Yritysten yhteistoimintaverkostojen valvonta Web-palveluympäristössä

Juha-Pekka Haataja

Helsinki 22. maaliskuuta 2005
Pro gradu -tutkielma
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Sisältö

Luku 1

Johdanto

Pärjätäkseen globaaleilla markkinoilla yritysten on tehostettava jatkuvasti työvoimansa ja pääomansa tuottavuutta. Eräs toiminnan tehostamiseen tapa on keskittyä niihin asioihin, joissa yritys on tehokkaimmillaan ja osaaminen on korkeimmalla tasolla. Tällöin puhutaan niin sanotuista **ydinliiketoiminnoista**. Ydinliiketoimintojen ulkopuolelle jäävät ja niitä tukevat toiminnot puolestaan pyritään **ulkoistamaan** eli ostetaan muualta.

Ulkoistettu toiminto (voidaan myös käyttää nimitystä **palvelu**) voidaan ostaa yhdeltä alihankkijana toimivalta organisaatiolta, tai vaihtoehtoisesti sen toteuttaa useasta yrityksestä koostuva liiketoimintaverkosto. Yksinkertaisten ja toimialallaan vakiintuneiden palveluiden tapauksessa alihankkijana toimii tyypillisesti vain yksi yritys. Jos palvelu on räätälöidympi, monimutkaisempi, todellisuudessa usean pienemmän palvelun kompositio, hoitaa sen tarjoamisen usein kokonainen vuorovaikuttavien yritysten rypäs eli **liiketoimintaverkosto**. Liiketoimintaverkosto muodostaa **virtuaalisen yrityksen**, joka tarjoaa useasta pienemmästä palvelusta koostuvaa **palvelukokonaisuutta** (käytetään myös nimeä **palvelukompositio**).

Ulkoistaminen on aina luonteeltaan kaksiteräinen miekka. Ulkoistaminenhan on viime kädessä toimintojen fyysistä ja hallinnollista hajauttamista. Ulkoistamisen ongelmat ovat siis luonteeltaan analogisia hajautettuun tietojenkäsittelyyn liittyvien ongelmien kanssa. Molemmissa tapauksissa saatava teoreettinen hyöty voi käytännössä kariutua hajautuksesta väistämättä seuraaviin epävarmuustekijöihin ja hallinnolliseen raskauteen. On siis tärkeä ymmärtää mitä ja miten ulkoistetaan.

Ulkoistaminen on temporaalisesti syklinen ilmiö. Rajua ulkoistamiskautta on usein seurannut liiketoimintojen uusi keskittäminen. Saadut hyödyt eivät olekaan vastanneet odotuksia. Nykyisessä ulkoistamistrendissä on kuitenkin yksi merkittävä ero entiseen: uuden ajan tietotekniikka. Viime vuosikymmeninä ja jopa vuosina on tietotekniikan ja tietojenkäsittelytieteen kehitys parantanut sekä hajautetun tietojenkäsittelyn että ulkoistuksen tuloksena syntyvien liiketoimintaverkostojen toimintaedellytyksiä. Vaikuttaakin siltä, että **sähköisen liiketoiminnan** kehitys voisi lähitulevaisuudessa tarjota uuden ja automatisoidumman tavan liiketoimintaverkostojen valvontaan ja hallinnointiin.

Samassa liiketoimintaverkostossa toimivien yritysten väliseen yhteistoimintaan liittyy **materiaalivirtoja**, **maksuvirtoja** sekä näitä ohjaavia **informaatiovirtoja**. Materiaalivirtojen käsittely eli **logistiikka** liittyy konkreettisten hyödykkeiden varastointiin ja liik-

keeseen. Maksuvirrat kuvaavat rahan abstraktia virtaa ja kulkevat materiaalivirtojen suhteen vastakkaiseen suuntaan liiketoimintaverkostoa. Informaatiovirrat kuvaavat fyysistä tai sähköistä viestintää, jonka tehtävänä on **seurata**, **valvoa** ja **hallita** liiketoimintaverkostoa kokonaisuudessaan (mukaanlukien verkoston materiaali- ja maksuvirrat).

Sähköisessä liiketoiminnassa informaatiovirrat koostuvat **sähköisistä viesteistä**. Vuorovaikuttavat tahot sopivat ennakkoon minimaalisesta määrästä yhteisiä ominaisuuksia. Molempien on puhuttava samaa kieltä (**syntaktinen** yhteisymmärrys) ja samoilla käsitteillä (**semanttinen** yhteisymmärrys). Syntaktisen yhteisymmärryksen tapauksessa toimijoilla on oltava yhteinen näkemys esimerkiksi käytettävästä **kommunikointiteknologiasta** ja semanttisen yhteisymmärryksen tapauksessa yhteisestä **ontologiasta** [4, 49]. Lisäksi toimijoilla on oltava näkemys siitä, mitä hyötyä kullekin osapuolelle liiketoimintaverkostoon osallistumisesta on (**pragmaattinen** yhteisymmärrys).

Liiketoimintaverkostossa käytävä sähköinen viestinvaihto voi noudattaa **implisiittistä** tai **eksplisiittistä koreografiaa**. Implisiittisen koreografian tapauksessa sovitaan viestitykseen liittyvistä konventioista vain yleisellä tasolla. Eksplisiittisen koreografian tapauksessa mennään askel pidemmälle ja sovitaan yhteisesti käytettävän kielen ja käsitteiden lisäksi myös verkostossa käytävän keskustelun tarkemmasta kulusta. Puhtaimmat implisiittistä koreografiaa edustavat lähestymistavat löytyvät semanttisen webin [3, 51, 53] ja sähköisten agenttijärjestelmien puolelta [52]. Helsingin yliopiston Web-Pilarcos projektissa on sen sijaan painopiste ollut ennen kaikkea ekplisiittisiin koreografioihin perustuvissa verkostoissa.

Eksplisiittinen koreografia sisältää paitsi kommunikointiin liittyvien viestien syntaksin ja semantiikan myös yleisesti hyväksytyn käyttäytymisen kuvauksen. Yleisemmin ilmaistuna ekplisiittinen koreografia kuvaa, mitä eri osapuolilta vaaditaan eli mitä on **pakko** tehdä, mikä on osapuolille sallittua eli mitä on **lupa** tehdä, ja mikä ei ole osapuolille sallittua eli mitä on **kiellettyä** tehdä. Kuva 1.1 havainnollistaa lupiin, kieltoihin ja vaatimuksiin pohjautuvan kuvaustavan ilmaisuvoimaa ja käyttöä. Vasemmalla kuvassa on tilanne, jossa sekä ehto X että sen negaatio !X ovat luvallisia. Seuraavana oikealle mentäessä on tilanne, jossa !X on muuttunut kielletyksi. Tästä seuraa, että !X:n negaatio eli X on muuttunut vaadituksi. X on edelleen myös luvallista. Oikeassa laidassa oleva kuva esittää edellisen kanssa päinvastaista tilannetta. Lupiin, kieltoihin ja vaatimuksiin perustuvia loogisia säännöstöjä kutsutaan **deonttiseksi logiikaksi**.

Kuva 1.1: Lupa, kielto, vaatimus -järjestelmän ilmaisuvoiman ja käytön havainnollistus.

Eksplisiittinen koreografia voidaan mallintaa suoraan deonttisen järjestelmän avulla

kuvatuiksi säännöiksi eli **politiikoiksi** [1, 15, 16, 38]. Sähköisen liiketoiminnan tapauksessa puhutaan tällöin usein **liiketoiminnallisista politiikoista** (business policy). Eksplisiittinen koreografia voidaan myös mallintaa **prosessilähtöisesti** eli kuvaamalla kunkin keskusteluun osallistujan noudattama **sähköinen prosessi**. Sähköisen liiketoiminnan kontekstissa käytetään usein tarkempaa termiä **sähköinen liiketoimintaprosessi** tai pelkästään ulkoisesti näkyvään kommunikointiin liittyen **ulkoinen (sähköinen) liiketoimintaprosessi**. Joskus ulkoisesta liiketoimintaprosessista puhuttaessa käytetään myös nimitystä **liiketoimintaprotokolla**.

Sähköisten viestien vaihtoon liittyvä ulkoinen liiketoimintaprosessi voidaan mallintaa useilla erilaisilla kielillä. Kysymykseen voi tulla esimerkiksi “perinteisempi” ohjelmistojen mallinnuskieli kuten Unified Modeling Language (UML) [33] tai uudempi sähköisten liiketoimintaprosessien / hajautetun työnkulun kuvaamiseen tarkoitettu kieli kuten Business Process Execution Language [22, 5] tai XML Process Definition Language (XPDL) [39]. Nykyisin on kehitteillä myös puhtaasti ulkoisen liiketoimintaprosessin kuvaamiseen tarkoitettuja kieliä kuten Web Services Choreography Description Language (WS-CDL) [35].

Uusin lähestymistapa on politiikka- ja prosessilähtöisen kuvauksen yhdistävä hybridiratkaisu. Hybridiratkaisussa ideana on kuvata noudatettava peruskäyttäytyminen prosessilähtöisesti ja erikoistaa prosessia sen jälkeen politiikkojen avulla erikoistilanteisiin sopivaksi. Sähköisten prosessien ja koreografioiden kuvaamiseen sopivia mallinnuskieliä on aktiivisesti kehitetty ja edelleen kehitetään muun muassa OMG-, WFMC-, W3C- ja OASIS-organisaatioiden [48, 47, 55, 56] alaisuudessa.

Eksplisiittisten koreografioiden eräs etu implisiittisiin nähden on se, että suunnitellun liiketoimintaverkoston käyttäytymistä voidaan suunnitteluaikaisesti analysoida sekä suoritusaikaisesti valvoa. Jotta nämä ekplisiittisen kuvaustavan teoreettisest hyödyt saataisiin kokonaisuudessaan realisoitua, tulisi käytettävän mallinnuskielen pohjautua johonkin formaalin semantiikan omaavaan kieleen. Tällaisia kieliä ovat esimerkiksi petriverkot [21] tai prosessialgebrat. Tunnetuin uudemmista prosessialgebroista lienee pii-kalkyyli [17, 45]. Edellä esitetyistä mallinnuskielistä pii-kalkyyliin perustuu esimerkiksi WS-CDL. Formaalin semantiikan omaavalla kuvauksella tehdyt kuvaukset on mahdollista **automaattisesti verifioida** jo suunnitteluaikana. Koreografia, joka käyttäytyy kuten on aiottu, on **oikeellinen** suhteessa sen spesifikaatioon. Oikeellisesti käyttäytyvän koreografian ominaisuuksiin kuuluu esimerkiksi **lukkiutumattomuus**.

Helsingin yliopiston tietojenkäsittelytieteen laitoksen Web-Pilarcos projektissa [54], tutkitaan ja kehitetään **kieliä**, **väliohjelmistopalveluita** ja **menetelmiä** sähköisesten sopimusten varassa toimivien liiketoimintaverkostojen tarpeisiin. Erityisinä kiinnostuksen kohteina on liiketoimintaverkostojen mallintaminen ja mallien verifiointi sekä liiketoimintaverkostojen muodostamiseen, valvomiseen ja hallinnointiin liittyvät **väliohjelmistopalvelut**. Web-Pilarcos projektissa tuotetut julkaisut ja tekniset raportit (esimerkiksi [7, 8, 9, 10, 11, 12, 13]) käsittelevät näitä teemoja useista eri näkökulmista. Tässä tutkielmassa tarkastellaan yritysten liiketoimintaverkostoon liittyvää sähköistä viestinvaihtoa automaattisen valvonnan näkökulmasta.

Sähköisiin sopimuksiin perustuvassa liiketoimintaverkostossa toimivien yritysten integroimisen yksi ydinongelma on se, miten automatisoidaan yritysten välisiin informaa-

tiovirtoihin liitettyjen toiminnallisten ja ei-toiminnallisten vaatimusten toteutumisen valvonta [2, 19]. Kokonaisuudessaan ihmisen harteille jätettynä valvontatehtävä on liian raskas ja virhealtis. Automatisoitu valvonta mahdollistaa ongelmatilanteiden suoritusaikaisen havaitsemisen ja joissain tapauksissa jopa niiden estämisen. Tämän lisäksi automatisoitu valvonta mahdollistaa liiketoimintaverkostoihin liittyvän tilan lähes reaaliaikaisen seurannan sekä valvontatiedon tallentamisen myöhempää käyttöä varten. Talletettua valvontatietoa voidaan käyttää varsinaisen valvontatehtävän lisäksi esimerkiksi vuorovaikutukseen liittyvien liiketoimintaprosessien tehostamiseen. Automaattisen valvonnan käyttämä tieto eli **valvontainformaatio** (käytetään myös nimitystä **valvonnan metainformaatio**) voidaan pitkälti tuottaa yritysten välisiin sähköisiin sopimuksiin kirjattujen **yhteisöarkkitehtuurien** pohjalta.

