisuuksia ja muutoksia, mutta WS-I Basic Profile ei näitä vielä tue.

- JAX-RPC tukee RPC- ja Document-keskeisiä operaatioita ja encoded- ja literal-tyylisiä viestien koodausta. JAX-RPC Subset API tarjoaa tuen ainoastaan Document/ Literal-tyylisille viesteille

- Sovellusliittymä tarjoaa tuen seuraaville SOAP:n tietotyypeille: boolean, byte, short, int, long, float, double, string, QName, base64Binary, hexBinary sekä kompleksisille tietotyypeille, jotka on strukturoitu primitiiveistä tietotyypeistä ja toisista kompleksisista tietotyypeistä.

- JAX-RPC 1.1 tukee kolmea eri palveluiden kutsumuotoa: staattista tynkää (*static stub*), dynaamista palvelun kutsumista (*Dynamic Invocation Interface, DII*) ja dynaamista välityspalvelinta (*dynamic proxy*). JSR 172 tukee näistä ainoastaan staattista palveluiden muodostamista ja kutsumista tynkäluokan avulla.

- Java ME -sovellus voi toimia ainoastaan palvelun kuluttajana, ei palvelun tarjoajana.

- Sovellusliittymä tarjoaa tuen ainoastaan kaksisuuntaisille synkronisille RPC-kutsuille. Se ei tarjoa tukea asynkronisille eikä yksisuuntaisille RPC-kutsuille

- Sovellusliittymä ei tue SOAP-viestien liitetiedostoja.

- JAX-RPC 1.1 tukee SOAP-viestien käsittelijöitä, joiden avulla voidaan esikäsitellä SOAP-viestejä, kuten esimerkiksi välimuistittaa tai salata nii80tä. Tämä ominaisuus on kuitenkin jätetty pois Java ME:n sovellusliittymästä.

- JAX-RPC 1.1 ei ota kantaa UDDI-rekistereihin, joten JAX-RPC Subset API ei myöskään tarjoa tukea niiden käytölle.

- JAX-RPC 1.1 tukee laajennettua tietotyyppien muuntamista (*Extensible Type Mapping*), jonka avulla voidaan muuntaa Java-tietotyyppejä XML-tyypeiksi ja toisin päin. Java ME:n JAX-RPC Subset API:sta tämä ominaisuus on jätetty pois.

- JAX-RPC 1.1 muuttaa SOAP-virheviestit SOAPFaultException- tai RemoteException-poikkeuksiksi. Java ME:n sovellusliittymä ei tue SOAPFaultExceptionia.

- JAX-RPC Subset API takaa tuen HTTP Basic -todentamiselle ja istuntojen hallinnalle.

XML:n jäsentämiseen tarkoitettu spesifikaatio määrittää JSR 172 JAXP XML-parsing API:n. Sovellusliittymä pohjautuu Java API for XMP Processing (JAXP) 1.1:een ja Simple API for XML:n 2.versioon (SAX2). Sen olennaisimmat ominaisuudet ovat seuraavat (Sun Microsystems 2004):

- tuki XML:n nimiavaruudelle
- tuki spesifikaatiossa määritetyille SAX2:n ominaisuuksille
- tuki sekä UTF-8- että UTF-16-merkistöille
- ei tukea Document Object Model:lle (DOM)
- ei tukea Extensible Stylesheet Language Transformations -tyylitiedostoille (XSLT)
- voi tukea DTD:a (Document Type Definition) XML:n kelpuutuksessa.

5.6.2 kSOAP

kSOAP on Java ME-ympäristössä yleisesti käytetty kolmannen osapuolen SOAP-kirjasto. kSOAP ja XML:n käsittelyn mahdollistavan kXML-kirjasto tarjoavat kevyen sovellusliittymän Web Services -tekniikoiden käyttöön. kSOAP ei tue kaikkia primitiivisiä tietotyyppejä, vaan tukemattomat tietotyypit muunnetaan SoapPrimitive-olioiksi ja kompleksit tietotyypit muunnetaan KvmSerializable-olioiksi. Ainoastaan int-, long-, string-tyypit muunnetaan vastaaviksi Java-tietotyypeiksi. kSOAP tarjoaa tuen myös SOAP 1.2 -versiolle. kSOAP tukee sekä document/ literal että RPC/ encoded -tyylisiä SOAP-viestejä. (kSOAP Project 2008.)

kSOAP:n versiossa 2 kSOAP kirjasto suunniteltiin täysin uudelleen. Uudessa versiossa mm. kirjaston koko rakenne muutettiin toimivammaksi, Literal-koodauksen tukea parannettiin ja SOAP-viestien serialisointia ja deserialisointia parannettiin ja se eriytettiin kokonaan omaksi paketiksi (kSOAP2 Project 2008).

kXML on kevyt XML-jäsennin, joka tarjoaa tuen XML:n nimiavaruudelle ja WBXML:lle (WAP Binary XML Content Format), kDOM:lle ja WML:lle (kXML Project 2008).

5.6.3 Service Connection API for Java ME (JSR 279)

JSR 172 on suppea sovellusliittymä Web Services -palveluiden käyttöön Java ME -ympäristössä. JSR 279 on valmisteilla oleva sovellusliittymä, jossa esitellään laajempi sovellusliittymä Web Services -palveluiden käyttöön. JSR 172 pe-

rustuu proseduurikutsuihin, jotka toteutetaan suunnitteluaikana generoitavien staattisten tynkäluokkien kautta. JSR 279:n tarkoituksena on tarjota joustavampi malli, joka tukee monipuolisemmin erilaisia Web Services -standardeja sekä dynaamista palveluiden kutsumista. Lisäksi se tarjoaa mahdollisuuden päästä ohjelmallisesti käsiksi XML-muotoisiin kutsu- ja vastausviesteihin. Sovellusliittymä tukee myös asynkronista viestinvälitystä sekä työntöperustaisia palveluita (*Push Services*). Lisäksi sovellusliittymä tukee RPC/ Encoded-tyylisiä viestejä. (Sun Microsystems 2006d.)

Sovellusliittymän puitteissa on tarkoitus luoda tuki WS-*-tekniikoille sekä identiteettipalveluille, jolloin sovellukset voivat hyödyntää paremmin palvelukeskeistä arkkitehtuurimallia.

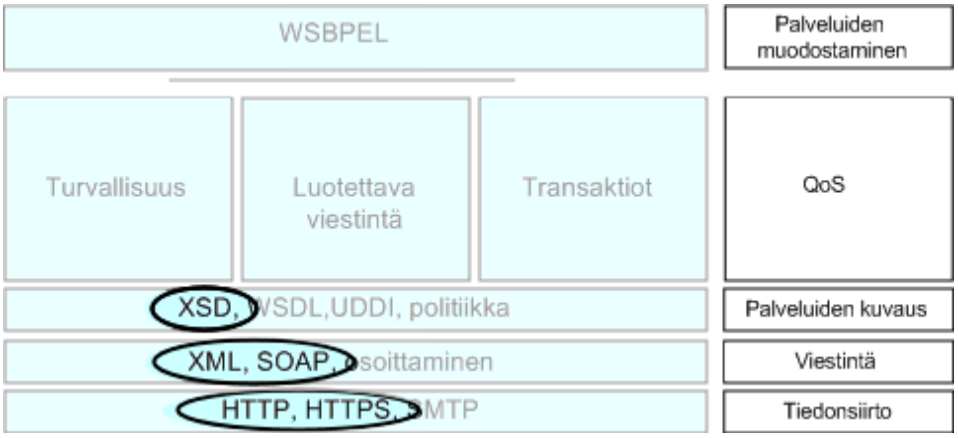
5.6.4 Yhteenveto Java ME:n Web Services -sovellusliittymissä tuetuista Web Services -tekniikoista

Taulukkossa 1 on esitetty tämän hetkisten Java ME -ympäristössä hyödynnettävissä olevien Web Services -sovellusliittymien ominaisuudet. JSR 279 ei ole vertailussa mukana, koska spesifikaation lopullista versiota ei ole vielä julkaistu eikä lopullinen sisältö ole vielä selvillä.

Tuetut Web Services -tekniikat mahdollistavat mobiilisovellusten vaihtaa tietoja SOAP-viestien välityksellä XML-muodossa (KUVIO 15). Muut palvelukeskeisessä ohjelmistojen integroinnissa käytettävät Web Services -tekniikat eivät ole tuettuja.

TAULUKKO 1. Java ME:n Web Services -sovellusliittymissä tuetut Web Services -tekniikat. Tuetut ominaisuudet on merkitty x-kirjaimella.

Ominaisuus / Sovellusliittymä	JSR 172	kSOAP	kSOAP2
SOAP 1.1	x	x	x
SOAP 1.2	-	x	x
UDDI	-	-	-
RPC/ Encoded -tyylinen viestintä	-	x	x
Document/ Literal -tyylinen viestintä	x	x	x
WS-BPEL	-	-	-
WS-Security	-	-	-
WS-Trust	-	-	-
WS-ReliableMessaging	-	-	-
WS-Addressing	-	-	-
WS-Coordination	-	-	-
WS-Policy	-	-	-
ID-FF	-	-	-
ID-WSF	-	-	-
ID-SIS	-	-	-



KUVIO 15. Palvelukeskeisen arkkitehtuurin mahdollistavien Web Services -tekniikoiden tuki Java ME -ympäristössä. Ympyröidyt tekniikat ovat tuettuja.

6 TUTKIMUKSEN SUORITUS

Aiemmissa luvuissa on selvitetty, mitä ovat palvelukeskeinen ohjelmistojen integraatio sekä Web Services -tekniikat. Lisäksi käytiin läpi, kuinka Web Services -tekniikat skaalautuvat mobiiliin tietojenkäsittelyyn ja miten tekniikoiden käyttö on mahdollista erityisesti Java ME -ympäristössä.

Taustatutkimuksessa selvisi, että Java ME:n Web Services -sovellusliittymät tukevat XML:n käsittelyä ja SOAP-viestejä, joiden avulla tietojen vaihtaminen sovellusten kesken on mahdollista. Tässä luvussa tutkitaan, mitä vaikutuksia tekniikoiden käytöstä aiheutuu, toisin sanoen onko käyttäminen järkevää.

Tässä luvussa kuvataan konstruktiot, joissa Java ME -asiakassovellukset käyttävät SOAP-protokollan avulla palvelimen tarjoamia Web Services -palveluita. Tarkoituksena on yhtäältä tutkia yleisesti Web Services -tekniikoiden käytön mahdollisuutta ja vaikutuksia ja toisaalta vertailla eri sovellusliittymien tehokkuutta keskenään. Tutkimuksessa otetaan huomioon myös käytettävä yhteysmuoto. Tekniikoiden vaikutuksia verrataan konstruktioon, jossa vastaava palvelu on toteutettu HTML-sivujen avulla. Toisin sanoen tutkimuksessa verrataan keskenään langattoman portaalin ja laajennetun langattoman Internetin malleja (ks. luku 4.1). Palvelinsovelluksen toiminnallisuuden arviointi on jätetty tämän tutkielman ulkopuolelle.

6.1 Taustaa

Tässä tutkielmassa yhdistyy kaksi tietojenkäsittelytieteen hyvin erilaista osa-aluetta. Web Services -tekniikat ja palvelukeskeinen ohjelmistojen integrointi sekä mobiili tietojenkäsittely ja mobiilisovellukset. Web Services -tekniikoiden toiminnallisuutta on tutkittu jonkin verran eri kannoilta ja perinteisten ohjelmistojen arviointimetriikat ovat vakiintuneet. Sen sijaan mobiilisovellusten toiminnallisuuden arviointiin metriikoita ei juuri ole luotu. (Ryan ja Rossi, 2005.)

Machado ja Ferraz (2005) ovat koostaneet suuntaviivat, joiden avulla voidaan arvioida Web Services -työkalupakin (*toolkit*) tehokkuutta. He ovat pyrkineet ottamaan kattavasti huomioon sellaiset seikat, jotka vaikuttavat työkalupakin toiminnallisuuteen ja antavat hyvän yleiskuvan työkalupakin toiminnallisuudesta. Machado ja Ferraz mainitsevat viisi työkalupakin toiminnallisuuteen

vaikuttavaa osa-aluetta: 1) XML-viestin koko, 2) viestin koon laskemiseen kuluva aika HTTP-pyynnön otsikkotietoja varten, 3) XML-jäsentimen tehokkuus, 4) olioiden serialisoinnin ja deserialisoinnin kustannukset, 5) yhteyden aloittamisen kustannukset.

Ryan ja Rossi (2005) ovat koostaneet mobiilisovellusten toiminnallisuuden arviointiin metriikoiden ja attribuuttien taulukon (TAULUKKO 2). He ovat jakaneet mitattavat attribuutit kolmeen eri osa-alueeseen riippuen siitä, minkä arviointiin ne liittyvät: sovellukseen, toiminnallisuuteen tai resurssien käyttöön. Taulukossa on yksityiskohtaisesti esitetty mitattava asia, metriikka ja mittayksikkö, joilla tapausta arvioidaan.

TAULUKKO 2. Ohjelmiston (*Software*), toiminnallisuuden (*Performance*) ja resurssien hyödyntämisen (*Resource Utilisation*) arvioinnin kohteet ja metriikat (Ryan ja Rossi 2005).

Attribute	Definition	Metric	Unit
Software			
Object Compilation Volume	The size of an executable module (e.g. Java .class file)	Executable Code Size (ECS)	byte
Object Serialisation Volume	The size of a serialised object	Serialised Object Size (SOS)	byte
Object Memory Volume	The size of an in-memory object	Object Memory Size (OMS)	byte
Method Execution Volume	The size of the extra memory required during the execution of a method	Execution Memory Size (EMS)	byte
Method Body Intensity	The processing intensity of a method in terms of executable statements/instructions	Number of Executed Instructions (NEI)	int
Method Interface Volume	The combined size of the parameters of a method interface	Size of Serialised Parameters (SSP)	byte
Method Invocation Frequency	The rate of occurrence of method invocation	Number of Invocations (NI)	int
Performance			
Method Execution Cost	The execution cost of a method, ignoring any overhead associated with call semantics	Method Execution Time (ET)	ms
Method Invocation Cost	The cost of calling a method, independent of its actual processing e.g. marshalling etc.	Method Invocation Time (IT)	ms
Object Migration Cost	The cost of moving an object instance (e.g. serialised Java object) between hosts	Migrate Instance Time (MIT)	ms
Class Migration Cost	The cost of moving a class implementation (e.g. Java .class file) between hosts	Migrate Class Time (MCT)	ms
Resource Utilisation			
Network	The network bandwidth between two hosts	Network Capacity (NC)	byte/s
Network Utilisation	The aggregate network bandwidth between two hosts	Network Usage (NU)	byte
Memory	The total memory available on a host	Memory Capacity (MC)	byte
Memory Utilisation	The aggregate memory usage of a host	Memory Usage (MU)	byte
Processor	The processing power of a host	Processor Capacity (PC)	int/s
Processor Utilisation	The aggregate processor usage of a host	Processor Usage (PU)	int

6.2 Tutkimusmenetelmä

Tässä tutkielmassa käytetään kvantitatiivista tutkimusmenetelmää analysoimaan eri Web Services -sovellusliittymien ja -lähestymistapojen toiminnallisuutta ja käyttäytymistä. Tätä varten on suoritettu seuraavat toimenpiteet:

1. Toteutettu eri sovellusliittymiä hyödyntävät asiakassovellukset ja instrumentoitu ne siten, että testausdataa voidaan kerätä sovelluksen suorituksen aikana.

2. Toteutettu palvelinsovellus, jonka tuottamia palveluita asiakassovellukset käyttävät hyväkseen.

3. Suoritettu ennalta määritetyt testitapaukset asiakassovelluksissa.

6.3 Testausympäristö

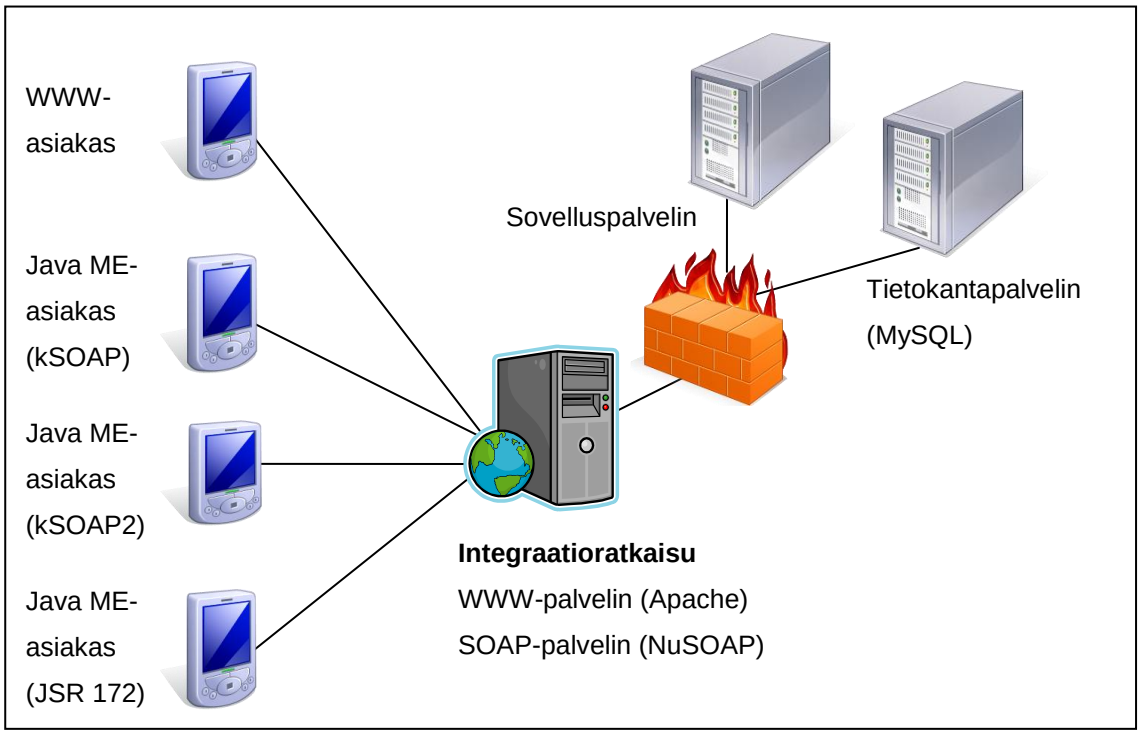
Konstruktioiden tekninen arkkitehtuuri (KUVIO 16) edustaa perinteistä kolmikerrosarkkitehtuuria. Arkkitehtuuri on mahdollisimman yksinkertainen, jolloin eri konstruktioiden vertaaminen on helppoa.

6.3.1 Asiakaskerros

Asiakassovellukset suoritettiin kaikki samassa päätelaitteessa, Nokia E70 -matkapuhelimessa. Nokia E70 -matkapuhelimen prosessori on Texas Instruments:n ARM-926-perustainen OMAP1710-prosessori, joka toimii 220MHz:n kellotaajuudella. Käyttöjärjestelmänä on Symbian OS 9.1, S60 3rd edition. Puhelin tukee GPRS-, HSCSD-, EDGE-, 3G- ja WLAN-datayhteyksiä. Javaa varten puhelimessa on CLDC 1.1 -konfiguraatio, MIDP 2.0 -profiili sekä tuki mm. JSR 172 -sovellusliittymälle. Internet-selain tukee HTTP-protokollan avulla HTML- ja XHTML-sivuja sekä WAP 2.0 -protokollan avulla WML-sivuja.

6.3.2 Palvelinkerros

Palvelinkerros simuloi tässä tutkielmassa integraatioväliohjelmistoa, joka tarjoaa yhteydet ja rajapinnat yrityksen eri järjestelmiin. Palvelinkerroksessa on Apache WWW -palvelin PHP-tuella. WWW-palvelimen yhteydessä on SOAP-palvelinohjelmisto (NuSOAP), jonka koodiin tehtiin pieniä muutoksia WSI-profiilin tukea varten. Lisäksi taustalla ovat tietokantapalvelin ja sovelluspalvelin, joita WWW-palvelin hyödyntää.



KUVIO 16. Järjestelmän tekninen ympäristö, jossa konstruktioiden toiminnallisuutta testataan.

6.4 Testauskonfiguraatiot

Web Services -tekniikoiden käyttöä mobiilissa päätelaitteessa arvioidaan neljän eri testauskonfiguraation avulla. Konfiguraatiot ovat:

1. JSR 172 -sovellus
2. KSOAP-sovellus
3. KSOAP2-sovellus
4. Web-asiakas

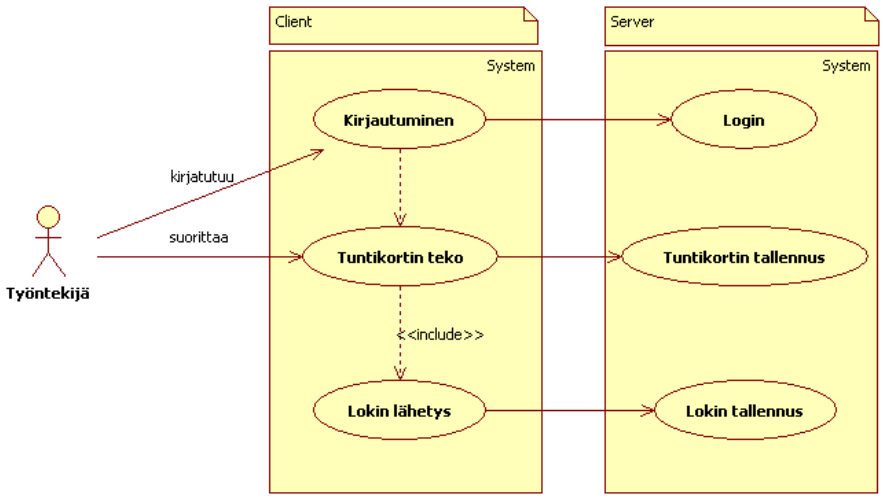
Kolme ensimmäistä konfiguraatiota ovat MIDlet-sovelluksia, joissa käytetään eri SOAP-sovellusliittymiä. Neljäs konfiguraatio esittää käyttäjälle saman toiminnallisuuden, mutta se suoritetaan laitteen WWW-selaimella HTML-sivujen avulla.

6.5 EviaProject-järjestelmän kuvaus

EviaProject on tuntiseurantajärjestelmä, jonka avulla on mahdollista raportoida projekteihin liittyviä tuntikortteja. Täytettävä tuntikortti kohdennetaan projektin vaiheeseen. Tuntikortilla raportoidaan vaiheeseen ennalta kiinnitettyjen työtehtävien tekemiseen kuluneita tunteja. Järjestelmä on hajautettu asiakas- ja palvelinkerroksiin

6.5.1 Asiakassovellus

Järjestelmän asiakassovellus on käyttöliittymäsovellus, jolla tallennetaan järjestelmään tuntikortti. Tuntikortin tallentaminen edellyttää kirjautumista palveluun ja tuntikortin tietojen täyttämiseen (KUVIO 17). Toiminnallisuuden arvioinnin mahdollistamiseksi järjestelmä lähettää lopuksi lokitiedot palvelimelle.



KUVIO 17. Tuntikortin tallennusprosessi EviaProject-järjestelmään käyttötapauskaavion muodossa mallinnettuna.