Automaattisella valvonnalla on useita toisiinsa liittyviä tehtäviä. Tutkielmassa on tavoitteena tarkastella automatisoitua valvontaa yleisellä tasolla, laajemmassa mittakaavassa ("in-the-large"), sekä syventää ymmärrystä erityisesti valituilla valvonnan osa-alueilla. Lopullisena tavoitteena on esittää algoritminen ratkaisu ja teknologiaspesifinen toteutusarkkitehtuuri valituille valvonnan osa-alueille. Tarkastelussa keskitytään valvontanäkökulmaan eikä esimerkiksi suuremmin oteta kantaa siihen milloin, missä ja kenen toimesta sähköiset sopimukset syntyvät tai miten sopimuksia tai liiketoimintaverkostoa kokonaisuudessaan hallinnoidaan. Myöskään mallinnuskysymyksiin ei puututa. Näitä kysymyksiä tutkitaan erillisinä Web-Pilarcos projektissa.

Valvontakoneiston käyttämän metainformaatiomallin kehittämisessä tukeudutaan olemassa olevaan, sähköisten sopimusten ja koreografioiden automaattista valvontaa koskevaan, aiempaan tutkimukseen. Merkittävä painoarvo on myös Web-Pilarcos projektissa aiemmin tuotetulla materiaalilla. Valvontakoneiston prototyyppi toteutetaan Web-palveluarkkitehtuuria [34] ja siihen liittyviä protokollia [32, 36] tukevan Java-pohjaisen Apache Axis-alustan [41] päälle ja liitetään osaksi Web-Pilarcos projektissa tuotettavaa laajempaa prototyyppiä.

Tutkielma jakaantuu rakenteellisesti siten, että luvussa 2 esitellään sähköisten liiketoimintaverkostojen olennaiset käsitteet ja niiden hallinnointiin suunnatun väliohjelmiston yleispiirteet. Esittelyssä keskitytään valvonnan kannalta keskeisiin asioihin. Tämän jälkeen pureudutaan syvemmin väliohjelmiston valvontafunktioon. Valvontafunktiota lähestytään ensin yleisemmällä tasolla ja myöhemmin keskitytään sen tutkielman kannalta olennaisimpaan osajoukkoon.

Kolmannessa luvussa määritellään algoritminen ratkaisumalli valvontafunktiosta rajatulle osajoukolle. Pääpaino on abstraktiotasoltaan alimpien valvontakerrosten toteuttamisessa. Tarpeellisin osin käsitellään myös abstraktiotasoltaan korkeampia valvontakerroksia.

Neljännessä luvussa tehdään katsaus valittuihin toteutusteknologioihin ja suoritusaikaisiin sovellusalustoihin. Ensin esitellään tutkielman kannalta olennaiset osat Web-palveluarkkitehtuurista sekä sen suhteesta nelitasoisen yritysarkkitehtuurin ja erityisesti Java 2 Enterprise Edition-spesifikaation (J2EE) [29] mukaisesti toteutettuihin palveluihin. Lopuksi luodaan katsaus Apache Axis-alustaan Web-palveluja käyttävien ja toteuttavien sovellusten näkökulmasta. Neljännen luvun tavoitteena on luoda lukijalle viidennen luvun tarpeisiin riittävä ymmärrys käytettävästä teknologiamaailmasta.

Viidennessä luvussa määritellään esitellylle algoritmiselle ratkaisumallille toteutusarkkitehtuuri ja rajapinnat Web-palvelumaailmassa (erityisesti Apache Axis-alustalla).

Tutkielman kuudennessa ja viimeisessä luvussa tehdään yhteenveto tutkielman olennaisesta sisällöstä ja tuloksista. Lisäksi pohditaan tutkielmasta seuraavia johtopäätöksiä sekä mahdollisia tutkimuksellisia jatkosuuntia.

Luku 2

Liiketoimintaverkostoja hallinnoiva väliohjelmisto ja automatisoitu valvontafunktio

Liiketoimintaverkostojen sähköinen hallinta koko laajuudessaan sisältää hyvin laajan kirjon tutkimusaiheita. Usein liiketoimintaverkostojen hallintaan liittyvät teknologiat lähtevät liikkeelle organisaatioiden välisen sähköisen viestinnän mahdollistamisesta. Esimerkiksi pidempään teollisessa käytössä olleet integraatiomallit, kuten Electronic Data Interchange (Ansi X12 Standards) eli EDI [46], primitiivisemmät sähköisen kommunikaation muodot kuten sähköposti ja telefax sekä uudemman sukupolven lähestymistavat kuten ebXML [25] ja RosettaNet [50] lähtevät perusoletuksissaan liikkelle sähköisistä viesteistä ja niiden mallintamisesta sekä hallinnasta. Viestipohjaiseen lähestymistapaan pohjimmiltaan nojaa myös XML-teknologiaan [28, 57] vahvasti tukeutuva Web Services -arkkitehtuuri [34] ja sen liiketoimintaprosessien integrointiin tähtäävät uudet laajennokset.

Myös Web-Pilarcos projektissa [54] kaiken takana on organisaatioiden toisillensa lähettämät viestit. Web-Pilarcos tuo viestipohjaisuuden lisäksi vahvasti mukaan formaaliin mallinnukseen ja verifiointiin liittyviä ominaisuuksia sekä sähköiseen sopimukseen perustuvan integraatiomallin. Uudemman sukupolven teknologioista esimerkiksi ebXML [25] perustuu viestipohjaisuuden ohella sähköisiin sopimuksiin. Web-Pilarcos projektissa pyritään kehittelemään joustavia, teknisessä mielessä osapuolet toisiinsa mahdollisimman väljästi sitovia, mekanismeja korvaamaan aiempia raskaita ja kokonaisvaltaisia arkkitehtuureja.

Web-Pilarcoksen väliohjelmistoratkaisuissa pyritään mahdollistamaan liiketoimintaverkoston osapuolten suoritusaikainen löytäminen ja liiketoimintaverkoston toimintaa ohjaavan sopimuksen suoritusaikainen muodostaminen, käyttäminen ja valvominen. Sopimuspohjaisen ratkaisun on tarkoitus olla mahdollisimman hienojakoinen, palvelukohtainen. Globaaleilla markkinoilla tarjolla olevista palveluista on tarkoitus pystyä muodostamaan (jopa) suoritusaikaisesti uusia, oikeellisesti käyttäytyviä, palvelukokonaisuuksia. Tarkoituksena on, että syntyneen sähköisen sopimuksen pohjalta organisaation sisäiset järjestelmät osaavat **reflektiivisesti** konfiguroida itsensä toimimaan sopimusta noudatta-

vassa sähköisessä vuorovaikutussuhteessa eli **federaatiossa**. Tavoitteena on, että sopimus ei rajoita tai pakota yritystä mihinkään tarpeettomaan, vaan sisältää minimaaliset vaatimukset yhteistoimintaan osallistumiselle. Yritys voi lisäksi osallistua yhtäaikaa useampaan federaatioon, joita jokaista kattaa erilainen ja erillinen sähköinen sopimus. Projektissa pyritään yhdistelemään ideoita esimerkiksi palvelulähtöisiin arkkitehtuureihin, viestipohjaisuuteen, reflektiivisyyteen, väliohjelmistoihin, työnkulkukuvauksiin, sekä formaaliin spesifiointiin ja verifiointiin liittyvää tutkimusta.

Tässä pääkappaleessa esitellään liiketoimintaverkostojen hallinnointiin suunnatun väliohjelmiston yleispiirteet, sekä pureudutaan syvemmin väliohjelmistoon liittyvään valvontafunktioon. Valvontafunktiota lähestytään ensin yleisemmällä tasolla ja myöhemmin rajataan käsittely sen tutkielman kannalta olennaisimpaan osajoukkoon.

2.1 Liiketoimintaverkostot ja koreografi at

Samassa sähköisessä liiketoimintaverkostossa toimivien yritysten väliseen yhteistoimintaan liittyy sähköisiä informaatiovirtoja. Nämä informaatiovirrat voidaan mallintaa eksplisiittisinä viestityskoreografioina. Koreografia kuvaa osapuolille sallitut, osapuolilta vaaditut ja sitä kautta myös osapuolilta kielletyt viestisekvenssit, sekä viestisekvensseihin liittyvän temporaalisen synkronoinnin.

Koreografia voidaan kuvata joko koko liiketoimintaverkoston näkökulmasta, jolloin usein puhutaan **globaalista eli kaikki osapuolet kattavasta** koreografiasta tai vaihtoehtoisesti erikseen jokaisen liiketoimintaverkoston osapuolen näkökulmasta jolloin usein puhutaan **lokaalista eli paikallisesta** koreografiasta.

Täydellinen koreografia sisältää sekä vuorovaikutukseen liittyvän hallintavirran (control flow), että sisältövirran (data flow) kuvauksen. Hallintavirta määrittelee vähintään viestien vaihtoon liittyvät osapuolet, sallitut viestityypit sekä viestinvaihdon sallitut tilat. Hallintavirta voi lisäksi sisältää erilaisia palvelunlaatuun liittyviä ominaisuuksia esimerkiksi koreografian suoritukselle asetettavia aikarajoja.

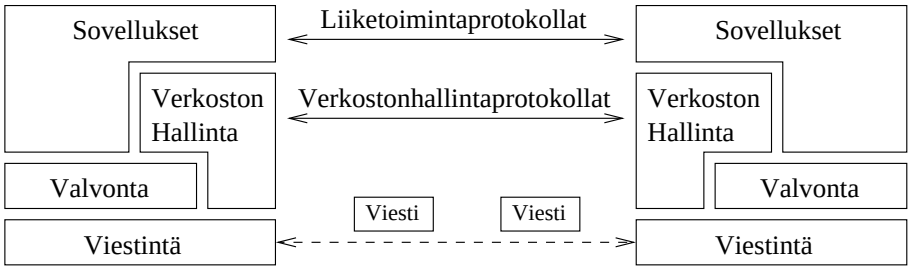
Viestinvaihtoon liittyviä tiloja ei yleensä määritellä suoraan vaan mallintaja kuvaa hallintavirran yksinkertaisella ohjelmointikielellä tai vastaavalla (joskus graafisella) notaatiolla prosessina tai työnkulkuna. Tällainen kuvaus voidaan jälkeenpäin muuntaa ajonaikaisen koneiston tarpeisiin sopivaan formaattiin. Tämän tutkielman puitteissa keskitytään erityisesti sellaisiin ekplisiittisesti määriteltyihin koreografioihin, jotka voidaan muuntaa suoritusaikaisen koneiston ymmärtämäksi **äärelliseksi tila-automaatiksi**.

Koreografiaan liittyvä sisältövirta kuvaa viestien sisällöt sekä eri viestien välisten sisältöjen sallitut suhteet eli **sisältörelaatiot**. Siinä, missä hallintavirta määrittelee osapuolten välisen viestinvaihdon tyypit ja järjestyksen, kuvaa sisältövirta sen mitkä ovat viestien sisältöjen sallitut arvot kullakin koreografian suoritushetkellä. Liiketoimintaverkoston tilan seurantaan ja valvontaan liittyvän väliohjelmiston kannalta hallintavirta on olennaisemmassa asemassa kuin sisältövirta. Viestien sisällöt on viime kädessä tarkoitettu sähköiset palvelut toteuttavien sovellusten tulkittavaksi. Mahdotonta sisältövirran oikeellisuuden automaattinen valvominen väliohjelmistotasolla ei ole, mutta erilaisten sisältöjen sovellusläheisyyden ja monimuotoisuuden vuoksi valvominen on usein syytä jättää

tarkoitukseen laadittujen sovellusten tai niitä käyttävien ihmisen vastuulle. Liiketoimintaverkoston itsensä tilan seurantaan ja valvontaan liittyvät tapahtumat voidaan riittävällä tarkkuudella poimia jo hallintavirrasta. Tämän tutkielman puitteissa sisältövirtaan ei enempää puututakaan.

2.2 Liiketoimintaverkostoa hallinnoiva väliohjelmisto

Liiketoimintaverkostoa hallinnoiva väliohjelmisto ympäristöineen voidaan valvonnan näkökulmasta jakaa neljään arkkitehturaaliseen osaan: väliohjelmistoa hyödyntäviin sovelluksiin, valvontakerrokseen, viestintäkerrokseen, sekä yleisiin verkostonhallintapalveluihin. Kuvassa 2.1 on esitetty edellä mainitut neljä arkkitehturaalista osaa sekä niiden sijoittuminen suhteessa toisiinsa.



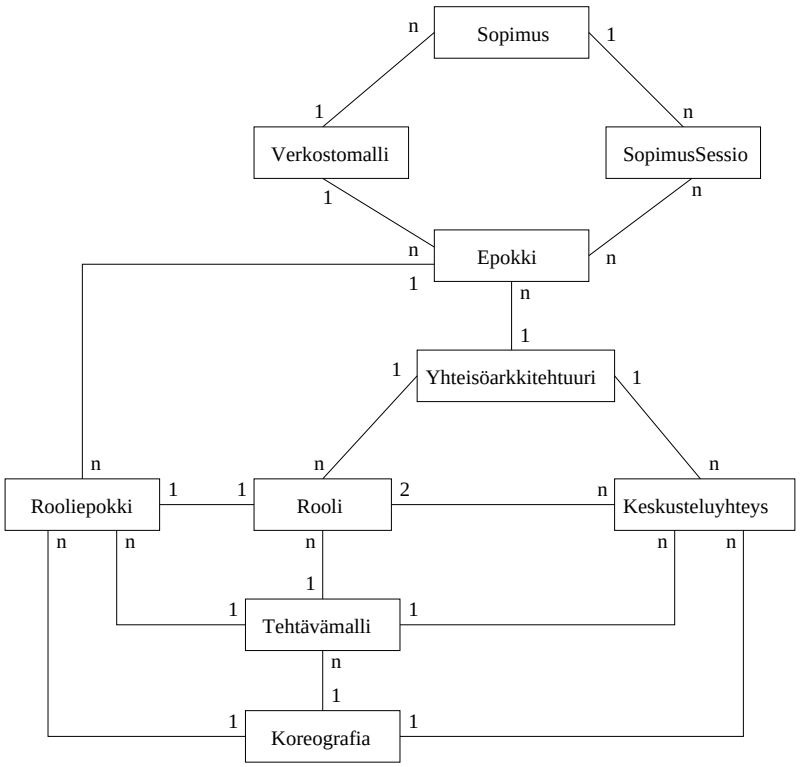
Kuva 2.1: Liiketoimintaverkostoa hallinnoivaan väliohjelmistoon liittyvät arkkitehturaaliset osat.