Tuntikortin täyttäminen jakaantuu viiteen vaiheeseen. Nämä vaiheet ovat:

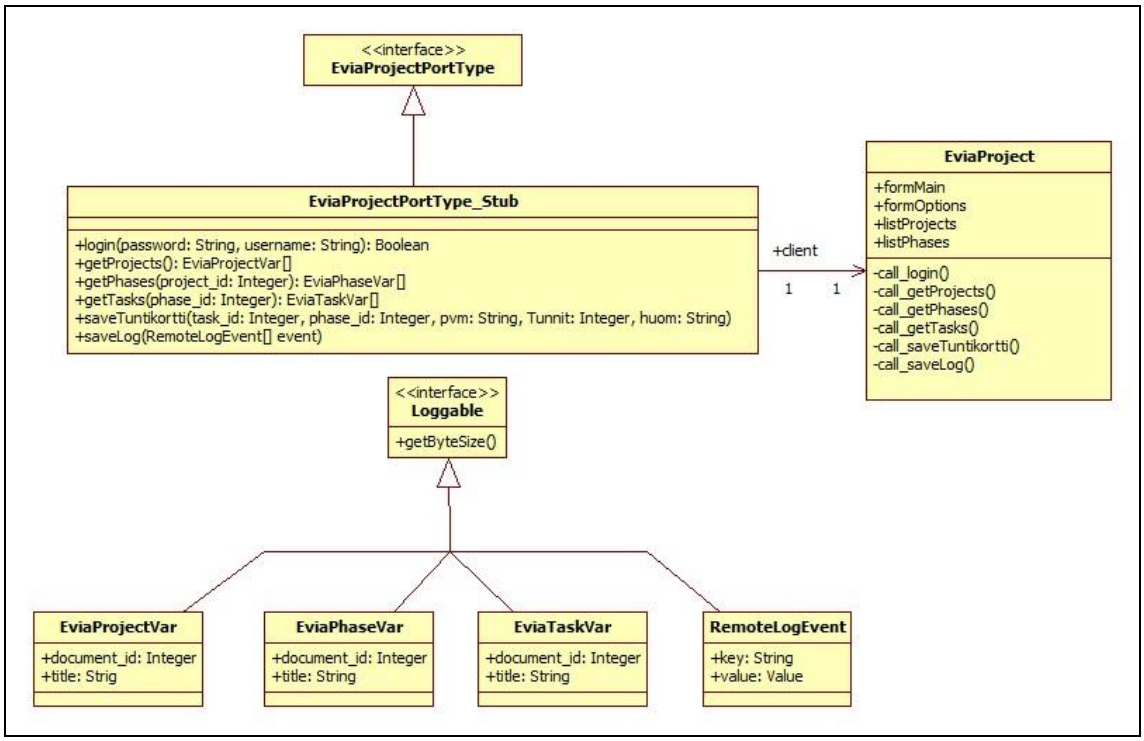
1. Sisäänkirjautuminen
2. Projektin valinta
3. Projektin vaiheen valinta

4. Työtehtävän valinta ja tuntien kirjaaminen

5. Tuntikortin tallennus

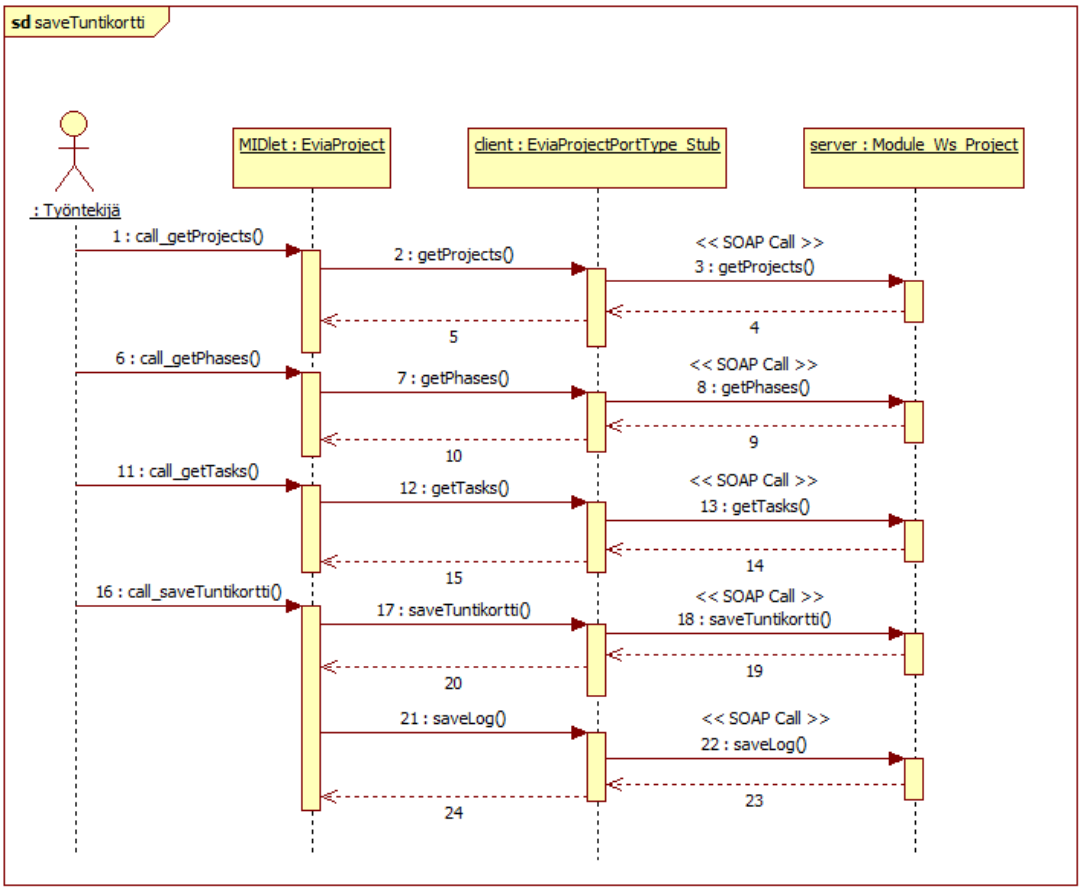
Sisäänkirjautumisessa käyttäjä kirjautuu järjestelmään henkilökohtaisilla tunnuksillaan. Tunnukset voidaan tallentaa MIDlet-sovelluksissa laitteen muistiin. Projektin valinnassa käyttäjälle listataan ne projektit, joissa hän on työntekijänä. Käyttäjä valitsee näistä yhden projektin, jolle tuntikortti raportoidaan. Vaiheen valinnassa tarkennetaan valintaa valitsemalla projektin vaiheen, johon tuntikortti kohdennetaan. Työtehtävien valinnassa valitaan haluttu työtehtävä ja kirjataan ajankohta, tuntimäärä sekä huomioitavat asiat. Tuntikortin tallennuksessa lähetetään tuntikortin tiedot palvelimelle.

Arkkitehtuurisesti asiakassovellusten toiminnallisuus on jaettu kahteen luokkaan: EviaProject ja EviaProjectPortType_Stub (KUVIO 18). EviaProject on MIDlet-luokka, joka sisältää käyttöliittymäkomponentit sekä käyttöliittymän tapahtumien hallinnan. EviaProjectPortType_Stub on staattinen tynkäluokka, joka huolehtii Web Services -palveluiden kutsumisista. EviaProject-luokka on kaikissa asiakassovelluksissa täysin sama. EviaProjectPortType_Stub on sen sijaan riippuvainen käytettävästä SOAP-sovellusliittymästä, joten sen sisäinen toteutus on hiukan erilainen jokaisessa asiakassovelluksessa. Jokainen EviaProjectPortType_Stub-luokka toteuttaa kuitenkin saman EviaProjectPortType-rajapinnan, jolloin palveluiden kutsuissa ja vastauksissa käytetään samoja tietotyyppejä: EviaProjectVar, EviaPhaseVar, EviaTaskVar ja RemoteLogEvent.



KUVIO 18. MIDlet-sovellusten luokkakaavio.

Tuntikortin tietojen täyttäminen etenee järjestelmässä vaiheittain. Jokaisen vaiheen yhteydessä sovellus hakee SOAP-kutsulla palvelimelta suoritusvaiheeseen liittyviä tarkempia tietoja (KUVIO 19).



KUVIO 19. Tuntikortin tallennuksen sekvenssikaavio

6.5.2 Palvelinsovellus

Palvelinsovelluksessa toteutetaan asiakassovellukselle tarjottavat palvelut, jotta tuntikortin täyttäminen ja tallennus ovat mahdollisia. Palvelinsovelluksen toiminnallisuus on rajattu tämän tutkielman ulkopuolelle.

6.6 Tutkimuksen toteutus

Tutkimus suoritettiin täyttämällä tuntikortteja jokaisella asiakassovelluksella eri yhteysnopeuksia käyttäen (GPRS, 3G, WLAN). Tuntikortti täytettiin aina samoilla tiedoilla samalle projektille. Näin siirrettävän tiedon määrä pysyi vakiona mahdollistaen tulosten vertailukelpoisuuden. Jokaisella konfiguraatiolla täytettiin kymmenen tuntikorttia jokaisella yhteysnopeudella. Yhdellä konfiguraatiolla täytettiin täten yhteensä 40 tuntikorttia, jolloin täytettyjen tuntikorttien yhteismäärä oli 120 kappaletta.

Tutkimukseen osallistui viisi koehenkilöä, kolme miestä ja kaksi naista. Koehenkilöt olivat iältään 25-60-vuotiaita. Henkilöt olivat käyttäneet vastaavaa järjestelmää tietokoneella, joten järjestelmän toimintalogiikka oli heille entuudestaan tuttu. Henkilöt täyttivät tuntikortteja eri konfiguraatioilla satunnaisessa järjestyksessä. Tällä pyrittiin ehkäisemään käytön oppimisen vaikutuksia.

6.7 Tutkimuksen kohteet

Tutkimuksen kohteet ja metriikat on valittu Machadon ja Ferraz:n sekä Ryanin ja Rossin esittämien menetelmien ja metriikoiden pohjalta. Lisäksi mukaan on otettu kohteita, joiden avulla voidaan mitata yleisemmällä tasolla Web Services -tekniikoiden käytön vaikutuksia suoritusaikaan.

Tässä tutkielmassa tutkittavia kohteita ovat:

- siirrettävän tiedon määrä tavuina
- tuntikortin täyttämiseen kuluva aika millisekunteina
- getProjects-metodin suorittamiseen kuluva aika millisekunteina
- SOAP-vastauksen deserialisointiin kuluva aika millisekunteina
- deserialisoitujen olioiden koko tavuina
- MIDlet-sovelluksen koko

Asiakassovellusten koodi on instrumentoitu siten, että näiden tietojen kerääminen on mahdollista.

6.7.1 Siirrettävän tiedon määrä tavuina

Siirrettävän tiedon määrää mittaamalla selvitetään, kuinka paljon tietoa siirtyy eri asiakassovelluksia käytettäessä. SOAP-viestien koon oletetaan olevan kirjallisuuden perusteella suurempi kuin HTML-muodon. Mittaamalla testataan, onko asia näin ja kuinka paljon enemmän tietoa mahdollisesti siirtyy. Lisäksi selvitetään, onko eri sovellusliittymien käytöllä eroa siirrettävän tiedon määrään.

Siirrettävän tiedon määrä mitataan jokaisen vaiheen aikana. Tiedot saadaan tutkimalla HTTP-kutsujen ja -vastausten tietoja. Tiedon määrässä huomioidaan ainoastaan viestien sisällön (*content*) koko, joka vaihtelee konfiguraatioittain.

6.7.2 Suoritusaika millisekunteina

Tuntikortin tallentamiseen kuluvan kokonaisajan mittaamisella saadaan selville, onko eri sovellusliittymien välillä merkittävää eroa ja esiintyykö suoritusajoissa eroa MIDlet-sovellusten ja Web-sovelluksen välillä. Suoritusajoista lasketaan minimi-, keski- ja maksimiarvot yhteysmuodoittain ja sovellustyypeittäin. Tallennusprosessin suoritus jakautuu vaiheisiin, jotka on nimetty sen aikana suoritettavan SOAP-kutsun mukaisesti. Tuloksissa tarkastellaan myös, kuinka suoritusaika jakautuu eri osavaiheiden kesken.

MIDlet-sovelluksissa suoritusajan mittaaminen aloitetaan, kun käyttäjä painaa ”Kirjaudu”-painiketta sovelluksen aloitusnäytössä. Suoritusajan mittaus lopetetaan, kun käyttäjä on tallentanut tuntikortin ja sovellus on vastaanottanut SOAP-kutsun palautusarvon. Web-asiakkaalla suoritusajan mittaus aloitetaan, kun käyttäjä painaa aloitussivulta ”Aloita”-painiketta, jolloin hänelle avautuu sisäänkirjautumissivu. Mittaaminen lopetetaan, kun käyttäjä on tallentanut tuntikortin ja selain on vastaanottanut tallennuksen jälkeisen HTML-sivun.

6.7.3 getProjects-metodin suorittamiseen kuluva aika millisekunteina

Tallennusprosessin vaiheista tarkemmin arvioidaan getProjects-vaihetta, jonka aikana suoritetaan MIDlet-sovelluksissa getProjects-metodi. Vaihe on valittu sen vuoksi, että sen aikana siirtyy eniten tietoa asiakkaan ja palvelimen välillä. Mittauksella selvitetään, onko suoritusaikojen välillä eroavaisuuksia sovellusten kesken ja miten siirrettävän tiedon määrän erot näkyvät suoritusajoissa MIDlet-sovellusten ja Web-asiakkaan kesken. Web-asiakkaalle on koodi instrumentoitu siten, että sille voidaan mitata vastaava suoritusaika kuin MIDlet-sovelluksilla. Vaiheen suorittamiseen kuluvaa aikaa arvioidaan laskemalla minimi-, maksimi- ja keskiarvot yhteysmuodoittain ja sovellustyypeittäin.

MIDlet-sovelluksissa vaiheen suoritukseen kuluvan ajan mittaaminen aloitetaan, kun käyttäjä valitsee valikosta ”Lisää tuntikortti”-valinnan. Tällöin sovellus pyytää käyttäjän projektit palvelimelta, joka palauttaa ne taulukkomuodos-

sa (ts. sovelluksessa kutsutaan getProjects-metodia). Mittaaminen lopetetaan, kun sovellus on vastaanottanut projektit. Web-sovelluksessa käyttäjä painaa ”Aloita”-painiketta, jolloin sivun lomaketiedot lähetetään palvelimelle ja palvelin generoi HTML-sivun, jossa on listattu käyttäjän projektit. Mittaaminen lopetetaan, kun sivu on latautunut selaimeen.