Sovellukset sisältävät varsinaisen liiketoimintalogiikan ja kommunikoivat keskenään tukemiensa liiketoimintaprotokollien mukaisesti. Valvontakerros seuraa sovellusten välistä kommunikointia ja valvoo, että se noudattaa sähköisessä sopimuksessa kuvattuja (etukäteen oikeelliseksi verifioituja) koreografioita. Viestintäkerros mahdollistaa viestinnän eli sähköisten viestien lähettämisen ja vastaanottamisen hajautetussa ympäristössä. Viestintäkerrokseen on tyypillisesti tarjolla useita vaihtoehtoisia teknologioita. Verkostonhallintapalvelut ovat vastuussa liiketoimintaverkoston toimijoiden löytämisestä sekä näiden välisten sähköisten sopimusten muodostamisesta. Verkostonhallintapalvelut vuorovaikuttavat keskenään hyvin määriteltyjä **verkostonhallintaprotokollia** käyttäen ja hyödyntävät viestintäkerrosta keskinäisessä kommunikaatiossaan samaan tapaan kuin varsinaiset sovelluksetkin. Verkostonhallintapalveluihin voidaan laskea mukaan myös erilaiset mallinnus- ja verifiointityökalut, jotka eivät kuulu verkostoa suoritusaikana hallinnoivien palveluiden joukkoon.

2.3 Liiketoimintaverkoston valvontaan liittyvä käsitteistö

Kuvassa 2.2 on esitelty liiketoimintaverkostojen valvontaan eksplisiittisesti tai implisiittisesti liittyvät mallinnuskäsitteet. **Sopimus** kuvaa koko liiketoimintaverkoston toiminnan

kattavan pitkäaikaisen vuorovaikutussuhteen. Sopimuksessa voi olla sekä liiketoiminnalliselta kannalta olennaiset sovitut asiat että osapuolten teknisen yhteistoiminnan kannalta olennaiset sovitut asiat. Web-Pilarcos projektissa keskitytään erityisesti tekniseen yhteistoimintaan liittyviin osiin.



Kuva 2.2: Liiketoimintaverkoston automatisoituun valvontaan eksplisiittisesti tai implisiittisesti liittyvät käsitteet.

Sopimukseen liittyy aina yksi **verkostomalli** sekä yksi tai useampia suoritusaaikaisia instansseja eli **sopimussessioita**. Sopimussessio eroaa sopimuksesta siinä, että se ylläpitää verkoston vuorovaikutuksiin liittyvää suoritusaikaista tilaa. Sopimussessioita voidaan käynnistää useita ja ne saavat oman tunnisteen sekä perivät sopimukselta sen alkutilan. Tämän jälkeen eri sopimussessioden tila voi kehittyä toisistaan riippumatta. Esimerkkinä sopimuksesta voisi olla paperikoneen tilaajan, paperikoneen valmistajan ja logistiikkaoperaattorin solmima puitesopimus paperikoneiden tilauksesta ja perilletoimituksesta. Esimerkiksi sopimussessiosta puolestaan käy yhden (sopimuksen puitteissa tehdyn) paperikoneen tilaus ja siihen liittyvän toimituksen eteneminen.

Jokaiseen tietyn sopimuksen puitteissa käynnistettyyn sopimusessioon liittyy sama verkostomalli sekä yksi tai useampi elinkaaren vaihe eli **epookki**. Yhteen epookkiin liittyy aina yksi **yhteisöarkkitehtuuri**, joka kuvaa epookkiin liittyvät liiketoiminnalliset roolit sekä roolien väliset **keskusteluyhteydet** (conversation). Paperikone-esimerkissä rooleja voisivat olla vaikkapa jo aiemmin mainitut paperikoneen tilaaja, paperikoneen valmistaja ja kuljetuksia hoitava logistiikkaoperaattori (todellisuudessa rooleja olisi luultavasti enemmän ja ne olisi jaettu useampaan yhteisöarkkitehtuuriin). Paperikoneen tilaajalla ja

valmistajalla olisi vähintäänkin paperikoneen tilaamiseen tarvittava keskusteluyhteys ja paperikoneen valmistajalla ja logistiikkaoperaattorilla olisi luultavasti vähintään toimitustilaukseen liittyvä keskusteluyhteys. Lisäksi logistiikkaoperaattorilla ja paperikoneen tilaajalla saattaisi olla logistisiin järjestelyihin liittyvä keskusteluyhteys.

Kunkin kahden roolin välillä on mahdollista olla yksi tai useampi keskusteluyhteys. Joidenkin roolien välillä sitä ei ole kenties lainkaan. Teoriassa on myös mahdollista, että tietty keskusteluyhteys yhdistää useamman kuin kaksi roolia. Tällöin syntyy esimerkiksi kolmen roolin välinen keskusteluyhteys. Tämän tutkielman puitteissa esitettävässä mallissa monenvälisyys on kuitenkin piilotettu rooleihin itseensä. Monenvälistä kommunikointia mallitettaessa on monenvälisen keskusteluyhteyden sijaan määriteltävä uusi rooli, johon monenvälisesti kommunikoivat roolit liitetään omalla keskusteluyhteydellään. Uusi rooli toteuttaa siihen liitettyjen roolien välisen kommunikoinnin synkronoinnin ja toimii tähtimäisenä monenvälisyyspisteenä. Monenvälisten keskusteluyhteyksien suora mallintaminen jätetään tulevaisuuden tutkimuskohteeksi.

Koska jokaiseen epookkiin liittyy oma yhteisöarkkitehtuurinsa voi myös roolien ja keskusteluyhteyksien määrä eri epookeissa vaihdella. Koko liiketoimintaverkoston epookki vaihtuu kun kaikki tiettyyn epookkiin liittyvät roolit ovat saaneet oman **rooliepookkinsa** loppuun. Rooliepookki loppuu kun sen kaikki pakolliset **tehtävät** on käyty läpi. Tiettyyn rooliepookkiin liittyvät tehtävät ja niiden väliset suhteet kuvaa **tehtävämalli**. Tehtäviä voidaan nimittää myös rooliepookin **elinkaaren vaiheiksi**. Kaikilla rooliepookeilla on oltava vähintään kaksi elinkaaren vaihetta: alku ja loppu. Niillä voi myös lisäksi olla yksi tai useampi vaihtoehtoinen välivaihe. Tehtävämalli voidaan esittää **tila-automaattina**, jolla on yksi alkutila, yksi lopputila sekä optionaalisesti useampia vaihtoehtoisia välitiloja.

Jokaiseen rooliepookkiin ja sitä kautta myös tehtävämalliin liittyy yksi koreografia. Koreografia kuvaa rooliepookkiin liittyvän viestinvaihdon mahdolliset tilat. Koreografia on kuvattu globaalista näkökulmasta. Kuten tehtävämalli myös koreografia voidaan esittää tila-automaattina jolla on yksi alkutila ja yksi lopputila sekä optionaalisesti useampia vaihtoehtoisia välitiloja.

Tehtävämalleihin, koreografioihin ja niiden tila-automaattiesityksiin tutustutaan tarkemmin luvussa kolme.

2.4 Automatisoidun valvontafunktion rooli väliohjelmistossa

Automatisoidulla valvontafunktiolla on liiketoimintaverkostoa hallinnoivassa väliohjelmistossa useita tehtäviä. Tehtävät voidaan jaotella kahden pääkategorian alle. Ensimmäinen päätehtävä on pitää yllä liiketoimintaverkoston tilaa kaikilla suoritusaikaisen koneiston tukemilla abstraktiotasoilla. Toinen päätehtävä on havaita verkoston toimintaan liittyviä virhetilanteita ja mahdollisuuksien mukaan jopa estää niitä tapahtumasta (tai ainakin minimoida tapahtuneen virheen vaikutusten leviäminen koko verkoston laajuuteen).

Kuvassa 2.3 on esitetty liiketoimintaverkostojen valvontaan liittyvät abstraktiotasot.

Kuva 2.3: Liiketoimintaverkostojen valvontaan liittyvät vertikaalitasot.

Sopimustason valvonta on vastuussa koko liiketoimintaverkoston tasolla tapahtuvasta valvonnasta. Sopimustason vastuulla on muun muassa valvonnan tilan levittäminen eri organisaatioiden välillä sekä epookkien vaihdon synkronointi liiketoimintaverkoston laajuisesti. Sopimustason valvonnan on myös pidettävä kirjaa sopimukseen liittyvistä sopimussessioista. Sopimustasolle välitetään kaikki alemmilla tasoilla havaitut virhetilanteet ja sopimustaso on vastuussa näiden virheiden tulkinnasta liiketoimintaverkoston tasolla. Sopimustason valvontaa ei tässä tutkielmassa käsitellä muuten kuin satunnaisin viittauksin. Sitä tutkitaan erillään Web-Pilarcos projektissa.

Sessiotason valvonta on vastuussa tietyn sopimussession valvomisesta yhdessä organisaatiossa. Sessiotason on oltava tietoinen siitä, missä epookissa sessio kulloinkin on ja ovatko kaikki epookkiin liittyvät oman organisaation rooliepookit päättyneet. Sessiotaso on tietoinen myös organisaatioon kuulumattomista rooleista ja saa näihin liittyviä tilatietoja sopimustason kautta. Sessiotaso edelleen välittää sopimustasolta tulleet tilatiedot oikeille roolitason instansseille samoin kuin roolitasolta tulleet tiedot sopimustasolle. Sessiotaso myös avustaa sopimustasoa epookin vaihdoissa ilmoittamalla sopimustasolle milloin kaikki organisaation roolit ovat saaneet omat rooliepookkinsa valmiiksi. Sessiotasolle voidaan myös konfguroida kuhunkin sessioon liittyvät politiikat, joita sessiotaso valvoo.

Roolitason valvonta on vastuussa yksittäiseen rooliin liittyvästä yleisestä valvonnasta. Roolitason on oltava tietoinen valvomansa roolin rooliepookin tilasta ja tiedotettava siinä tapahtuvista tilamuutoksista sessiotasoa. Roolitaso saa tiedon rooliepookin elinkaaren tilamuutoksista tehtävämallitasolta. Roolitaso pitää yllä myös organisaation ulkopuolisten roolien tilaa ja saa tähän liittyviä tilatietoja sessiotasolta. Roolitasolle voidaan konfiguroida roolispesifisiä politiikkoja, joiden toteutumista roolitaso valvoo. Politiikkavalvontaa tehdään kuitenkin vain oman organisaation hallinnoimille rooleille. Roolitaso välittää tehtävätasolta saapuvat virheilmoitukset ylemmille valvontakerroksille.

Tehtävätason valvonta on vastuussa tiettyyn rooliin liittyvän rooliepookin elinkaaren seurannasta. Jokaista rooliepookkia varten tehtävätasolle konfiguroidaan oma rooliepookkispesifinen metatieto eli tehtävämalli. Tehtävämallin avulla tehtävätason valvonta tietää, missä elinkaarensa vaiheessa rooliepookki kullakin ajan hetkellä on. Tehtävämallitaso saa tehtävävaikutuksia ja virheraportteja koreografiatasolta. Tehtävävaikutusten perusteella päätellään, milloin epookin elinkaaressa siirrytään uuteen vaiheeseen. Koreografiatasolta saapuneet virheraportit tallennetaan tehtävämallin yhteyteen myöhempää tarkastelua varten ja edelleen ohjataan roolitason valvontakerrokselle.

Koreografiatason valvonta on vastuussa tiettyyn rooliepookkiin liittyvien keskusteluyhteyksien tilan seurannasta ja valvonnasta. Koreografiataso toimii rooliepookkikohtaisen metatiedon varassa, jonka perusteella se päättelee onko kulloinkin lähetettävä tai vastaanotettava viesti sovitun koreografian mukainen. Koreografiataso voi myös tarvittessa proaktiivisesti estää virheellisten viestien etenemisen oman organisaation ulkopuolelle tai organisaation sisään.

Sovellustason valvonta on vastuussa kunkin yksittäisen sovelluksen sisäisen toiminnan oikeellisuuden valvonnasta. Sovellustason valvonnan logiikka voi olla upotettu suoraan osaksi sovelluksen liiketoiminnallista logiikkaa tai sovellus voi vaihtoehtoisesti periä valvontalogiikan käyttämältään sovelluskehykseltä. Sovellustason valvontaan ei tässä työssä jatkossa puututa.