6.7.4 SOAP-viestien deserialisointiin kuluva aika millisekunteina

SOAP-viestien deserialisointiin kuluvalla ajalla selvitetään, onko Java ME-sovellusliittymien SOAP-viestien käsittelyssä ja niiden muuntamisessa Java-tietorakenteeksi merkittäviä eroja. Arviointi suoritetaan laskemalla sovellusliittymittäin viestien deserialisointiin kuluvan ajan minimi-, keski- ja maksimiarvot.

6.7.5 Deserialisoidun tietorakenteen koko tavuina

Deserialisointi-vaiheessa MIDlet-sovellus muuntaa SOAP-tietorakenteen Java-tietorakenteeksi. Muunnettavista tietorakenteista tarkastellaan erityisesti getProjects-metodin palautusarvon muuntamista EviaProjectVar[]-taulukoksi. Metodin palauttama taulukon koko saadaan mitattua tavuina. Kokoa voidaan verrata SOAP-vastausviestin kokoon, jolloin voidaan verrata, kuinka paljon SOAP-viestin koko kutistuu (ts. deserialisoinnin kustannukset), kun se muutetaan Java-tietorakenteeksi. Java-tietorakenne on kaikissa sovelluksissa täysin identtinen, joten vertailu on mahdollista.

6.7.6 MIDlet-sovelluksen koko

Tutkielmaan valitussa päätelaitteessa on toteutettu JSR 172 -sovellusliittymä, jolloin Web Services -palveluita voidaan kutsua tämän sovellusliittymän kautta ilman lisäosien tarvetta. Sen sijaan kSOAP-kirjastot tulee sisällyttää niitä hyväksikäyttäviin sovelluksiin, jolloin sovelluksen fyysinen koko kasvaa. Sovellusten koot selvittämällä, voidaan arvioida kSOAP-kirjastojen käytön vaikutukset. Sovellusten koot saadaan päätelaitteesta.

6.8 Tutkimuksen rajoitteet

Testitapaukset suoritetaan julkisessa matkapuhelinverkossa, jolloin verkon toiminnassa voi esiintyä häiriöitä ja yhteysnopeuksien vaihtelua. Toisaalta tämä antaa varsin realistisen kuvan, mitä sovellusten käyttäminen todellisuudessa on.

Testitapauksen suorittajan toiminnassa esiintyy luonnollista vaihtelua suoritusten kesken. Tällä inhimillisellä tekijällä on vaikutusta jonkin verran suoritusten kokonaisaikoihin. Arvioitaessa yksittäisten metodien ja vaiheiden toiminnallisuutta, jotka suoritetaan sovelluksen sisäisinä prosesseina, ei käyttäjän toimet vaikuta näiden mittausten tuloksiin.

7 TULOKSET

7.1 Siirrettävän tiedon määrä

Tarkasteltaessa siirrettävän tiedon määrää (KUVIO 20) huomataan, että käytettäessä laitteen verkkoselainta, siirrettävän tiedon määrä on yli puolet pienempi (5107 B) kuin käytettäessä MIDlet-sovelluksia (JSR 172 10966 B, kSOAP 11794 B ja kSOAP2 11087 B). MIDlet-sovelluksissa käytettävien SOAP-viestien monimutkaiset XML-rakenteet kasvattavat siirrettävän tiedon määrää. Tämä on huomattavissa varsinkin eniten tietoa sisältävästä getProjects-vaiheessa, jossa XML-rakenteesta aiheutuvan tiedon määrän kasvu kertautuu.

Eri SOAP-sovellusliittymiä vertailtaessa erot ovat huomattavasti pienempiä. Käytettävät SOAP-sovellusliittymät muodostavat jokainen rakenteeltaan hieman erilaiset SOAP-kutsuviestit, jolloin viestien koot poikkeavat aavistuksen toisistaan. Listauksessa 4 on esitetty getTasks-metodiin liittyvät SOAP-pyynnöt. kSOAP-sovellusliittymän uusi versio (kSOAP2) pienentää siirrettävän tiedon määrää 707 tavulla verrattuna kSOAP:n ensimmäiseen versioon (kSOAP). JSR 172 -sovellusliittymää käytettäessä siirrettävän tiedon yhteismäärä on SOAP-sovellusliittymistä pienin (10966 B).

LISTAUS 4. Sovellusliittymien tuottamat SOAP-pyynnöt getTasks-metodin kutsussa.

JSR 172

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tns="http://eviaproject.org/types">
  <soap:Body>
    <tns:Phase_id>
      <phase_id xmlns="">5900</phase_id>
    </tns:Phase_id>
  </soap:Body>
</soap:Envelope>
```

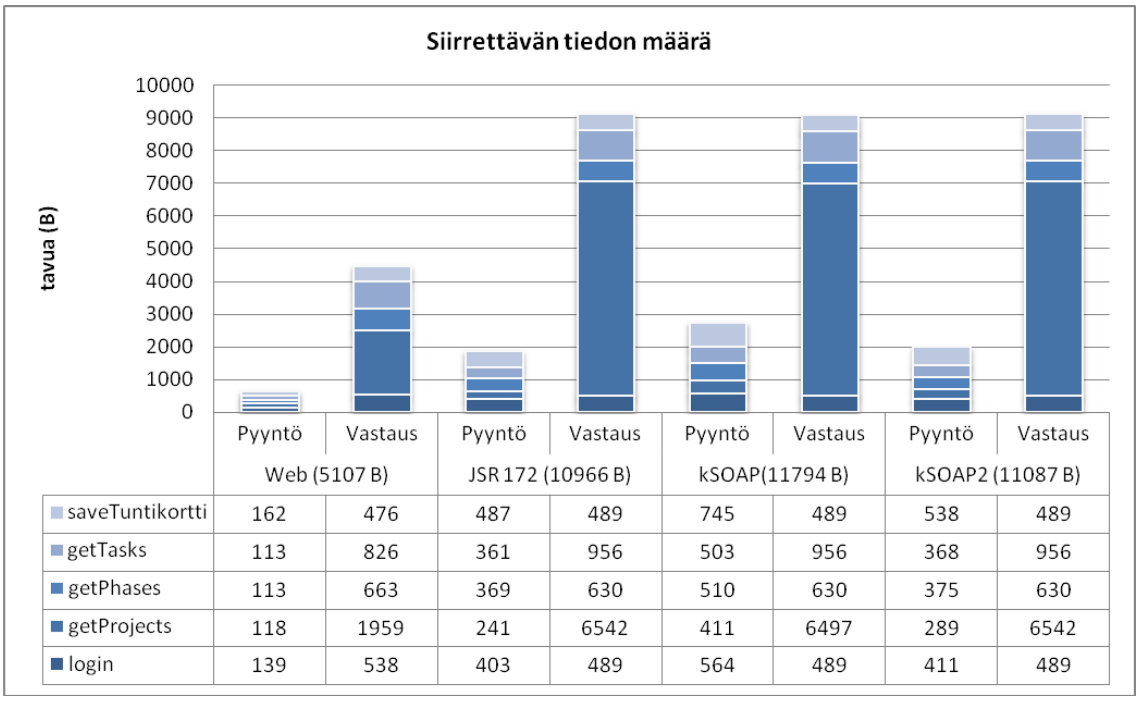
kSOAP2

```
<v:Envelope xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns:d="http://www.w3.org/2001/XMLSchema"
xmlns:c="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:v="http://schemas.xmlsoap.org/soap/envelope/">
  <v:Header />
  <v:Body>
    <n0:getTasks id="o0" c:root="1" xmlns:n0="typens:getTasks">
      <phase_id i:type="d:int">5900</phase_id>
    </n0:getTasks>
  </v:Body>
```

```
</v:Envelope>

kSOAP
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <getTasks xmlns="typens:getTasks" id="o0" SOAP-ENC:root="1">
      <phase_id xmlns="" xsi:type="xsd:int">5900</phase_id>
    </getTasks>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Palvelimen tuottamat vastausviestit ovat sisällöltään eri SOAP-sovellusliittymiä käytettäessä täysin identtiset. Ainoa pieni ero tiedon määrän koossa syntyy getProjects-kutsun vastausviestissä, jossa kSOAP-sovellusliittymälle viestin sisältö esitetään erikoismerkkien oikeinnäkyvyyden vuoksi ISO-8859-1-merkistöllä UTF-8-merkistön sijaan. ISO-8859-1-merkistön käyttö pienentää viestin kokoa 45 tavulla.



	Web (5107 B)		JSR 172 (10966 B)		kSOAP(11794 B)		kSOAP2 (11087 B)	
	Pyyntö	Vastaus	Pyyntö	Vastaus	Pyyntö	Vastaus	Pyyntö	Vastaus
saveTuntikortti	162	476	487	489	745	489	538	489
getTasks	113	826	361	956	503	956	368	956
getPhases	113	663	369	630	510	630	375	630
getProjects	118	1959	241	6542	411	6497	289	6542
login	139	538	403	489	564	489	411	489

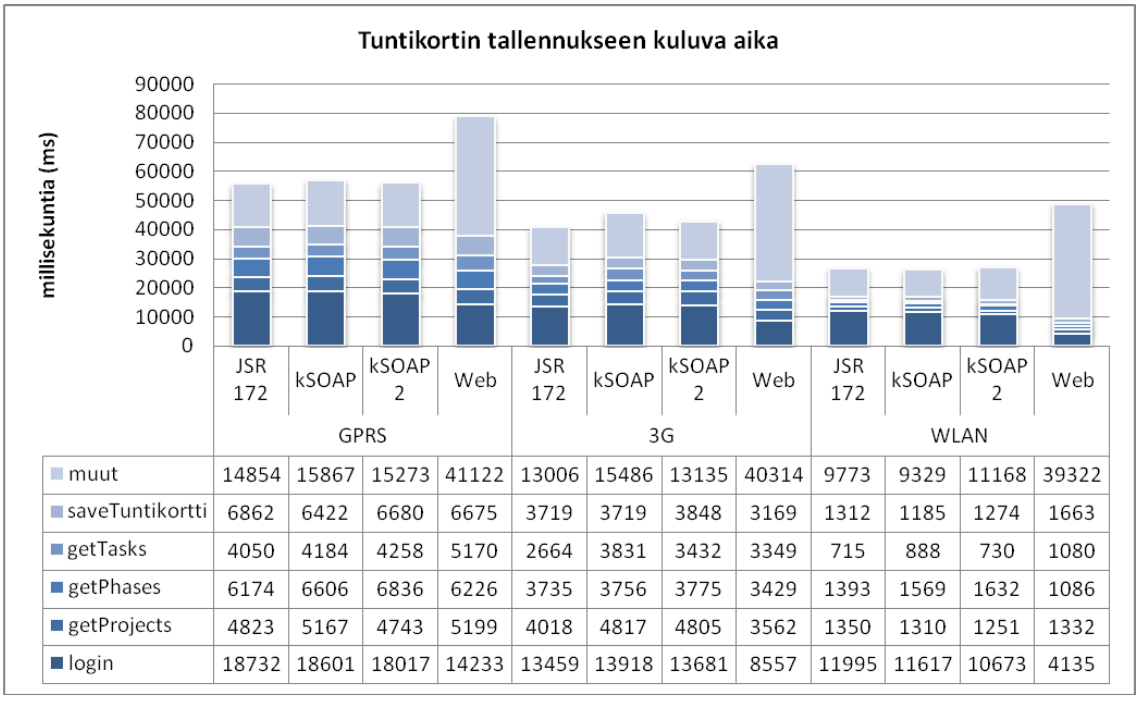
KUVIO 20. Tallennusprosessin aikana siirrettävän tiedon määrä suoritusvaiheittain ositettuna.

7.2 Tuntikortin tallennukseen kulunut kokonaisaika

Kuviossa 21 on esitetty tuntikortin tallentamiseen kuluneiden kokonaisaikojen keskiarvot sovellustyypeittäin ja yhteysmuodoittain. Suoritusaikojen keskiarvot on ositettu jokaisen osavaiheen (login, getProjects, getPhases, getTasks ja save-Tuntikortti) keskiarvoiseen suoritusaikaan. Kohta ”Muut” ei varsinaisesti ole mikään suoritusvaihe, vaan se sisältää kaiken muun käyttäjän toimintaan kuluvan keskiarvoisen ajan, jota ei ole sisällytetty osavaiheisiin, mutta jotka kuitenkin huomioidaan kokonaisajassa. Tällaisia toimenpiteitä ovat esimerkiksi valintojen suorittaminen alasvetovalikoista, tietojen kirjoittaminen sekä selainta käytettäessä osoittimen liikuttaminen näytöllä. MIDlet-sovelluksissa login-vaiheeseen sisältyy yhteysmuodon valitseminen. Web-selainta käytettäessä yhteysmuodon valinta sisältyy kohtaan ”Muut”. Yhteysmuodon valinnassa käytetään laitteen omaa sisäistä dialogia, joten siihen kuluva aika on sekä MIDlet-sovelluksia että selainta käytettäessä identtinen.

Tuloksista selviää, että tuntikortin tallennus on MIDlet-sovelluksia käyttäen selvästi nopeampaa kuin Web-selainta käytettäessä. MIDlet-sovelluksilla kuluu GPRS-yhteydellä tuntikortin tallentamiseen aikaa noin 56 sekuntia ja Web-sovelluksella noin 79 sekuntia. WLAN-yhteyttä käytettäessä MIDlet-sovelluksilla tallentamiseen kuluu aikaa noin 26 sekuntia ja Web-sovelluksella noin 49 sekuntia. WLAN-yhteyttä käytettäessä selaimella täytettävän tuntikortin tallentamiseen kuluu aikaa miltei kaksinkertainen määrä kuin MIDlet-sovelluksilla.

Tarkasteltaessa tallennusprosessin jakaantumista eri osavaiheiden kesken voidaan havaita, että login- ja Muut-vaiheisiin kuluu suurin osa kokonaisajasta. Yllättävää sen sijaan on, että tiedonsiirtomäärältään suurimpaan vaiheeseen getProjects-vaiheeseen kuluu GPRS- ja WLAN-yhteysmuodoilla jopa aavistuksen vähemmän aikaa kuin getPhases-vaiheeseen.



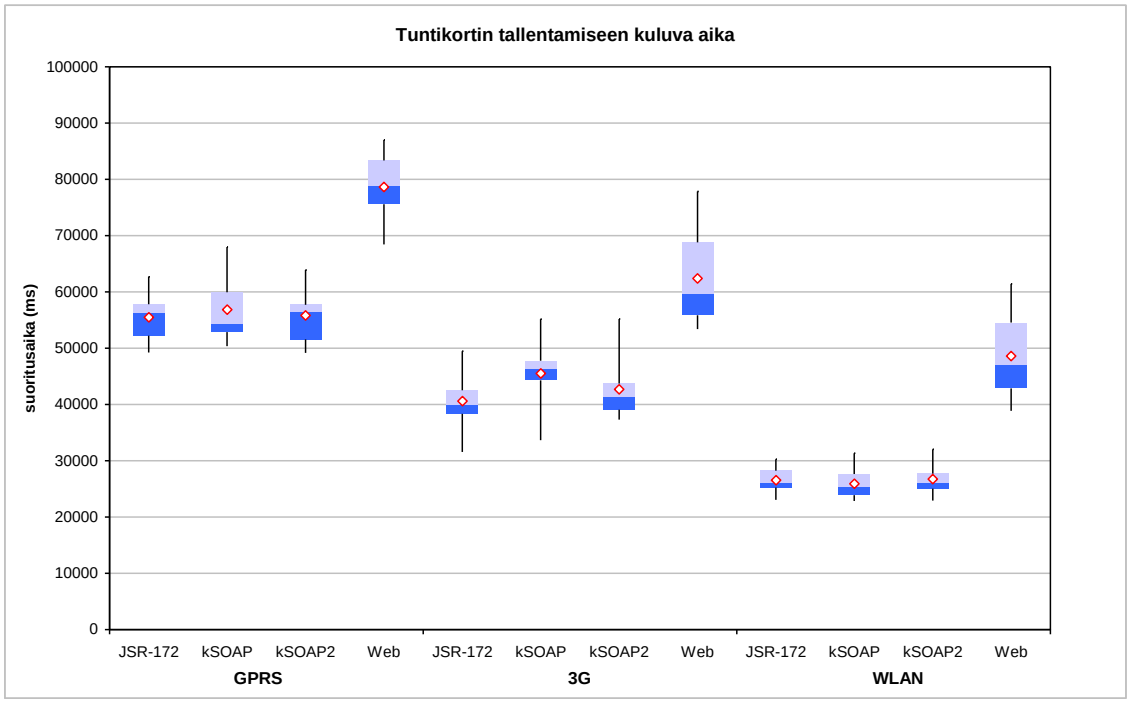
	GPRS				3G				WLAN			
	JSR 172	kSOAP	kSOAP 2	Web	JSR 172	kSOAP	kSOAP 2	Web	JSR 172	kSOAP	kSOAP 2	Web
muut	14854	15867	15273	41122	13006	15486	13135	40314	9773	9329	11168	39322
saveTuntikortti	6862	6422	6680	6675	3719	3719	3848	3169	1312	1185	1274	1663
getTasks	4050	4184	4258	5170	2664	3831	3432	3349	715	888	730	1080
getPhases	6174	6606	6836	6226	3735	3756	3775	3429	1393	1569	1632	1086
getProjects	4823	5167	4743	5199	4018	4817	4805	3562	1350	1310	1251	1332
login	18732	18601	18017	14233	13459	13918	13681	8557	11995	11617	10673	4135

KUVIO 21. Tuntikortin tallennukseen kuluvien aikojen keskiarvot millisekunteina vaiheittain ositettuna.

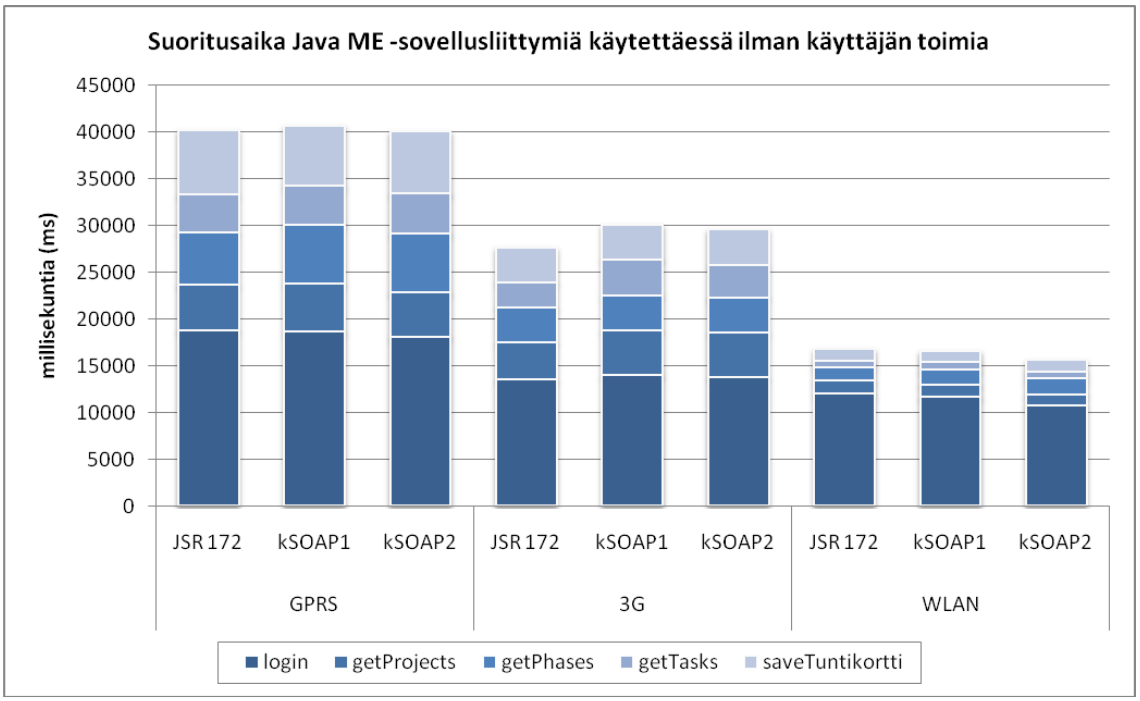
Suurin ero vaiheiden suoritusajoissa Web-sovelluksen ja MIDlet-sovellusten välillä syntyy ”Muut”-kohdassa. Selainta käytettäessä käyttäjän suorittamiin toimenpiteisiin kuluu jopa yli kolminkertainen määrä aikaa. Web-sovelluksella tallennettavan tuntikortin suoritusajan suurempi hajonta (KUVIO 22) selittyy inhimillisistä tekijöistä. Käyttäjän tekemiin toimenpiteisiin kuluu hiukan eri määrä aikaa eri testauskerralla. Lisäksi sovelluksen käytön oppimisvaikutukset ovat näkyvämmät, koska käyttäjä joutuu tekemään enemmän toimenpiteitä käyttöliittymässä kuin MIDlet-sovelluksissa.

MIDlet-sovellusten kesken suoritusajat ovat hyvin tasaiset. Ainoastaan 3G-yhteysmuodolla erot minimiajoissa ovat useita sekunteja (JSR 172 31,5 s ja kSOAP2 37,3 s). Myös hajonta on 3G-yhteydellä muita yhteysmuotoja suurempi.

Vertailtaessa MIDlet-sovelluksia keskenään ilman käyttäjän tekemiä toimenpiteitä, suoritusajoissa ei ole havaittavissa suuria eroja, jotka säilyisivät eri yhteysmuodoilla (KUVIO 23). kSOAP2 vaikuttaisi olevan aavistuksen (3,2 s - 4,8 s) nopeampi kuin kSOAP eri yhteysmuotoja käytettäessä.



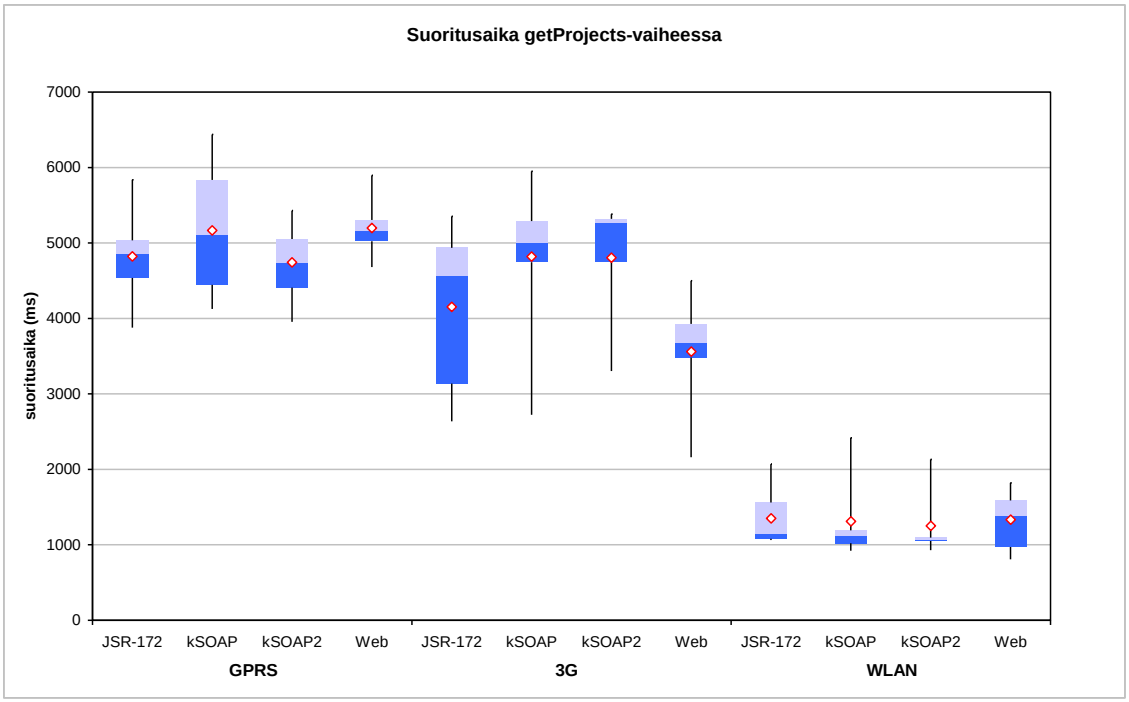
KUVIO 22. Tuntikorttien tallentamiseen kuluvien aikojen jakautuminen kvartiileittain yhteysmuodoittain ja sovellustyypeittäin jaoteltuina (keskiarvot merkitty vinoneliöillä).



KUVIO 23. Java ME -sovellusliittymien suoritusajat tuntikortin tallennusprosessissa ilman käyttäjän toimia.