Tasoriippumaton valvonta on nimitys niille valvontafunktion osille, jotka eivät selkeästi liity mihinkään aiemmin esiteltyyn liiketoimintaverkostojen valvonnan vertikaaliseen abstraktiotasoon. Tasoriippumaton valvonta kuvaa ennenkaikkea niitä valvontaominaisuuksia, jotka ovat organisaatiossa aina olemassa ja toiminnassa kaikille sovelluksille (niillekin, jotka eivät kuulu mihinkään liiketoimintaverkostoon). Tasoriippumaton valvonta näkyy liiketoimintaverkostossa lähinnä niissä tilanteissa, joissa sovelluksen suorittama toiminto ei riko liiketoimintaverkoston kattavaa sopimusta, mutta sitä ei silti voida suorittaa organisaation sisäisistä syistä. Tällöin esimerkiksi sopimuksen mukainen viesti saatetaan pysäyttää organisaation rajalla, koska se rikkoo sopimuksessa mainitsematonta organisaation sisäistä politiikkaa (viesti voi vaikkapa sisältää luottamuksellista tietoa tai viruksen tai olla vain lähetetty väärään kellonaikaan). Tasoriippumatonta valvontaa ei tässä tutkielmassa käsitellä enempää.

2.5 Automatisoidun valvontafunktion ylläpitämä liiketoimintaverkoston tila

Liiketoimintaverkoston tila voidaan jakaa samalla periaatteella kuin aiemmin esitetyt valvonnan vertikaaliset abstraktiotasotkin.

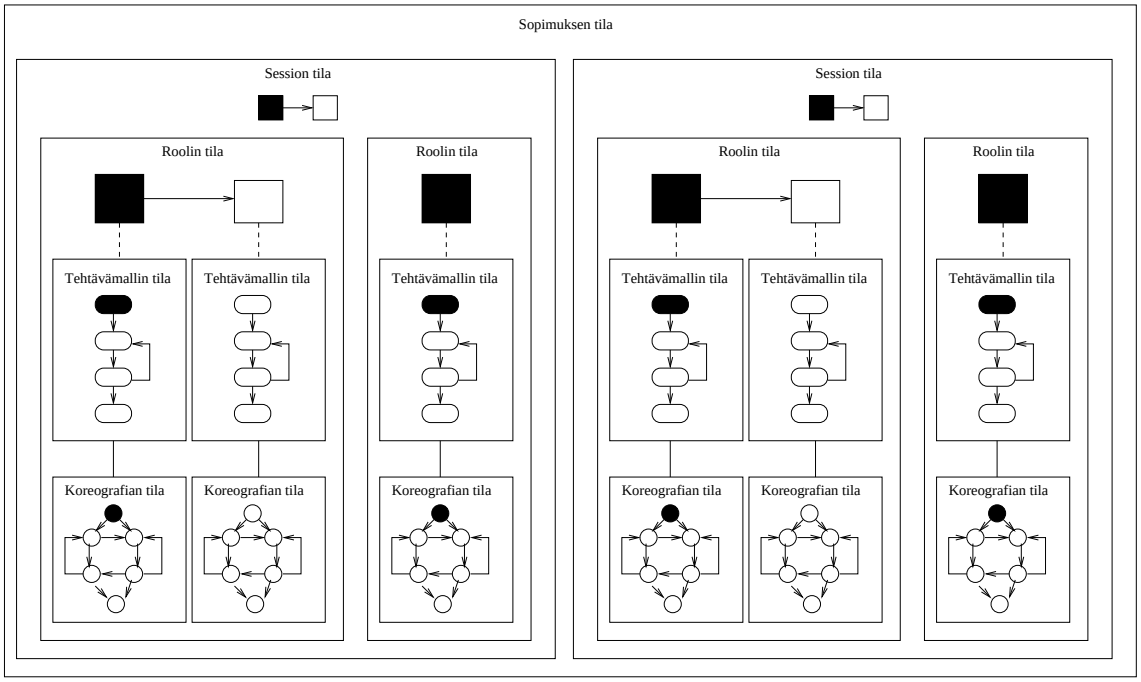
Sopimuksen tila koostuu sopimukseen liittyvistä sopimussessioista ja niiden tilasta. Lisäksi sopimuksen tilaan voidaan laskea kuuluvaksi sopimusneuvottelun aikana kiinnitetyt kaikkia sopimussessioita koskevat parametrit ja politiikat.

Sopimussession tila koostuu sessioon liittyvien roolien tilasta. Lisäksi sopimussession tila sisältää tiedon siitä missä epookissa sessio kulloinkin on.

Roolin tila koostuu rooliepookin tilasta (aktiivinen/deaktivoitu) sekä rooliepookkiin liittyvistä tehtämallin tilasta ja koreografian tilasta.

Tehtävämallin tila muodostuu tehtäväkerroksen valvontafunktion ylläpitämästä, rooliepookin elinkaaren vaiheita mallintavasta, tila-automaatista. Varsinaisen tila-automaatin lisäksi tehtävämallin tila sisältää tila-automaatin suoritusjäljen (execution trace) sekä koreografiakerroksen ilmoittamien virhetilanteiden jäljen (exception trace).

Koreografian tila muodostuu rooliepookkiin liittyvien ulkoisten keskusteluyhteyksien tilaa ylläpitävästä tila-automaatista.



Kuva 2.4: Valvontafunktion eri abstraktiotasoilla ylläpitämä tila.

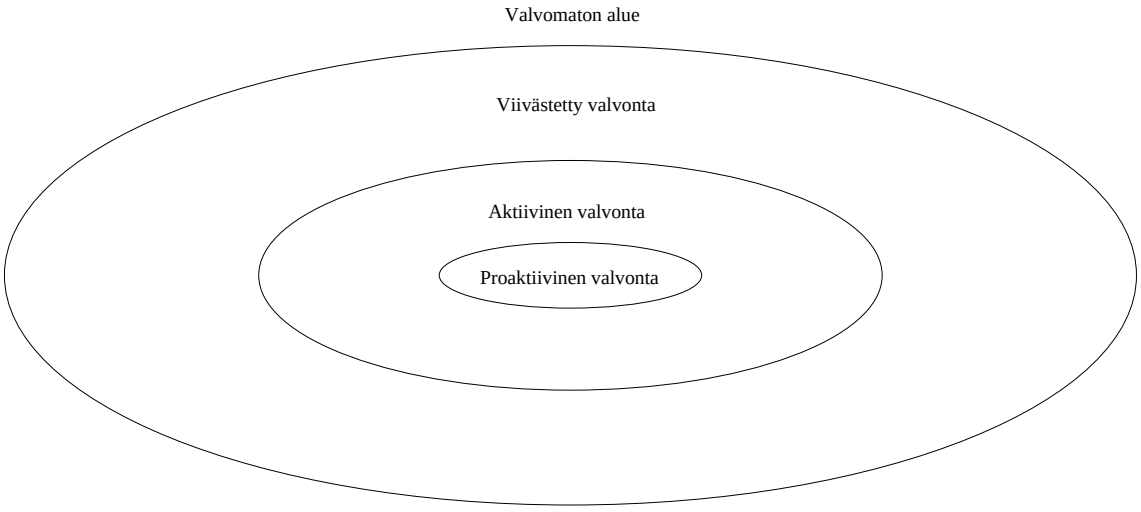
Kuva 2.4 sisältää visuaalisen yhteenvedon eri tasoilla ylläpidettävästä tilasta. Koreografian tilaa lukuunottamatta kaikki ylläpidettävä tila välitetään liiketoimintaverkoston jokaiselle osapuolelle, elleivät nämä ole sitä erikseen kieltäneet.

2.6 Valvonnan automatisoinnin kypsyystasot

Valvonnan automatisointiin voidaan liittää **kolme kypsyystasoa** sen mukaan kuinka pitkälle vietyä valvonnan automatisointi on. Valvonnan automatisoinnin kypsyystasot ovat **viivästetty**, **aktiivinen** ja **proaktiivinen** taso [18].

Primitiivisimmällä valvonnan automatisoinnin kypsyystasolla puhutaan viivästetystä tai passiivisesta valvonnasta. Viivästetyssä valvonnassa valvottavat tapahtumat tallennetaan pysyvään lokiin ja varsinaiset valvontarutiinit suoritetaan vasta myöhemmässä vaiheessa tallennettua lokia hyväksikäyttäen. Viivästetyllä valvonnalla ei voida taata järjestelmän suoritusaikaista eheyttä millään tavalla, eikä myöskään käytännössä valvoa aikakriittisiä tapahtumia. Viivästetyn valvonnan vahvuus on sen helppo ja suorituskykymielessä tehokas toteutettavuus. Joitakin mielenkiintoisia asioita ei voida (suorituskyvyn kannalta) tehokkaasti valvoa muuten kuin viivästetysti.

Viivästetyn valvonnan tuottamaa lokia voidaan varsinaisen valvontatehtävän ohella käyttää järjestelmän ja siihen liittyvien prosessien ja vuorovaikutussuhteiden parempaan ymmärtämiseen ja niiden tehostamiseen. Liiketoimintaverkostoon liittyvien viestipohjaisten informaatiovirtojen tapauksessa viivästetyn valvonnan tuottama loki koostuu olennaisesti viestien lähetykseen tai vastaanottoon liittyvistä yleisistä tunnisteista ja viestien tyypeistä sekä aikaleimoista. Lisäksi lokiin on syytä kirjoittaa onko lokimerkintä tehty viestin lähettävässä vai vastaanottavassa organisaatiossa.



Kuva 2.5: Valvonnan automatisoinnin kypsyystasot.

Toisella valvonnan automatisoinnin kypsyystasolla puhutaan aktiivisesta valvonnasta. Aktiivisen valvonnan tapauksessa valvontarutiineja suoritetaan samanaikaisesti valvottavan järjestelmän suorituksen kanssa. Valvontarutiinien suorittaminen tapahtuu rinnakkain valvottavan järjestelmän kanssa siten, että valvontajärjestelmä ei suoraan vaikuta valvottavan järjestelmän suoritukseen. Aktiivisella valvonnalla saavutetaan se hyöty viivästettyyn valvontaan nähden, että virhetilanteet ja aikakriittisten tapahtumien myöhästyneet suoritukset havaitaan lähes realiaikaisesti ja voidaan saattaa valvonnan tuloksista kiinnos-

tuneiden tahojen tietoisuuteen.

Aktiivinen kypsyystaso mahdollistaa esimerkiksi ennalta sovittujen viestityskoreografioiden rikkomisesta aiheutuvien virhetilanteiden havaitsemisen heti rikkomuksen tapahduttua. Jos osapuolten liiketoimintaprosessit ovat pitkälle optimoituja ja herkkiä esimerkiksi myöhästelyille tai virheille saavutetaan aktiivisella valvonnalla selkeä hyöty. Myös koreografioisssa tapahtuvien siirtymien pohjalta etenevien tehtävämallien tilaa voidaan ylläpitää suoritusaikana, jolloin rooliepookkien loppumiset ja sitä kautta epookinvaihtamistarpeet ja edelleen myös itse epookin vaihdot voidaan pitkälti automatisoida.

Aktiivinen valvonta ei kuitenkaan riitä, jos epookkien vaihtojen on tapahduttava yhtäaikaisesti kaikissa liiketoimintaverkoston toimijoissa. Riittämättömyys on seurausta siitä, että aktiivinen valvontakerros ei pysty keskeyttämään sovellusten toimintaa epookin vaihdon ajaksi. Erityisesti tilanteessa, jossa eri organisaatioiden rooliepookit päättyvät ajallisesti hyvin kaukana toisistaan tarvitaan jatkossa esiteltävää proaktiivista valvontaa. Proaktiivinen valvonta estää tiettyjä osapuolia etenemästä uuteen epookkiin ennenkuin kaikki muut osapuolet ovat saaneet edellisen epookin suoritettua.

```
<MonitoringEntry>
   <timeStampGM>Mon Oct 04 21:22:01 EEST 2004</timeStampGM>
   <timeStamp>1096914121652</timeStamp>
   <sessionID>PurchaseSession_1</sessionID>
   <conversationID>PurchaseConversation</conversationID>
   <senderID>Buyer</senderID>
   <receiverID>Seller</receiverID>
   <myRole>Buyer</myRole>
   <messageType>placeOrderRequest</messageType>
   <event>PURCHASE_START</event>
</MonitoringEntry>

<MonitoringEntry>
   <timeStampGM>Mon Oct 04 21:22:06 EEST 2004</timeStampGM>
   <timeStamp>1096914126064</timeStamp>
   <sessionID>PurchaseSession_1</sessionID>
   <conversationID>PurchaseConversation</conversationID>
   <senderID>Seller</senderID>
   <receiverID>Buyer</receiverID>
   <myRole>Buyer</myRole>
   <messageType>getProductInfoResponse</messageType>
   <exception>MonitoringException</exception>
</MonitoringEntry>
```

Kuva 2.6: Koreografiatasolla toimivan valvontafunktion tuottama loki (esimerkki).