7.3 Suoritusaika getProjects-vaiheessa

Tarkasteltavaksi valitun getProjects-vaiheen suoritusaikojen erot ovat eri asiakassovellusten kesken pienet, pois lukien kSOAP:n ja kSOAP2:n suoritus ajat 3G-yhteydellä (KUVIO 24). Java ME -sovellusliittymien suoriutumisnopeuksissa ei ole muilla yhteysmuodoilla suuria eroja. 3G- ja WLAN-yhteysmuodoilla Web-sovelluksella on pienimmät minimiarvot (2,1 s ja 0,8 s), mutta vaiheen suorittaminen ei näyttäisi olevan erityisesti nopeampaa, vaikka siirrettävän tiedon määrä on vähäisempi (ks. luku 7.1). kSOAP-sovellusliittymiä käytettäessä suoritusaikojen keskiarvot ovat 3G-yhteydellä suurimmat (4,8 s), miltei GPRS:n tasolla. Vaihtelut yhteyksien muodostamisessa ja yhteyksissä ovat myös suurimmat 3G-yhteydellä. Muita yhteyksiä käytettäessä erot sovellustyyppien välillä ovat huomattavasti pienempiä.



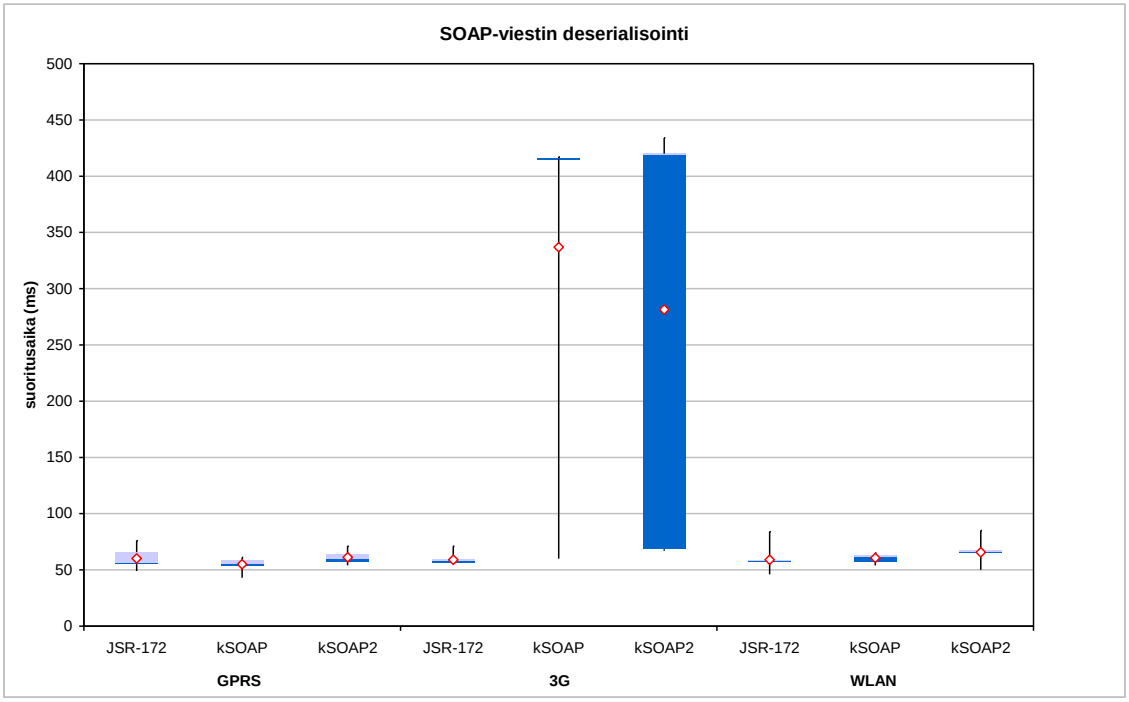
KUVIO 24. getProjects-vaiheen suoritusaikojen jakautuminen kvartiileittain sovellustyypeittäin ja yhteysmuodoittain jaoteltuna (keskiarvot esitetty vinoneliöillä).

7.4 SOAP-viestien deserialisointi

SOAP-viestien deserialisointiin kuluvat ajat ovat eri sovellusliittymien kesken erittäin tasaiset kahta poikkeusta lukuun ottamatta. 3G-yhteydellä kSOAP:n ja kSOAP2:n suoritukset ovat jostain syystä poikkeavat. Maksimiarvot ja suori-

tusaikojen hajonnat ovat erittäin suuria verrattuna muihin yhteysmuotoihin ja JSR 172:n suoritus arvoihin. Vastaava ilmiö ei toistu muilla yhteysmuodoilla. On mahdollista, että sovelluksen ajon aikana, jokin korkeamman prioriteetin säie on varannut suorittimen resursseja, jolloin sovelluksen suorittaminen on keskeytynyt hetkeksi. Viivästys on ollut joka tapauksessa vain muutamia sekunnin kymmenesosia, jolloin vaikutus suorituksen kokonaisaikaan on ollut marginaalinen.

Keskimäärin SOAP-rakenteen muunto Java-rakenteeksi kestää noin 0.06 sekuntia (KUVIO 25). GPRS-yhteydellä getProjects-metodin kutsuun kuluu aikaa noin 5 sekuntia, joten vastauksen deserialisointiin kuluu vain erittäin pieni osa tuosta ajasta.

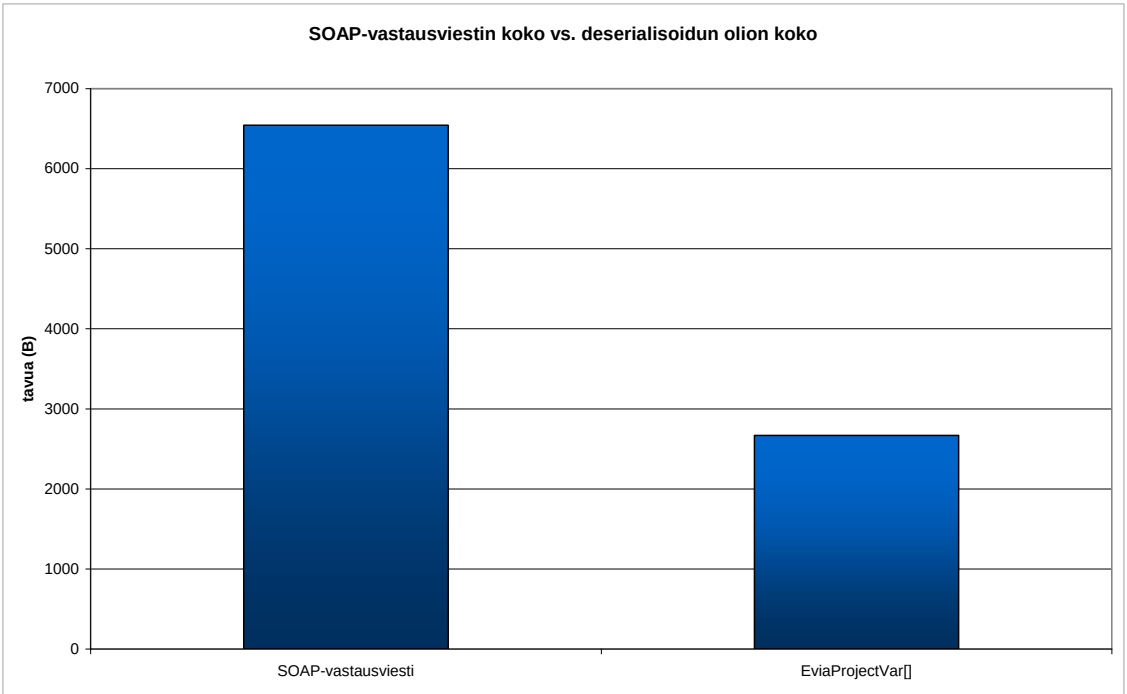


KUVIO 25. SOAP-viestin deserialisointiin kuluvien suoritusaikojen jakautuminen kvartiileittain sovellustyypeittäin ja yhteysmuodoittain jaoteltuina (keskiarvot merkitty vinoneliöillä).

7.5 Deserialisoidun olion koko

Aiemmin selvisi, että JSR 172 -sovellusliittymää käytetettäessä getProjects-metodin SOAP-palautusarvon koko 6542 tavua. Kun viesti on deserialisoitu Java-rakenteeksi (EviaProjectVar[]), kutistuu tietomäärä 2668 tavuun. Tässä tapa-

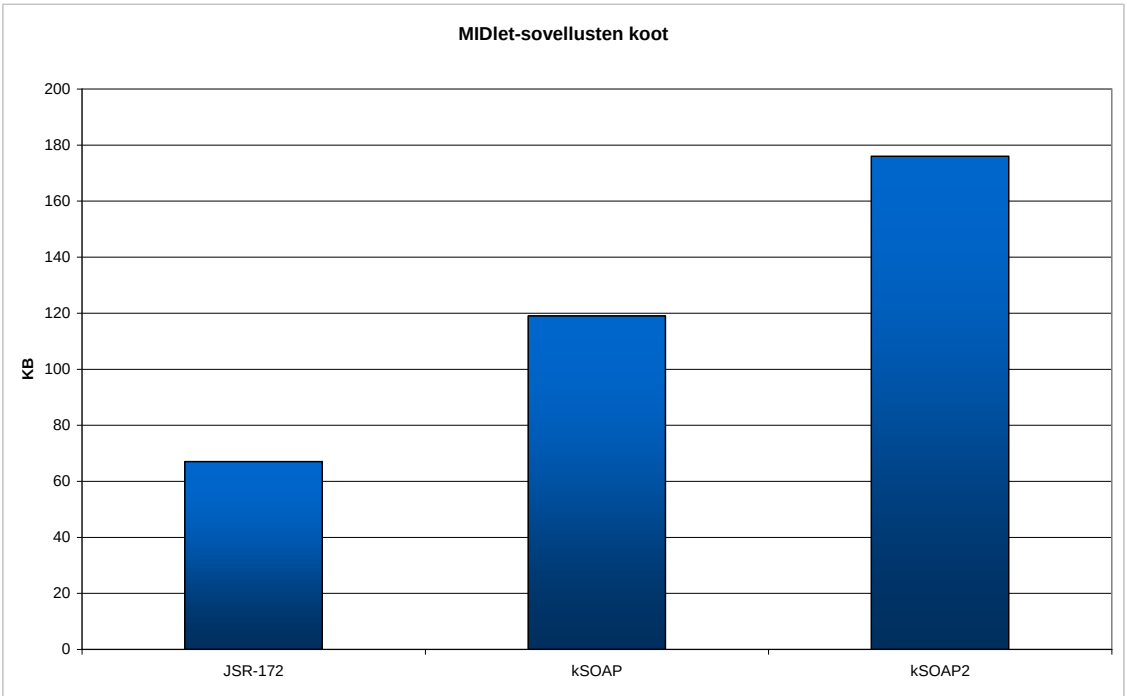
uksessa SOAP-viestin koko on siis noin 2,5 kertainen Java-tietorakenteen kokoon verrattuna (KUVIO 26).



KUVIO 26. SOAP-tietorakenteen koko vs. Java-tietorakenteen koko getProjects-metodissa

7.6 MIDlet-sovelluksen koko

MIDlet-sovelluksen fyysinen koko saadaan selville laitteen tiedoista. kSOAP:n ensimmäinen versio kasvattaa sovelluksen kokoa 52 kilotavulla (JSR 172 67 KB, KSOAP 119 KB) (KUVIO 27). kSOAP-kirjaston versio 2 kasvattaa sovelluksen kokoa vielä 57 kilotavua lisää, jolloin sovelluksen lopullinen koko on 176 kilotavua.



KUVIO 27. MIDlet-sovellusten koot kilotavuina.

8 POHDINTA

Tässä tutkielmassa selvitettiin Web Services -tekniikoitten käytön mahdollisuuksia ja vaikutuksia mobiilisovelluksiin erityisesti Java ME -ympäristössä. Web Services -tekniikat liittyvät palvelukeskeiseen ohjelmistojen integrointiin, joten pohjimmiltaan kyse on siis ohjelmistojen integroinnista. Ohjelmistot kannattaa integroida erityisen integraatioväliohjelmiston avulla (ks. esim. Linthicum 2000), joka huolehtii tiedon siirrosta eri järjestelmien välillä, tietojen muunnoksista ja integraatioprosessin kokonaiskontrolloinnista (Tähtinen 2005). Lisäksi se tarjoaa jonkinlaisen esitystavan integraatioprosessien tarkkailuun. Integroitavat järjestelmät tarjoavat rajapinnat, joiden kautta integraatioväliohjelmisto hyödyntää järjestelmien resursseja.

Ohjelmistoja voidaan integroida tarpeesta ja laajuudesta riippuen eri lähestymistavoin. Lähestymistapa ottaa kantaa, millä tavoin järjestelmäintegraatio toteutetaan. Lähestymistapa voi olla puhtaasti informaatiokeskeinen, jolloin ainoastaan hyödynnetään toisten ohjelmistojen tietokantoja tai muita pysyviä tietoja. Kuitenkin lähestymistavat, joissa hyödynnetään myös toisten järjestelmien toiminnallisuuksia, tuovat pidemmällä aikavälillä enemmän arvoa (Linthicum 2004).

Yksi ohjelmistojen integroinnin lähestymistapa on palvelukeskeinen ohjelmistojen integrointi. Palvelukeskeisessä lähestymistavassa järjestelmän liiketoimintaprosessit ja resurssit kuvataan sekä toteutetaan laitteistoriippumattomina palveluina. Integraatioväliohjelmistona toimii Enterprise Service Bus (ESB), joka huolehtii yhteyksistä eri ohjelmistojen välillä ja integraatioprosessien kontrolloinnista (ks. esim. Keen ym. 2004).

Palvelukeskeinen lähestymistapa asettaa vaatimukset Web Services -tekniikoille. Niiden on mahdollistettava palveluiden kuvaus ja rekisteröinti sekä laitteistoriippumaton, turvallinen ja luotettava viestintä (Ferguson ym. 2003). Näihin haasteisiin on tarjolla lukuisia ratkaisuja. Web Services -tekniikoita ovat kehittäneet pääasiassa ohjelmistotoimittajat, minkä vuoksi joihinkin ongelmiin on tarjolla useitakin eri ratkaisumalleja. OASIS-, WS-I- ja W3C-organisaatiot tekevät kuitenkin töitä tekniikoiden standardoimiseksi ja yhteensopivuuden parantamiseksi. Siitä huolimatta ratkaisumallien runsaus hidastaa tekniikoiden yleistymistä ja kypsymistä.

Mobiilien päätelaitteiden ohjelmointi on haasteellista laitteiden rajoittuneiden resurssien johdosta. Päätelaitteiden kehityksen johdosta on voitu selvittää Web Services -tekniikoiden käytön mahdollisuuksia liikkuvassa tietojenkäsittelyssä. Mobiileissa päätelaitteissa voidaan hyödyntää Web Services -tekniikoita kahdella eri tavalla. Tekiikoita voidaan hyödyntää suoraan mobiilisovelluksissa (laajennetun langattoman Internetin malli) tai palvelinpäässä, jolloin tiedot esitetään päätelaitteen Web-selaimen avulla käyttäjälle (langattoman portaalin malli) (Yuan ja Long 2002).

Langattoman laajennetun Internetin mallissa voidaan minimoida käyttäjän tiedonsyöttö, yksinkertaistaa prosesseja, helpottaa sovelluksen käyttöä ja pienentää tiedonsiirrosta aiheutuvaa odotusaikaa. Toisaalta tiedon prosessoinnin tapahtuessa päätelaitteessa, prosessorin käyttö ja muistin käyttö kasvaa aiheuttaen suuremman virran kulutuksen. (Zahreddine ja Mahmoud 2005.) Langattoman portaalin mallin avulla Web Services -tekniikoita voidaan käyttää laajemmin palvelinkerroksessa, jossa tekniikat ovat kypsempiä.

Web Services -tekniikat sopivat kieli- ja alustariippumattomuutensa vuoksi hyvin mobiilisovellusten käytettäväksi (ks. esim. Steele 2003). Web Services-palveluita kutsumalla sovellusten fyysinen koko pienenee, kun kaikkia tietoja ei tarvitse tallentaa sovellukseen. Toisaalta viestien XML-rakenne kasvattaa siirrettävän tiedon ja tiedon prosessoinnin määrää (ks. esim. Zahreddine ja Mahmoud 2005).

Java ME -ympäristö on suunniteltu resursseiltaan rajoittuneiden laitteiden sovellusalustaksi. Web Services -palveluiden hyödyntäminen Java ME -ympäristössä on mahdollista esimerkiksi JSR 172- ja kSOAP-sovellusliittymien avulla. Sovellusliittymien hyödyntäminen tuo sovelluksille Web Services -tekniikoiden käytön hyödyt Java ME -ympäristössä.

Java ME:n Web Services -tekniikat eivät tällä hetkellä tue SOA-mallin mukaista integraatioratkaisua. Sovellusliittymissä ei ole tukea UDDI-rekistereille, WS-*-spesifikaatioille eikä todentamis- ja identifiointi-palveluille. Lisäksi sovellusliittymät eivät tue sovellusten toimimista Web Services -palveluiden tarjoajina. Web Services -tekniikoista ainoastaan SOAP-protokolla ja XML:n käsittely ovat tuettuja. Nämä mahdollistavat sovellusten välisen viestityksen, mutta pelkästään SOAP- ja WSDL-spesifikaatiota käyttämällä ajaudutaan kaksipisteyhteys-tyyliseen integraatioon (Keen ym. 2004).

Kehitteillä on Service Connection API (JSR 279) -spesifikaatio, jonka lähtökohtana on laajentaa Web Services -tekniikoiden tukea Java ME -ympäristössä. Spesifikaatiossa määritetään tuki UDDI-rekistereille ja dynaamisille palveluille. Lisäksi sovellusliittymä mahdollistaa eri sovelluskehysratkaisujen käytön, joiden puitteissa voidaan toteuttaa tuki WS-*-lisäkomponenteille sekä identifiointi- ja todentamisratkaisuille. MIDlet-sovellukset voisivat toimia palveluiden tarjoajina. Spesifikaatio on vasta arviointivaiheessa, joten sen lopullinen sisältö ei ole vielä selvillä.

Vertailtaessa langattoman portaalin mallia ja langattoman laajennetun Internetin mallia kävi ilmi, että käytettäessä Web Services -tekniikoita asiakaskerroksessa, suoritusaika on huomattavasti pienempi kuin selainpohjaisella sovelluksella. Ero selittyy pääasiassa sillä, että selainta käytettäessä käyttäjä joutuu tekemään enemmän toimenpiteitä käyttöliittymässä. MIDlet-sovelluksissa käyttäjän tiedot voidaan tallentaa sovelluksen muistiin, joten tunnistautumistietoja ei tarvitse kirjoittaa joka kerta. Toisin sanoen tässä toteutuu yksi Web Services -tekniikoiden käytön hyöty – tietojen syötön minimointi.

Kirjallisuuden perusteella voitiin olettaa, että siirrettävän tiedon määrä kasvaa SOAP-protokollaa käytettäessä. Saatujen tulosten perusteella tämä pitikin paikkansa, sillä siirrettävän tiedon määrä oli MIDlet-sovelluksilla noin kaksinkertainen verrattuna HTML-muotoon. getProjects-metodin suorittamisen aikana siirtyneestä tiedosta 60 % oli muuta kuin varsinaista tietosisältöä, toisin sanoen XML:sta aiheutuvia ylimääräisiä merkkejä.

Vaikka tietomäärä oli huomattavasti suurempi MIDlet-sovelluksia käytettäessä, se ei näkynyt tiedonsiirto- ja käsittelyajoissa. Tämä selittyy sillä, että siirrettävän tiedon määrä oli yksittäisen suoritusvaiheen aikana niin pieni, ettei tiedon määrän kasvulla ollut vaikutusta tiedonsiirtoon kuluneeseen aikaan. Nykyisten mobiilien päätelaitteiden prosessorit ovat myös sen verran tehokkaita, että SOAP-viestin muuntaminen Java-tietorakenteeksi sujuu niiltä nopeasti. getProjects-metodin vastausviestin deserialisointiin kului aikaa noin 6 sekunnin sadasosaa, mikä on karkeasti ottaen noin sadasosa tiedonsiirtoon ja -käsittelyyn kuluneesta kokonaisajasta.

Vertailtaessa eri Java ME:n Web Services -sovellusliittymien suoritusaikoja keskenään suuria eroja ei ollut havaittavissa. Sen sijaan eroja löytyy arvioitaessa käytettävän Web Services -sovellusliittymän vaikutuksia sovelluksen kokoon.

JSR 172 -sovellusliittymää käytettäessä sovellusliittymä on toteutettu päätelaitteessa, jolloin ylimääräisiä kirjastoja ei tarvitse liittää sovellukseen. kSOAP:n versiota 2 käytettäessä sovelluksen koko kasvaa yli sadalla kilotavulla.

WLAN-yhteysmuotoa käytettäessä verkkoyhteys on täysin riittävä integroidun ohjelmiston sulavaan käyttämiseen. Yhteys rajoittuu kuitenkin suppealle alueelle, yleensä sisätiloihin, jolloin käytettäessä sovellusta muualla vaihtoehdoiksi jää 3G- ja GPRS-yhteydet. Näillä yhteysmuodoilla SOAP-kutsun suorittamiseen kului aikaa useita sekunteja, jolloin verkon hidastava vaikutus oli selvästi havaittavissa käytön aikana. Tämän vuoksi mobiilisovellusta suunniteltaessa onkin otettava huomioon, milloin verkkoyhteyden käyttäminen on tarkoituksen mukaista. 3G-yhteys osoittautui lisäksi toiminnaltaan epävarmaksi aiheuttaen suurta hajontaa suoritusaikoihin. Jatkossa tätä asiaa tulisikin tutkia tarkemmin, jotta voitaisiin selvittää, mistä tällainen ilmiö aiheutuu.

Tutkimustulosten perusteella voidaan sanoa, että tietomäärän ollessa suhteellisen pieni tehokkuuden kannalta ei ole väliä, mitä Java ME -sovellusliittymää käyttää. Tutkimuksessa käytetyllä aineistolla suuria eroavaisuuksia eri sovellusliittymien välillä ei ilmennyt. Mahdolliset eroavaisuudet SOAP-viestien käsittelyn tehokkuudessa voivat olla havaittavissa suurempia tietomääriä käytettäessä. Mikäli päätelaite tukee JSR 172 -sovellusliittymää, on tämän sovellusliittymän käyttäminen perusteltua sovelluksen koon vuoksi. Mikäli WS-*-spesifikaatiota ei voida tukea asiakaskerroksessa, voidaan tuki näille saavuttaa käyttämällä langattoman portaalin mallia.

Service Connection API (JSR 279) -sovellusliittymän valmiuksia palvelukeskeiseen arkkitehtuuriin tulisi tutkia mahdollisimman pian spesifikaation lopullisen version valmistumisen jälkeen. Sovellusliittymän lopullinen sisältö ei ole vielä valmistunut, joten ei vielä ole täysin varmaa, mitä WS-*-spesifikaatioita se tulee tukemaan.

WS-*-spesifikaatioiden toteutukset lisäävät SOAP-otsikkotietojen määrää, joten olisi mielenkiintoista tutkia, mitä vaikutuksia tällä tiedon määrän kasvulla on suoritusaikaan. Näiden spesifikaatioiden vaikutuksia voitaisiin tutkia esimerkiksi toteuttamalla ne kSOAP-sovellusliittymän päälle.

Rajoituksista huolimatta tämä tutkielma antaa viitettä, että laajennetun langattoman Internetin malli on tehokkaampi kuin langattoman portaalin malli. Web

Services -tekniikoita on mahdollista hyödyntää mobiilissovelluksissa ja laitteiden ominaisuuksien hyödyntämisen ansiosta sovellusten käyttäminen on nopeampaa. Tiedonmäärän kasvullakaan ei näytä olevan suurta vaikutusta suoritusaikaan.

Tässä tutkielmassa langattoman portaalin mallia testattiin ainoastaan päätelaitteen omalla Web-selaimella. Jatkotutkimuksissa mallia voitaisiin testata muillakin selaimilla kuin pelkästään testattavassa päätelaitteessa olleella oletusselaimella. Näin saataisiin kattavampi kuva langattoman portaalin mallin tehokkuudesta.

Yksi liikkuvan tietojenkäsittelyn tavoite on yhdistää mobiilit päätelaitteet saumattomasti osaksi organisaation kokonaistietojenkäsittelyä. Kun mobiilisovelluksissa voidaan hyödyntää palvelukeskeisen arkkitehtuurin mahdollistavia Web Services -tekniikoita, ollaan entistä lähempänä tätä tavoitetta. Integraatioväliohjelmiston ei tarvitsisi enää huolehtia, onko asiakassovellus mobiilisovellus vai jokin järeämpi sovellus, vaan molemmat voisivat hyödyntää samaa palvelurajapintaa. Tämä tarkoittaisi suurta edistystä siinä, kuinka laajasti mobiilit päätelaitteet voivat hyödyntää yritysjärjestelmien resursseja. Web Services -tekniikat tarjoavat laitteistoriippumattoman, turvallisen ja luotettavan pääsyn yritysjärjestelmien tietoihin ja toiminnallisuuksiin. Mobiilien päätelaitteiden avulla näihin tietoihin ja toimintoihin pääsy olisi ajasta ja paikasta riippumatonta. Tulevaisuudessa tulisikin keskittyä Web Services -tekniikoiden yhteensopivuuden parantamiseen ja täysimääräiseen hyödyntämiseen liikkuvassa tietojenkäsittelyssä ja Java ME -ympäristössä.