Aktiivista valvontaa voidaan pitää minimaalisena vaadittavana kypsyystasona, ennenkuin todella voidaan puhua sähköisiin sopimuksiin perustuvien järjestelmien automatisoidusta valvonnasta. Aktiivinen valvonta on suorituskykymielessä raskaampaa kuin viivästetty valvonta. Algoritmisesti ne ovat yhtä raskaita, mutta koska aktiivista valvontaa

suoritetaan yhtäaikaisesti varsinaisten sovellusten kanssa (ei esimerkiksi erillisinä eräajoina) on järjestelmiin kohdistuva käytännön kuorma selvästi suurempi. Aktiivinen valvonta ei kuitenkaan välttämättä aiheuta merkittävää haittaa järjestelmän suoritustehokkuudelle ainakaan tyypillisissä, pitkäkestoisissa ja ei-reaaliaikaisissa, sähköisen liiketoiminnan transaktioissa.

Kolmannella valvonnan automatisoinnin kypsyystasolla puhutaan proaktiivisesta valvonnasta. Proaktiivisen valvonnan olennaisin ero aktiiviseen valvontaan nähden on se, että valvontarutiineja ja valvottavaa järjestelmää suoritetaan yhtäaikaisesti mutta keskenään synkronoidusti. Myös aktiivisen valvonnan tapauksessa valvontarutiineja voidaan suorittaa yhtäaikaisesti valvottavan järjestelmän suorituksen kanssa, mutta valvontarutiinit toimivat tällöin sovellusten kanssa aidosti rinnakkain eivätkä voi suoraan vaikuttaa sovellusten toimintaan.

Käytännössä proaktiivinen valvonta toimii siten, että aina kun valvottavassa järjestelmässä on tapahtumassa jokin mielenkiintoinen tapahtuma (esimerkiksi uuden viestin lähetys tai vastaanotto), valvottava järjestelmä pysäytetään ja suoritetaan tapahtuman oikeellisuutta suhteessa valvonnan metatietoon verifioivat valvontarutiinit. Jos tapahtuma on järjestelmän aiotun toiminnan mukainen, niin se päästetään tapahtumaan (esimerkiksi viesti päästetään läpi aiottuun kohteeseensa). Vastaavasti virheelliset tapahtumat voidaan estää ja niistä voidaan raportoida ylemmille valvontakerroksille.

Proaktiivinen valvonta on suorituskyvyn kannalta raskasta ja asettaa toteutukselle sitä kautta suurempia vaatimuksia kuin alempien kypsyystasojen valvonta. Erityisenä vaatimuksena voidaan mainita se, että proaktiivisesti toimivien valvontarutiinien on yleensä sijaittava fyysisesti lähellä sovelluksia (esimerkiksi samassa fyysisessä muistiavaruudessa). Lisäksi proaktiivisten valvontarutiinien on tarpeen säilyttää valvonnassa käyttämäänsä metatietoa lokaalisti omassa välimuistissaan (cache). Toteutusarkkitehtuurin kannalta tästä seuraa vaatimus pystyä pitämään yrityksessä mahdollisesti useissa paikoissa sijaitsevien valvontapisteiden käyttämä metatieto ajan tasalla suhteessa metatiedon primääriversioon. Proaktiivisten valvontarutiinien välimuistissaan ylläpitämät, valvotun järjestelmän tilaan ja virhetilanteisiin liittyvät, tiedot on myös pystyttävä päivittämään metatiedon keskitettyyn primääriversioon ilman liiallista ylimääräistä rasitusta (overhead).

Kuvassa 2.6 on esimerkki toteutetun valvontajärjestelmän prototyyppitoteutuksen koreografiatasolla tuottamasta lokista. Jokaisessa lokiin kirjoitetussa tietueessa on aikaleima kahdella tavoin ilmaistuna, session tunniste, keskusteluyhteyden tunniste, lähettäjäroolin tunniste, vastaanottajaroolin tunniste, tieto siitä onko havaitsija lähettävässä vai vastaanottavassa roolissa sekä viestin tyyppi. Aktiivisen ja proaktiivisen valvonnan tapauksessa voi lokissa olla lisäksi tieto valvonnan tuloksesta eli havaitusta virhetilanteesta (exception) tai aktivoituneesta tehtävävaikutuksesta (event).

Kuvan 2.6 kuvitteellisessa esimerkissä ostaja eli **Buyer** roolissa oleva taho lähettää myyjälle eli **Seller** roolissa olevalle taholle tilauspyynnön eli **placeOrderRequest** viestin. Viesti on oikeellinen ja sen lähettämisestä generoituu **PURCHASE_START** tehtävävaikutus lähetettäväksi tehtävämallia valvovalle valvontakerrokselle. Loki on kirjoitettu lähettävää roolia edustavaa sovellusta valvovan monitorin toimesta. Vastauksena tilauspyyntöön myyjä roolissa oleva taho on lähettänyt tuoteinformaatiota sisältävän **getProductInfoResponse** viestin, joka on havaittu virheelliseksi. Esimerkki sinällään on

varsin keinotekoinen, mutta se antaa kuvan toisaalta siitä, mitä nykyisen prototyypin koreografiakerros lokiin kirjoittaa ja toisaalta siitä, mitä valvontalokin ylipäätään olisi tarpeen sisältää.

Valvonnan kypsyystasojen käyttöalueiden laajuutta on visualisoitu kuvassa 2.5. Proaktiivisen valvonnan käyttöalue on kaikkein suppein. Aktiivisen valvonnan käyttöalue on laajempi kuin proaktiivisen, mutta pienempi kuin viivästetyn valvonnan. Viivästetyn valvonnan käyttöalue on laaja, mutta senkin ulkopuolelle jää asioita, joita ei automatisoidusti teoreettisista syistä pystytä tai pragmaattisista syistä kannata valvoa.

Jatkossa tämän tutkielman puitteissa keskitytään aktiiviseen ja proaktiiviseen valvontaan ja niiden algoritmiseen realisoimiseen. Valvontakerroksista käsitellään koreografiatasoa, tehtävämallitasoa, roolitasoa, ja (sopimus)sessiotasoa.

Luku 3

Valvontafunktion algoritminen toteuttaminen

Edellisessä luvussa esiteltiin pääpiirteissään liiketoimintaverkostojen automatisoituun valvontaan liittyvät abstraktiotasot. Tässä luvussa on tarkoitus pureutua tarkemmin kahteen alimpaan abstraktiotasoon: koreografiatasoon ja tehtävämallitasoon. Lisäksi käsitellään jonkin verran roolitasoa, sessiotasoa ja sopimustasoa sekä niihin liittyvää problematiikkaa. Sopimustasoa tutkitaan Web-Pilarcos projektissa omana (tästä työstä) erillisenä kokonaisuutenaan.

Koreografiatasolla valvontafunktiolla on **kaksi perustehtävää**. Valvontafunktion **ensimmäinen perustehtävä** on tukea ylempien vertikaalitasojen valvontafunktioita ja sitä kautta verkostonhallintapalveluiden toimintaa. Tämän perustehtävän toteuttamiseksi täytyy koreografiatason olla tietoinen valvomiensa keskusteluyhteyksien tilasta. Koreografiatason on myös kyettävä tämän viestinvaihtoon perustuvan tilan pohjalta päättelemään, milloin viestinvaihto on edennyt tehtävämallitasoa kiinnostavaan vaiheeseen ja tiedottamaan tehtämallitasoa siitä. Koreografiatason ensimmäisen perustehtävän (suoritusaikaiseen) toimintaan vaaditaan vähintään aktiiviseen valvontaan kykenevä koreografiatason valvontafunktio.

Koreografiatason valvontafunktion **toinen perustehtävä** on täydentää koreografioiden mallinnusvaiheessa tapahtuvaa **staattista verifiointia** mahdollistamalla viestinvaihdon oikeellisuuden suoritusaikainen todentaminen eli **dynaaminen verifiointi**. Dynaamisen verifioinnin tarkoituksena on varmistaa, että suoritusaikainen viestinvaihto tapahtuu aiemmin oikeelliseksi verifioidun koreografiakuvauksen mukaisesti. Tähän liittyen koreografiatason valvontafunktion on myös kyettävä ilmoittamaan havaituista virhetilanteista ylemmille valvontakerroksille ja mahdollisesti jopa estettävä järjestelmän virheellinen toiminta. Koreografiatason toisen perustehtävän suoritusaikaiseksi toteuttamiseksi vaaditaan vähintään aktiiviseen valvontaan kykenevä koreografiatason toteutus. Jos virheiden havaitsemisen ohella halutaan estää sovellusten virheellisen käyttäytymisen vaikutusten leviäminen, tarvitaan proaktiiviseen valvontaan kykenevä koreografiatason valvontafunktio.

Koreografiatasosta seuraava abstraktiotaso eli tehtävämallitaso pitää yllä rooliepookkiin liittyvän tehtävämallin tilaa. Kuten yksi koreografiakin, yksi tehtävämalli liittyy aina

jonkin roolin tiettyyn epookkiin ja tehtävämallitason tärkein tehtävä onkin rooliepookin elinkaareen liittyvien vaiheiden, tehtävien, seuranta. Tehtävämallitason on oltava tietoinen siitä, missä elinkaaren vaiheessa rooliepookki on kulloinkin menossa ja havaittava milloin rooliepookki loppuu. Rooliepookin elinkaaressa tapahtuvista muutoksista (mukaanlukien rooliepookin loppuminen) on aina tiedotettava tehtävämallitason päällä toimivaa roolitasoa.

Roolitason valvontafunktio ylläpitää tilatietoa siihen liittyvän hallintoalueen roolien elinkaaren tilasta. Lisäksi roolitason valvontafunktio voi pitää yllä kopiota muiden liiketoimintaverkoston osapuolten roolien elinkaarien tilasta. Oman organisaationsa rooleihin liittyvät tapahtumat roolitaso saa käyttöönsä paikalliselta tehtävämallitasolta ja muiden organisaatioiden rooleihin liittyvät tapahtumat sessiotasolta. Roolitaso välittää kaikki oman organisaationsa roolien elinkaaren tilassa tapahtuvat muutokset sessiotasolle, joka edelleen välittää ne sopimustasolle. Sopimustaso on vastuussa elinkaaren tilamuutosten levittämisestä muille liiketoimintaverkoston osapuolille.

Roolitasosta seuraava abstraktiotaso eli (sopimus)sessiotaso ylläpitää tietoa siitä, ovatko oman organisaation rooliepookit päättyneet vai edelleen käynnissä. Sessiotason tulee tietää milloin kaikki oman organisaation rooliepookit ovat päättyneet ja tiedottaa tästä ylempänä olevaa sopimustasoa. Sopimustaso synkronoi liiketoimintaverkoston laajuisen epookin vaihdon siten, että kaikki osapuolet vaihtavat epookkia yhtäaikaa. Sopimustaso edelleen tiedottaa epookinvaihdoista sessiotasoa, joka vaihtaa kunkin oman organisaationsa roolien rooliepookkia.

Kunkin valvonnan tason algoritmiseen toteuttamiseen vaaditaan sopiva metatietosisältö sekä sen varassa toimiva algoritmi. Jatkossa käydään tarkemmin läpi koreografia- ja tehtävämallitasojen käyttämät metatietosisällöt ja valvonta-algoritmit. Lisäksi luodaan katsaus rooli- ja sessiotasojen toimintaperiaatteisiin siltä osin kuin nykyisen prototyyppitoteutuksen puitteissa on mahdollista/tarpeellista.

3.1 Johdatus äärellisiin tila-automaatteihin

Tila-automaatti voidaan esittää viisikkona **(Q, E, A, n, o)** , jossa

- **Q** on äärellinen joukko tiloja **{q1, q2, ..., qk}**,
- **E** on äärellinen **siirtymäaakkosto** (input alphabet),
- **A** on äärellinen **vaikutusaakkosto** (output alphabet),
- **n** on **siirtymäfunktio n:Q x E -> Q**, joka kuvaa automaatin nykytilan ja siirtymäaakkoston merkin joksikin automaatin muista tiloista,
- **o** on **vaikutusfunktio o:Q x E -> A**, joka kuvaa automaatin nykytilan ja siirtymäaakkoston merkin joksikin vaikutusaakkoston merkiksi.

Tila-automaattiesitys on Web-Pilarcoksessa ja tässä tutkielmassa valittu liiketoimintaverkoston koreografia- ja tehtävämallikerrosten käyttämän metatiedon esitysmuodoksi.

Tila-automaattiesityksellä vältytään kiinnittäytymästä tiettyyn, epägeneeriseen, kuvauskieleen. Periaatteessa kaikille formaaleille käyttäytymiskuvauksille on olemassa niitä vastaava tila-automaattiesitys. Esimerkkinä kuvauskielistä, joilla tehdyille kuvauksille voidaan tuottaa suhteellisen suoraviivainen tila-automaattimuunnos ovat Petri-verkot [21] ja prosessialgebrat kuten pii-kalkyyli [17]. Tila-automaatit omaavat vakiintuneen aseman tietojenkäsittelytieteessä ja niiden semantiikka ja teoreettiset ominaisuudet on hyvin tutkittu ja määritelty.

Tila-automaattien merkittävin ongelma valvontametatiedon näkökulmasta on niiden koon eksponentiaalinen kasvu suhteessa mallinnettavan ongelman monimutkaisuuteen. Tällöin puhutaan usein niin sanotusta **tilaräjähdysongelmasta**. Tehtävämallitasolla tilaräjähdysongelmaan ei käytännössä koskaan törmätä, mutta koreografiatasolla tilanne on toinen. Koreografiatason malleissa esiintyy sekä monenvälisyyttä että rinnakkaisuutta, jotka molemmat kasvattavat tila-automaatin kokoa. Ongelmaa helpottaa kuitenkin se, että liiketoimintaverkostoissa käytetyt yksittäiset koreografiat ovat luonteeltaan yksinkertaisia. Usein ne koostuvat yhden, kahden tai muutaman viestin lähetyksestä.