LÄHDELUETTELO

Apte N., Deutsch K. ja Jain R., 2005. Teoksessa Special interest tracks and posters of the 14th international conference on World Wide Web. ACM Press, 1178.

Arokoski A., Jääskeläinen J., Kontio M., Köykkä S., Tervo T., Vierimaa K., 2002. Mobiiliteknologiat. IT Press.

Arsanjani A., Hailpern B., Martin J. ja Tarr P., 2003. Web Services Promises and Compromises. Queue 1(1), 48-58.

Ashri R., Atkinson S., Ayers D., Hagling M., Ray B., Machin R., Nashi N., Taylor R. & Wiggers C., 2001. Java Mobile Programming. Wrox Press.

Burner M., 2003. The Deliberate Revolution - Creating Connectedness with XML Web Services. Queue 1(1), 28-56.

Campadello S., 2003. Middleware Infrastructure for Distributed Mobile Applications. Helsingin yliopisto, väitöskirja.

Endrei M., Ang J., Arsanjani A., Chua S., Comte P., Krogdahl P., Luo M., Newling T., 2004. Patterns: Service-Oriented Architecture and Web Services [online]. IBM [viitattu 3.4.2008]. Saatavilla www-osoitteesta: <http:/ / www.redbooks.ibm.com/ redbooks/ pdfs/ sg246303.pdf>

Farley P. ja Capp M., 2005. Mobile Web Services. BT Technology Journal 23(3), 202-213.

Ferguson D., Storey T., Lovering B. ja Shewchuk J., 2003. Secure, Reliable, Transacted Web Services [online]. IBM [viitattu 7.5.2008]. Saatavilla www-osoitteesta: <http:/ / www-128.ibm.com/ developerworks/ webservices/ library/ ws-securtrans/ >.

Ferris C. ja Farrel J., 2003. What Are Web Services? Communtications of the ACM 46(6), 31.

Hanslo W. ja MacGregor K., 2004. The Efficiency of XML as an Intermediate Data Representation for Wireless Communication. Teoksessa Proceedings of the 2004 annual research conference of the South African institute of

computer scientists and information technologists on IT research in developing countries SAICSIT '04. ACM Press, 279-283.

Helal S., 2002. Pervasive Java. IEEE Pervasive Computing 1(1), 82-85.

High Jr., R., Kinder S. ja Graham S., 2005. IBM SOA Foundation – Architecture Overview [online]. IBM [viitattu 7.5.2008]. Saatavilla www-osoitteesta: <http:/ / download.boulder.ibm.com/ ibmdl/ pub/ software/ dw/ webservices/ ws-soa-whitepaper.pdf>.

Hirsch F., Kemp J. ja Ilkka J., 2006. Mobile Web Services: Architecture and Implementation. Wiley.

Hjelm J., 2000. Designing wireless information services. Wiley.

Hogg K., Chilcott P., Nolan M. ja Srinivasan B., 2004. An Evaluation of Web Services in the Design of a B2B Application. Teoksessa 27th Australian Computer Science Confrence, The University of Otago, Dunedin, New Zealand. Confrences inf Research and Practice in Information Technology, Vol 26, Estivill-Castro, 331-340.

Hohpe G. ja Woolf B., 2004. Enterprise Integration Patterns - Designing, Building and Deploying Messaging Solutions. Addison-Wesley.

IBM, 2007. Standards and Web services [online]. IBM [viitattu 3.4.2008]. Saatavilla www-osoittteesta: <http:/ / www.ibm.com/ developerworks/ webservices/ standards/ >.

Ibrahim M. ja Long G., 2007. Service-Oriented Architecture and Enterprise Architecture, Part 1: A framework for understanding how SOA and Enterprise Architecture work together [online]. IBM [viitattu 7.5.2008]. Saatavilla www-osoitteesta: <http:/ / www.ibm.com/ developerworks/ webservices/ library/ ws-soa-enterprise1/ >.

Juric M.B., Basha S.J., Leander R., Nagappan R., 2001. Professional J2EE EAI. Wrox Press Ltd.

Keen M., Acharaya A., Bishop S., Hopkins A., Milinski S., Nott C., Robinson R., Adams J. ja Verschueren P., 2004. Patterns: Implementing an SOA Using

an Enterprise Service Bus [online]. IBM [viitattu 4.3.2008]. Saatavilla www-osoitteesta:
<http:/ / www.redbooks.ibm.com/ redbooks/ pdfs/ sg246346.pdf>

Kleijnen S. ja Raju S., 2003. An Open Web Services Architecture. Queue 1(1), 38-46.

Knudsen J., 2003. Understanding MIDP 2.0's Security Architecture [online]. Sun Microsystems [viitattu 7.5.2008]. Saatavilla www-osoitteesta: < http:/ / developers.sun.com/ mobility/ midp/ articles/ permissions/ >

Kontio M., 2002. Mobile Java with J2ME. IT Press.

kSOAP Project, 2003. The home of the kSOAP at Enhydra.org [online][viitattu 2.3.2008]. Saatavilla www-osoittesta: <http:/ / ksoap.objectweb.org/ >.

kSOAP2 Project, 2006. kSOAP 2[online][viitattu 2.3.2008]. Saatavilla www-osoitteesta: <http:/ / ksoap2.sourceforge.net/ >.

kXML Project, 2005. kXML News [online][viitattu 2.3.2008]. Saatavilla www-osoitteesta: <http:/ / kxml.sourceforge.net/ >.

Lee J., Siau K., Hong S., 2003. Enterprise Integration with ERP and EAI. Communication of ACM 46(2), 54-60.

Liberty Alliance Project, 2008. The Liberty Alliance [online][viitattu 27.3.2008]. Saatavilla www-osoitteesta <http:/ / www.projectliberty.org/ >.

Linthicum D.S., 2000. Enterprise Application Integration. Addison-Wesley.

Linthicum D.S., 2001. B2B Application Integration: e-Business-Enable Your Enterprise. Addison-Wesley.

Linthicum D.S., 2004. Next Generation Application Integration: From Simple Information to Web Services. Addison-Wesley.

Machado A.C.C ja Ferraz C.A.G, 2005. Guidelines for performance evaluation of web services. Teoksessa ACM International Conference Proceeding Series; Vol. 125. ACM, 1-10.

Mahmoud Q., 2003. Future Java Technology for the Wireless Services Industry [online]. Sun Microsystems [viitattu 7.5.2008]. Saatavilla www-osoitteesta: < http:/ / developers.sun.com/ mobility/ midp/ articles/ j2mefuture/ >.

Mahmoud Q., 2005. Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI) [online]. Sun Microsystems [viitattu 7.5.2008]. Saatavilla www-osoitteesta: <http:/ / java.sun.com/ developer/ technicalArticles/ WebServices/ soa/ index.html>.

McLaughlin B., 2002. Java & XML. O´Reilly.

Nokia, 2002. Mobile Internet Technical Architecture - Technologies and Standardization. IT Press.

OASIS, 2004. Universal Description, Discovery and Integration v3.0.2 (UDDI).

OASIS, 2006. WS-Security Core Specification 1.1.

OASIS, 2007a. Web Services Business Process Execution Language Version 2.0.

OASIS, 2007b. WS-Trust 1.3.

OASIS, 2007c. Web Services Reliable Messaging (WS-1 ReliableMessaging) Version 1.1.

OASIS, 2007d. Web Services Coordination (WS-Coordination) Version 1.1.

Ort E., 2005. Service-Oriented Architecture and Web Services: Concepts, Technologies, and Tools [online]. Sun Microsystems [viitattu 7.5.2008]. Saatavilla www-osoitteesta: <http:/ / java.sun.com/ developer/ technicalArticles/ WebServices/ soa2/ >.

Ortiz C.E., 2002. A Survey of J2ME Today [online]. Sun Microsystems [viitattu 7.5.2008]. Saatavilla www-osoitteesta: <http:/ / developers.sun.com/ mobility/ getstart/ articles/ survey/ >.

Ortiz C.E., 2006a. Mobile Services Architecture Specification [online]. Sun Microsystems [viitattu 7.5.2008]. Saatavilla www-osoitteesta: <http:/ / developers.sun.com/ mobility/ midp/ articles/ msaintro/ >.

Ortiz C.E., 2006b. Understanding the Web Services Subset API for Java ME [online]. Sun Microsystems [viitattu 7.5.2008]. Saatavilla www-osoitteesta: <http:/ / developers.sun.com/ mobility/ midp/ articles/ webservices/ >.

Riggs R, Taivalsaari A ja VandenBrink M, 2001. Programming Wireless Devices with Java 2 Platform, Micro Edition. Addison-Wesley.

Ryan C. ja Rossi P., 2005. Software, Performance and Resource Utilisation Metrics for Context-Aware Mobile Applications. Teoksessa Proceedings of the 11th IEEE International Software Metrics Symposium. IEEE Computer Society, 12.

Sánchez-Nielsen E., Martín-Ruiz Sandra & Rodríguez-Pedrianes J., 2006. An open and dynamical service oriented architecture for supporting mobile services. Teoksessa Proceedings of the 6th international conference on Web engineering, Palo Alto, California, USA. ACM Press, 121-128.

Satyanarayanan M., 1996. Fundamental Challenges in Mobile Computing. Teoksessa Annual ACM Symposium on Principles of Distributed Computing Philadelphia, Pennsylvania, United States, May 23 – 26. ACM Press, 1-7.

Schiller J., 2001. Mobiili tietoliikenne. IT Press.

Steele R., 2003. A Web Services-based System for Ad-hoc Mobile Application Integration. Teoksessa Proceedings of the Interntional Confrence on Information Technology: Computers and Communications (ITCC'03) 28-30 April 2003. IEEE Computer Society, 248-252.

Sun Microsystems, 1999. The Jini™ Architecture Specification (AR).

Sun Microsystems, 2000a. Connected Limited Device Configuration (CLDC) Specification.

Sun Microsystems, 2000b. Mobile Information Device Profile (JSR-37) Specification.

Sun Microsystems, 2002. PDA Profile 1.0 Specification (JSR 75).

Sun Microsystems, 2003a. Information Module Profile (JSR-195) Specification.

Sun Microsystems, 2003b. Connected Limited Device Configuration 1.1 (JSR 139) Specification.

Sun Microsystems, 2004. J2ME Web Services Specification, 1.0 (JSR 172)

Sun Microsystems, 2006a. Mobile Information Device Profile 2.1 (JSR 118) Specification

Sun Microsystems, 2006b. Mobile Information Device Profile 3 (JSR 271) Specification

Sun Microsystems, 2006c. Mobile Service Architecture Specification (JSR 248)

Sun Microsystems, 2006d. Service Connection API for Java ME 0.1 Early Draft Review (JSR 279)

Topley K., 2002. J2ME in a Nutshell. O´Reilly.

Tähtinen S., 2005. Järjestelmäintegraatio : tarve, vaihtoehdot, toteutus. Talentum.

W3C Working Group, 2000. SOAP Messages with Attachments.

W3C Working Group, 2001. Web Services Description Language (WSDL) 1.1.

W3C Working Group, 2004. Web Services Architecture [online]. W3C Working Group [viitattu 7.5.2008]. Saatavilla www-osoitteesta: <http:/ / www.w3.org/ TR/ 2004/ NOTE-ws-arch-20040211/ >.

W3C Working Group, 2006. Web Services Addressing 1.0 - Core.

W3C Working Group, 2007a. SOAP Version 1.2 Part 2: Adjuncts (Second Edition).

W3C Working Group, 2007b. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition).

Web Services Interoperability Organization, 2004. Basic Profile 1.1.

White J., 2001. An introduction to Java 2 micro edition (J2ME); Java in small things. Teoksessa Proceedings of the 23rd international conference on

Software engineering Toronto, Ontario, Canada, May 12 – 19. IEEE Computer Society, 724-725.

Yuan M.J. ja Long J., 2002, Java readies itself for wireless Web services [online]. JavaWorld.com [viitattu 7.5.2008]. Saatavilla www-osoitteesta: <http:/ / www.javaworld.com/ javaworld/ jw-06-2002/ jw-0621-wireless.html>.

Zahreddine W. ja Mahdmoud Q.H, 2005. An Agent-based Approach to Composite Mobile Web Services. Teoksessa Proceedings of the 19th International Confrence on Advanced Information Networking and Applications (AINA'05) Volume 2 28-30 March 2005. IEEE Computer Society, 189-192.