Toinen potentiaalinen ongelma on äärellisyys. Äärellisyysogelmalla tarkoitetaan sitä, että tila-automaatin kaikki tilat, siirtymäaakkosto, vaikutusaakkosto, siirtymäfunktio ja vaikutusfunktio on tunnettava jo ennen yhteistoiminnan aloittamista. Web-Pilarcos projektin lähestymistavan mukaisessa, sopimuksiin perustuvassa liiketoimintaverkostossa, tämä ei yleensä muodostu ongelmaksi, koska koreografiat on etukäteen suunniteltu, formaalisti mallinnettu, oikeellisiksi verifioitu ja sopimukseen kirjattu. Jos kuitenkin tavoitteena olisi avoimempi (agenttiorientoitunut) ympäristö, jossa koreografioiden tarkka kulku ei olisi etukäteen tiedossa, vaan koreografia voisi (tiettyjen rajoitteiden puitteissa) muodostua minkälaiseksi hyvänsä, niin tarvittaisiin dynaamisempi tila-automaattimalli. Dynaamisessa mallissa tilat, aakkostot, siirtymä- ja vaikutusfunktio voisivat muuttua tai ne voitaisiin muodostaa yhteistoiminnan ollessa käynnissä.

Toinen tapa dynaamisuuden aikaansaamiseksi olisi koostaa pieniä staattisia tila-automaatteja suoritusaikana suuremmiksi tila-automaattikokonaisuuksiksi. Lähtökohtana tällaiselle mallille voisi toimia vaikkapa CS-WS (Conversation Support for Web Services) [6] hankkeiden kaltaiset lähestymistavat. CS-WS lähestymistavassa kokonaiskoreografiaa ei ole ennalta määritelty, vaan kaikki mahdolliset yksinkertaiset keskusteluprotokollat on mallinnettu pieninä tila-automaatteina. Suoritusaikana saapuvia ja lähteviä viestejä valvotaan siten, että viesti pyritään ensin tulkitsemaan käynnissä olevan keskustelun tila-automaatissa. Jos tämä ei onnistu pyritään koreografiavarastosta löytämään jokin sellainen tila-automaatti, jonka puitteissa viesti voidaan tulkita. Tila-automaatit muodostavat näin hierarkkisen, puumaisen ja dynaamisen rakenteen. Kokonaiskoreografia muodostuu tällöin siis vasta suoritusaikana, mutta kuitenkin toimii ennaltamääritellyssä viitekehyksessä. Edellä esitetyn kaltaisissa ympäristöissä oikeellisuuden verifiointi muodostuu kuitenkin hyvinkin hankalaksi ongelmaksi. Kuinka esimerkiksi varmistetaan, ettei dynaaminen tila-automaattikompositio lukkiudu missään tilanteessa?

Staattisen tila-automaattimallinkin tapauksessa tila-automaatit voitaisiin etukäteen pilkkoa useammaksi pienemmäksi väljästi synkronoiduksi tila-automaatiksi. Nämä erillisesti valvottavat tila-automaatit voitaisiin muodostaa esimerkiksi keskusteluyhteyskohtaisesti. Tila-automaattien toiminta synkronoitaisiin tällöin suoritusaikana niissä pisteissä, joissa

keskusteluyhteydet ovat riippuvaisia toistensa etenemisestä. Näin tarvittavien tilojen määrä pienentyisi kokonaisuudessaan selvästi. Lisäksi rinnakkaista toimintaa sisältävä yhteen keskusteluyhteyteen liittyvä tila-automaatti voitaisiin edelleen jakaa kooltaan pienemmiksi tila-automaateiksi. Tämä pienentäisi automaattien kokonaistilantarvetta entisestään.

Useista väljästi synkronoiduista tila-automaateista koostuvan koreografiakerroksen toteuttaminen on kuitenkin varsin vaativa tehtävä (varsinkin jos muodostetut tila-automaatit on hajautettu fyysisesti). Siksi tämän työn ja laaditun prototyypin puitteissa yhteen rooliepookkiin liitetään aina yksi tehtävämallitason tila-automaatti ja yksi koreografiatason tila-automaatti.

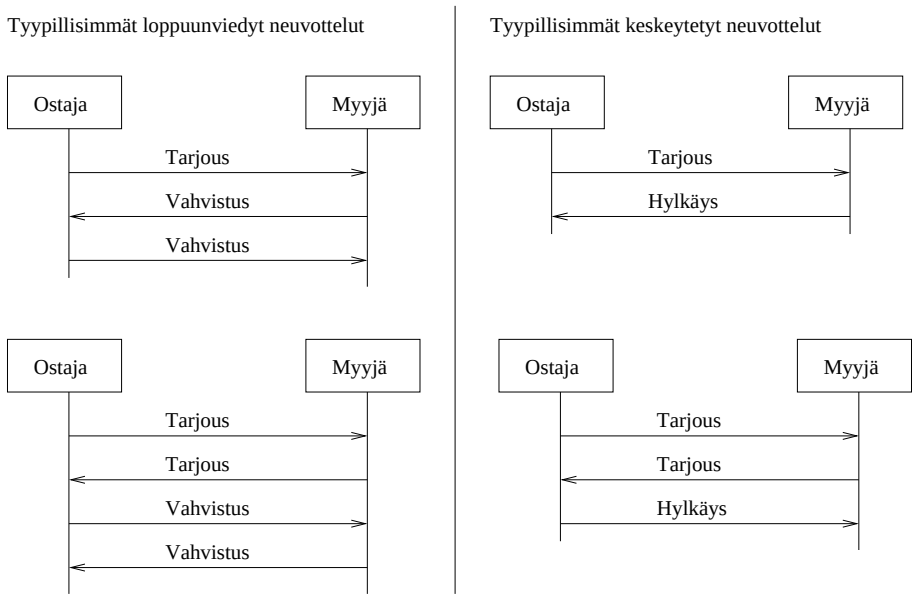
3.2 Koreografi atason valvontafunktio

Koreografiatason valvontafunktion toteuttava valvonta-algoritmi (käytetään myös termiä valvontarutiini) toimii koreografiatason valvontakerroksen ytimessä suorittaen valvontaa hyvin määritellyn metatiedon varassa. Valvontarutiinin käyttämä metatieto voidaan automaattisesti (tai lähes automaattisesti) tuottaa sähköisessä sopimuksessa olevan yhteisöarkkitehtuurin pohjalta. Sopimuksessa olevat koreografiakuvaukset on tehty jollain kuvaamiseen sopivalla kielellä. Joissain tapauksissa kielenä voi olla puhtaasti luonnollinen kieli. Kehittyneemmissä järjestelmissä kuvauskielenä voidaan käyttää UML-kieltä [33] tai liiketoimintaprosessien ja työnkulkujen kuvauksiin tarkoitettuja kieliä kuten BPEL [22] tai XPDL [39]. Tämänkaltaisista kielistä puuttuu kuitenkin usein formaalisti määritelty semantiikka, mistä johtuen kielillä tuotettujen kuvaustenkaan semantiikkaa ei ohjelmallisesti voida verifioida. Web-Pilarcos projektin (ja tämän tutkielman) puitteissa koreografiakuvauksista puhuttaessa tarkoitaan kuitenkin ennenkaikkea formaalin semantiikan omaavalla kielellä tuotettuja verifoitavissa olevia koreografiakuvauksia. Erityisesti Petriverkoilla [21] tai erilaisilla prosessialgebroilla (kuten pii-kalkyyli [17]) tai niihin perustuvilla korkeamman tason koreografiakielillä (kuten WS-CDL [35]) laadittuja malleja. Tällaiset mallit voidaan etukäteen verifioida käyttäytymiseltään oikeellisiksi ja mikä tärkeintä mallien pohjalta voidaan tuottaa äärellinen tila-automaatti ajonaikaisen valvontakoneiston käyttöön.

3.2.1 Koreografi aesimerkki: Ostajan ja myyjän välinen neuvottelu

Koreografiatason valvontametatiedon ja valvonta-algoritmin toiminnan havainnollistamiseksi on tarpeen esitellä esimerkinomaisesti kahden osapuolen, ostajan ja myyjän, väliseen yhteistoimintaan liittyvä viestinvaihtokoreografia. Tässä esitetty koreografiakuvaus on epäformaali ja sen tärkein tehtävä on tukea myöhemmin määriteltävän tila-automaattiesityksen ymmärtämistä.

Oletetaan siis tilanne, jossa liiketoimintaverkoston tiettyyn vaiheeseen eli epookkiin liittyy kaksi roolia sisältävä yhteisöarkkitehtuuri. Kaksi roolia, **Ostaja** ja **Myyjä**, käyvät keskenään neuvottelun. Oletetaan lisäksi, että roolien välillä on kyseisessä epookissa vain yksi keskusteluyhteys nimeltään **Neuvottelu**. Neuvottelun tyypillisimmät kulut on esitetty kuvassa 3.1.



Kuva 3.1: Kahdenvälisen neuvottelukoreografian tyypillisimmät kulut (graafinen esitys).

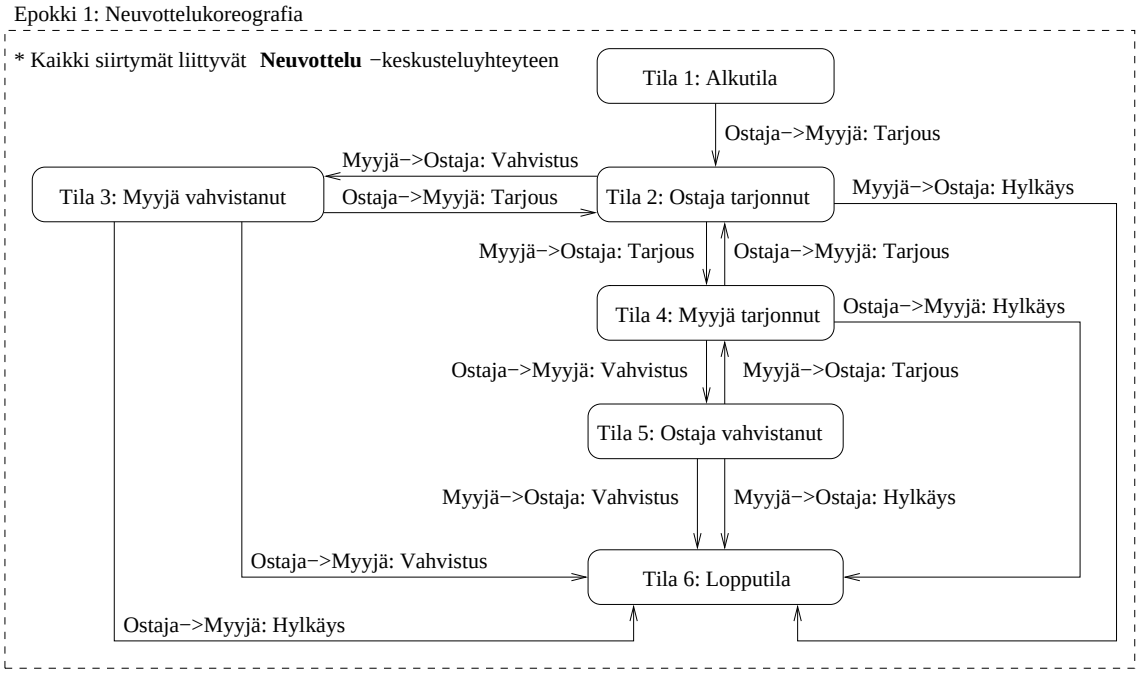
Kuvassa 3.1 vasemmalla on kaksi tyypillisintä onnistunutta neuvottelua. Vasemmalla ylhäällä olevassa skenaariossa ostaja tekee myyjälle tarjouksen lähettämällä viestin tyyppiä **Tarjous** (viestien sisältöjä ei tässä käsitellä tarkemmin). Myyjä hyväksyy tarjouksen ja vahvistaa hyväksymisen lähettämällä ostajalle viestin tyyppiä **Vahvistus**. Vahvistus on myyjän kannalta sitova. Saatuaan vahvistuksen ostaja kuittaa kaupan lähettämällä myyjälle viestin tyyppiä **Vahvistus**. Tämä vahvistus on puolestaan ostajan kannalta sitova. Edellä esiintyi kaksi neuvottelukoreografian peruskulkua. Koreografia lähtee käyntiin kun ostaja lähettää tarjouksen myyjälle ja päättyy onnistuneesti kun molemmat ovat kuitanneet kaupan vahvistuksella.

Kuvassa 3.1 vasemmalla alhaalla on laajennos ylemmästä skenaariosta, jossa myyjä ei suoraan hyväksy ostajan tarjousta, vaan lähettää siihen vastineena oman vastatarjouksensa. Tällaisia tarjous-vastatarjous kierroksia voi olla periaatteessa kuinka monta tahansa. Tässä esimerkissä kuitenkin ostaja hyväksyy myyjän ensimmäisen vastatarjouksen ja lähettää myyjälle vahvistuksen. Myyjä puolestaan kuittaa neuvottelun loppumisen omalla vahvistuksellaan.

Kuvan 3.1 oikealla puolella on kaksi tyypillisintä esimerkkiä epäonnistuneesta neuvottelusta. Oikeassa yläkulmassa on skenaario, jossa myyjä hylkää heti ostajan lähettämän tarjouksen **Hylkäys**-viestillä. Oikeassa alakulmassa on skenaario, jossa myyjä lähettää ostajalle vastatarjouksen, mutta ostaja hylkää vastatarjouksen **Hylkäys**-viestillä havaitessaan, että myyjän vastatarjous on liian kaukana omasta ehdotuksestaan. Kuten onnistuneenkin neuvottelun tapauksessa, myös epäonnistuneessa tilanteessa voi neuvottelukierroksia olla useita ennen hylkäyspäätöksen syntymistä. Siinä missä onnistunut neuvottelu päättyy aina kahteen **Vahvistus**-viestiin, neuvottelun epäonnistumiseen riittää yksi **Hylkäys**-viesti - kummalta tahansa osapuolelta.

3.2.2 Neuvottelukoreografi an tila-automaattiesitys

Yleisessä tapauksessa neuvottelukoreografia voidaan mallintaa äärellisenä tila-automaattina, jossa jokainen mahdollinen viestien vaihto kuvataan omana tilasiirtymänään. Kuvassa 3.2 on esitetty neuvotteluprotokollan tila-automaatti kokonaisuudessaan. Automaatissa on kuusi tilaa joista tila 1 on alkutila, tila 6 on lopputila ja muut tilat ovat välitiloja. Tila-automaatti käynnistyy kun ostaja lähettää **Tarjous**-viestin myyjälle. Tämän jälkeen jokainen ostajan ja myyjän välillä tapahtuva viestinvaihto siirtää automaatin tilaa, kunnes automaatti on päätynyt **Hylkäys**-viestin tai kahden peräkkäisen **Vahvistus**-viestin seurauksena lopputilaansa. Lopputilaan saavuttaessa automaatin epookki loppuu ja (liiketoimintaverkostosta riippuen) automaatti joko jää lopputilaansa, alustetaan uudestaan tai korvataan uudella automaatilla.

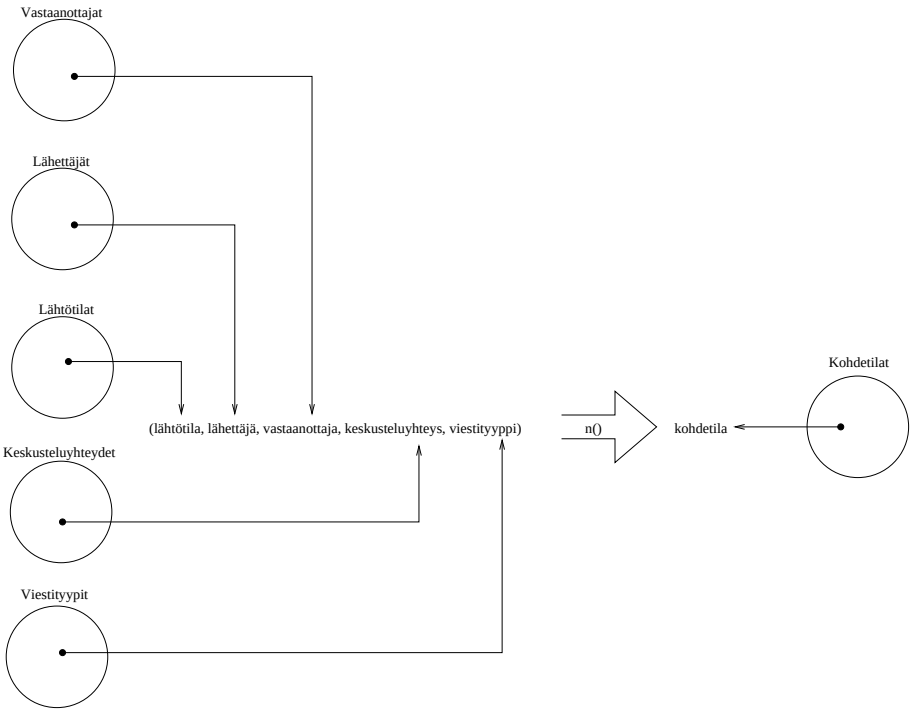


Kuva 3.2: Kahdenväliseen neuvottelukoreografiaan liittyvä tila-automaatti (graafinen esitys).

Kuvassa 3.2 esitettyyn tila-automaattiin liittyvän **Neuvottelu**-epookkiin kuuluva tehtävämalli on esitetty kuvassa 3.10. Tehtävämalleihin tutustutaan tarkemmin seuraavassa kappaleessa.

Kuten kuvasta 3.2 voidaan nähdä, seuraavaksi suoritettavan tilasiirtymän suhteessa **lähtötilaan** yksilöi nelikko **(lähettäjä, vastaanottaja, keskusteluyhteys, viestityyppi)**. Tila-automaatin siirtymäfunktio voidaankin esittää muodossa **y = n(lähtötila, lähettäjä, vastaanottaja, keskusteluyhteys, viestityyppi)**, jossa **y**:n mahdolliset arvot kuuluvat automaatin kohdetilojen **kohdetila = tilat / alkutilat** joukkoon.

Koska siirtymäfunktion jokainen maalijoukon alkio on ainakin yhden lähtöjoukon alkion kuva (eli jokaiseen tila-automaatin kohdetilaan on olemassa siirtymä jostain toisesta



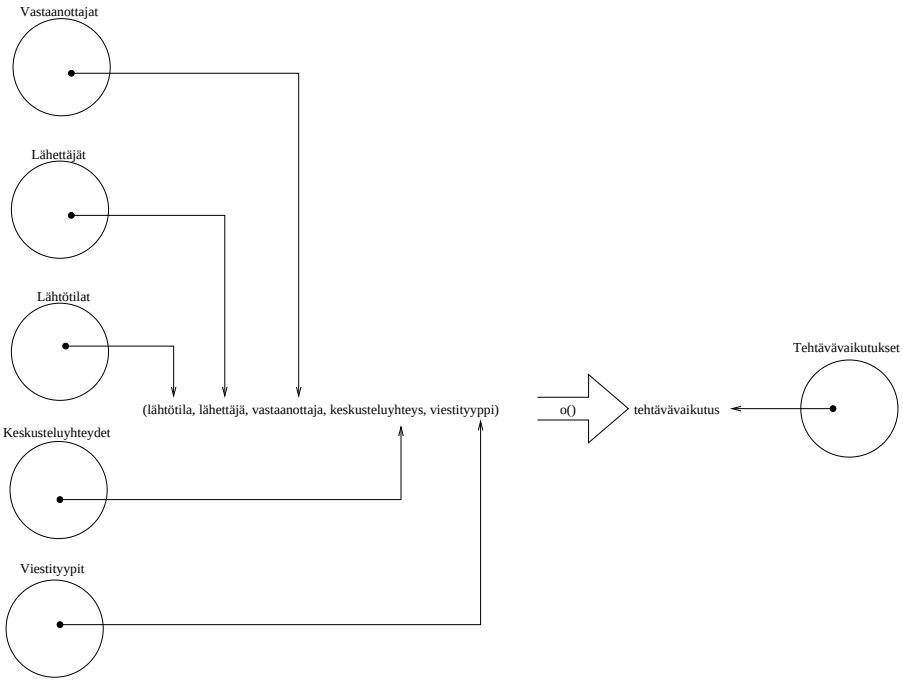
Kuva 3.3: Koreografiatason tila-automaatin siirtymäfunktio (graafinen esitys).

tila-automaatin tilasta), on kyseessä surjektiivinen funktio eli surjektio. Toisaalta, koska jokainen maalijoukon alkio voi olla useamman kuin yhden alkion kuva (eli jokaiseen tila-automaatin kohdetilaan voi olla siirtymä useammasta kuin yhdestä muusta tilasta) kyseessä ei ole injektio eikä siten myöskään bijektio.

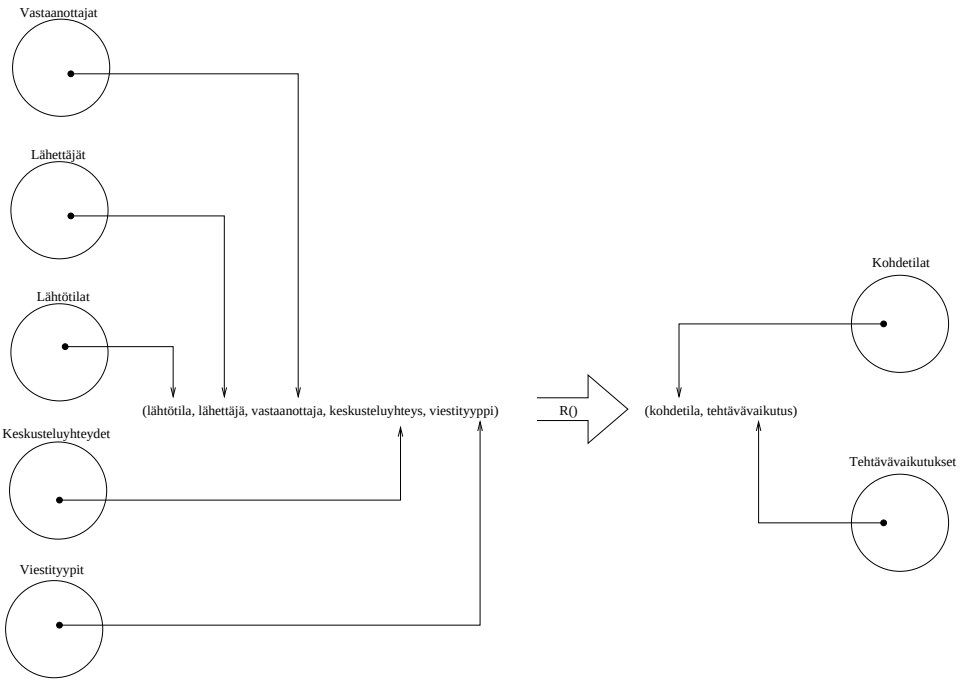
Esitetty tila-automaattimalli muodostaa liiketoimintaverkoston suoritusaikaisen verifioinnin vaatimuksiin vastaavan valvontametatietomallin ja täyttää siten koreografiatason valvontakerroksen toisen päätehtävän tarpeet. Metamallin määrittelevä kuvaus on esitetty graafisesti kuvassa 3.3.

Tehtävämallitason tilaseurannan mahdollistamiseksi täytyy koreografiatason tila-automaattiin liittää myös vaikutusfunktio. Kuhunkin tila-automaatin tilasiirtymään pitää olla mahdollista liittää tehtävämallitason tilasiirtymän aktivoiva **tehtävävaikutus**. Koreografiatason metatietomalli määrittelee siis siirtymäfunktion lisäksi kuvassa 3.4 graafisesti esitetyn vaikutusfunktion **y = o(lähtötila, lähettäjä, vastaanottaja, keskusteluyhteys, viestityyppi)**, jossa **y**:n mahdolliset arvot kuuluvat metatietomalliin liittyvän rooliepookin tehtävävaikutusten joukkoon.

Kokonaisuudessaan koreografiatason valvontakerroksen metatietomallin määrittelee siis tila-automaatti, jonka virittää kuvassa 3.5 esitetty relaatio **Y = R(lähtötila, lähettäjä, vastaanottaja, keskusteluyhteys, viestityyppi)**, jossa **Y**:n mahdolliset arvot muodostuvat järjestetyistä pareista **(q,a)** siten, että **q** kuuluu tila-automaatin kohdetilojen joukkoon ja **a** kuuluu rooliepookkiin liittyvän tehtävämallin sisältämien tehtävävaikutusten joukkoon. Tilanteissa, joissa koreografiatason tilasiirtymä ei aiheuta lainkaan tehtävävaikutusta saa **a** arvokseen erityisen **tyhjän vaikutuksen** (null effect).



Kuva 3.4: Koreografiatason tila-automaatin vaikutusfunktio (graafinen esitys).



Kuva 3.5: Koreografiatason tila-automaatin virittävä relaatio (graafinen esitys).

Koreografiatason metatietomalli voidaan valvonta-algoritmia varten koodata tietorakenteiksi useilla erilaisilla tavoilla. Tämän työn puitteissa määritellään kaksi erilaista koo-

daustapaa: matriisiesitys ja hajautustauluesitys. Lisäksi esitellään kumpaakin koodaustapaa hyödyntävät valvonta-algoritmit.

3.2.3 Tila-automaatin matriisiesitys ja sitä hyödyntävä algoritmi

Ehkä yksinkertaisin tapa koodata koreografiatason valvontametatietona toimiva tila-automaatti algoritmiseen muotoon on suoraviivainen matriisiesitys. Matriisiesityksessä tila-automaatin jokainen tilasiirtymä ja siihen mahdollisesti liittyvä tehtävävaikutus muunnetaan yhdeksi matriisin riviksi. Kuvassa 3.6 on matriisiesitys kuvassa 3.2 esitetystä tila-automaatista. Yhtä tilasiirtymää kuvaava matriisin rivi sisältää seitsemän saraketta. Sarake sisältää tilasiirtymään liittyvän **lähtötilan**, tilasiirtymän laukaisevaan viestin liittyvät **lähettäjän**, **vastaanottajan**, **keskusteluyhteyden** ja **viestityypin** sekä tilasiirtymän **kohdetilan**. Nämä sarakkeet määrittelevät dynaamiseen verifiointiin tarvittavan tila-automaatin. Seitsemänteen sarakkeeseen kirjataan tilasiirtymän mahdollisesti aiheuttama **tehtävävaikutus**.

Kuvan 3.6 tila-automaatin tapauksessa tietyt tila-automaatin siirtymät laukaisevat **Neuvottelu_Alkoi**, **Neuvottelu_Keskeytyi** tai **Neuvottelu_Päättyi** -tehtävävaikutukset. **Neuvottelu_Alkoi** tehtävävaikutus syntyy, kun tila-automaatti siirtyy pois alkutilastaan. **Neuvottelu_Päättyi** tehtävävaikutus syntyy kahden perättäisen (eri osapuolilta tulevan) vahvistuksen jälkeen. **Neuvottelu_Keskeytyi** tehtävävaikutus syntyy jommalta kummalta osapuolelta tulevan hylkäämisen jälkeen.

Lähtötila	Lähettäjä	Vastaanottaja	Keskusteluyhteys	Viestityyppi	Kohdetila	Tehtävävaikutus
1	Ostaja	Myyjä	Neuvottelu	Tarjous	2	Neuvottelu_Alkoi
2	Myyjä	Ostaja	Neuvottelu	Vahvistus	3	
2	Myyjä	Ostaja	Neuvottelu	Tarjous	4	
2	Myyjä	Ostaja	Neuvottelu	Hylkäys	6	Neuvottelu_Keskeytyi
3	Ostaja	Myyjä	Neuvottelu	Tarjous	2	
3	Ostaja	Myyjä	Neuvottelu	Hylkäys	6	Neuvottelu_Keskeytyi
3	Ostaja	Myyjä	Neuvottelu	Vahvistus	6	Neuvottelu_Päättyi
4	Ostaja	Myyjä	Neuvottelu	Tarjous	2	
4	Ostaja	Myyjä	Neuvottelu	Vahvistus	5	
4	Ostaja	Myyjä	Neuvottelu	Hylkäys	6	Neuvottelu_Keskeytyi
5	Myyjä	Ostaja	Neuvottelu	Tarjous	4	
5	Myyjä	Ostaja	Neuvottelu	Hylkäys	6	Neuvottelu_Keskeytyi
5	Myyjä	Ostaja	Neuvottelu	Vahvistus	6	Neuvottelu_Päättyi

Kuva 3.6: Kahdenvälisen neuvottelukoreografian tilasiirtymät (matriisiesitys).

Matriisimuotoista tila-automaattia hyödyntävä valvonta-algoritmi on esitetty kuvas-

sa 3.7. Kuten kuvasta 3.7 voidaan suoraviivaisesti nähdä, huonoimmassa tapauksessa (worst case), eli tapauksessa jossa algoritmi joutuu käymään silmukassa kaikki tila-automaatin rivit läpi löytämättä oikeellista tilasiirtymää, on automaatin aikavaativuus lineaarista suuruusluokkaa

$$O(n)$$

suhteessa tila-automaattiin sisältyvien tilasiirtymien määrään **n**.

Algoritmi **Verifi oi**(Viestin mukana kulkeneet metatiedot M):

1. Kirjoita lokimerkintä johon sisältyy metatiedon M mukana kulkevat tiedot sekä aikaleima.
2. Eristä viestin mukana kulkeneista metatiedoista M session tunniste S.
3. Hae paikallisesta välimuistista sessioon S liittyvä matriisimuotoinen koreografi an tila-automaatti T.
   (a) Jos ei löydy niin tee sama haku metatietoa välittävältä rajapinnalta ja talleta T paikalliseen välimuistiin assosioituna sessioon S.
   (b) Jos kelvollista tila-automaattia T ei edelleenkään löydy paikallisesta välimuistista niin
   - Viivästetyn valvonnan tapauksessa tee asiasta lokimerkintä ja palaa operaatiosta normaalisti.
   - Aktiivisen valvonnan tapauksessa lähetä lokimerkinnän lisäksi asiasta notifi kaatio tehtävämallitasolle.
   - Proaktiivisen valvonnan tapauksessa edellisten lisäksi palaa algoritmin suorituksesta poikkeuksellisesti virheenä verifi oinnin epäonnistuminen ja virheen syynä virheellinen session tunniste (metadataa ei löydetty).
4. Käy silmukassa läpi matriisimuotoisen tila-automaatin T rivejä $T_1..T_n$
   (a) Tutki tila-automaatin riviä T_i
      i. Onko $T_i.lahtotila$ sama kuin $T.nykytila$?
      ii. Onko $T_i.lahettaja$ sama kuin $M.lahettaja$?
      iii. Onko $T_i.vastaanottaja$ sama kuin $M.vastaanottaja$?
      iv. Onko $T_i.keskusteluyhteys$ sama kuin $M.keskusteluyhteys$?
      v. Onko $T_i.viestityyppi$ sama kuin $M.viestityyppi$?
   - Jos kaikki kentät täsmäävät, niin rivi esittää kelvollista tilasiirtymää, joten,
      i. Tee tilasiirtymä päivittämällä $T.nykytila$ kenttään $T_i.kohdetila$ kentän arvo.
      ii. Jos $T_i.tehtavavaikutus$ kenttä sisältää kelvollisen arvon, niin tee asiasta lokimerkintä ja lähetä lisäksi lähetä kentän arvo tehtävämallikerrokselle ja käsittele paluuarvona saatu ohje O.
         A. Jos O käskee invalidoimaan automaatin T niin poista se välimuistista.
         B. Jos O käskee säilyttämään olemassaolevan automaatin niin talleta T, jonka nykytila on päivitetty, paikalliseen välimuistiin assosioituna sessioon S.
         C. Jos virheensietoprotokolla on käytössä niin talleta automaatin nykytila $T.nykytila$ persistenttiin muotoon myöhempää toipumisprotokollaa varten.
         D. Lopeta algoritmin suorittaminen onnistuneesti.
5. Jos silmukka päättyi ilman, että sopivaa tilasiirtymää löytyi seurauksena on verifi oinnin epäonnistuminen.
   - Viivästetyn valvonnan tapauksessa tee asiasta lokimerkintä ja palaa operaatiosta normaalisti.
   - Aktiivisen valvonnan tapauksessa lähetä tehtävämallitasolle tieto virhetilanteesta ennen normaalia operaatiosta palaamista.
   - Proaktiivisen valvonnan tapauksessa lokimerkinnän ja virhetiedon lähetyksen lisäksi palaa algoritmin suorituksesta poikkeuksellisesti virheenä verifi oinnin epäonnistuminen ja syynä se, ettei sopivaa tilasiirtymää löydetty.

Kuva 3.7: Matriisimuotoon koodattua tila-automaattia hyödyntävä valvonta-algoritmi.

3.2.4 Tila-automaatin hajautustauluesitys ja sitä hyödyntävä algoritmi

Matriisimuotoista tila-automaattia hieman monimutkaisempi, mutta aikavaativuudeltaan kevyempi, tapa koodata koreografiatason tila-automaatti on hajautustauluesitys. Hajautustauluesitys voidaan konstruoida lähes yhtä suoraviivaisesti tila-automaatin siirtymien pohjalta kuin matriisiesityskin. Tila-automaatin hajautustauluesityksessä kukin tila-automaatin

siirtymä ja siihen mahdollisesti liittyvä tehtävävaikutus koodataan kuvassa 3.5 esitetyn relaation mukaisesti hajautusavaimeksi ja sitä vastaavaksi taulukkoriviksi.

Kuvassa 3.8 on neuvottelukoreografian tila-automaattia vastaava hajautustauluesitys. Hajautustauluesitys on luonteeltaan hyvin lähellä kuvassa 3.6 esitettyä matriisiesitystä. Kukin kuvan 3.6 matriisin rivi on nyt vain jaettu kahteen osaan: hajautusavaimeen ja relaation maalijoukkoa esittävään kaksisarakkeiseen matriisiin. Ideana hajautustauluesityksessä on määritellä hajautusavain sekä hajautusfunktio, joka kuvaa kunkin hajautusavaimen yksikäsitteisesti jonkin kaksisarakkeisen kohdematriisin rivin indeksiksi. Kuvassa 3.8 on esimerkki hajautusfunktio nimeltä **Hash()**, joka kuvaa kenttien **Lähtötila**, **Lähettäjä**, **Vastaanottaja**, **Keskusteluyhteys** ja **Viestityyppi** katenoiduista arvoista koostuvan avaimen yksikäsitteisesti kohdematriisin rivin indeksiksi. Kohdematriisina on kaksisarakkeinen matriisi, jonka valituksi tullut rivi sisältää **Kohdetila** ja **Tehtävävaikutus** kenttien arvoista muodostuvan parin.

Esimerkiksi kuvan 3.8 ylimmäisellä rivillä funktio

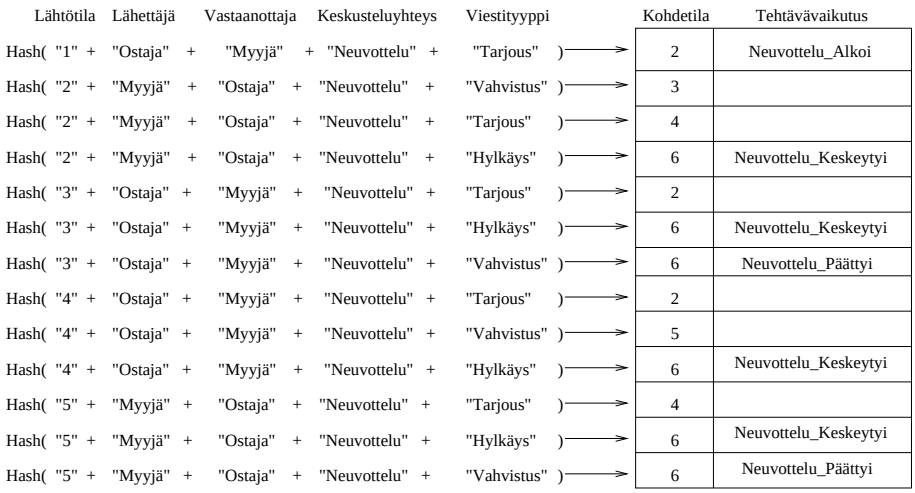
$$Hash(\text{"1"} + \text{"}Ostaja\text{"} + \text{"}Myyja\text{"} + \text{"}Neuvottelu\text{"} + \text{"}Tarjous\text{"})$$

kuvaa katenoimalla aikaansaadun hajautusavaimen **1OstajaMyyjaNeuvotteluTarjous** kohdematriisin indeksiksi **0**. Kyseiseltä matriisin riviltä löytyvät tilasiirtymän kohdetila **2** ja siirtymän aiheuttama tehtävävaikutus **Neuvottelu_Alkoi**.

Hajautustaulu voidaan käytännössä konstruoida useilla eri tavoilla samoin kuin hajautusavain ja -funktiokin. Tämän työn puitteissa ei kiinnitetä mitään yksittäistä konstruointitapaa. Olennaista on lähinnä se, että hajautustaulun avain rakentuu jollain tapaa edellä esitetyn viiden kentän arvosta ja valittu hajautusfunktio kuvaa avaimen yksikäsitteisesti sitä vastaavaksi **(kohdetila, tehtävävaikutus)** pariksi. Olemassa olevissa ohjelmointikielissä on usein valmiita välineitä hajautustaulujen toteuttamiseen. Esimerkiksi Java-kielessä valmiina olevat luokat **java.util.HashMap** tai **java.util.Hashtable** soveltuvat suoraan edellä esitetyn kaltaisen hajautustaulurakenteen toteuttamiseen. Erona näissä toteutuksissa on, että hajautusavain ei kuvaudu kohdematriisin rivin indeksiksi, vaan jokainen **(kohdetila, tehtävävaikutus)** pari talletetaan erillisenä objektina jokaista hajautusavainta vasten. Java-luokat piilottavat näin hajautustaulun fyysisen toteutustavan käyttäjältä ja käyttäjän vastuulle jää ainoastaan sitoa kutakin tila-automaatin riviä edustava objekti sopivaan hajautusavaimeen.

Hajautustaulumuotoista tila-automaattia hyödyntävä valvonta-algoritmi on esitetty kuvassa 3.9. Algoritmia suoritettaessa aikaa kuluu olennaisesti hajautusavaimen muodostamiseen lähtöarvoista sekä matriisin indeksin laskemiseen avaimen pohjalta. Koska hajautusavaimen muodostaminen ja matriisin indeksin laskeminen ovat (lähes) vakioajassa tapahtuvia operaatioita, on algoritmin aikavaativuus vakio suhteessa kaikkiin olennaisiin muuttujiin. Ainoa aikavaativuuteen teoriassa vaikuttava parametri on muodostettavan hajautusavaimen ja sen lähtöarvojen koko. Koska lähtöarvoina toimivissa merkkijonoissa on oletettavasti käytetty lyhyitä (ihmisen luettavaksi suunniteltuja) merkkijonoja, ovat hajautusavaimen muodostaminen ja hajautusfunktion arvon laskeminen kuitenkin käytännössä kestoltaan vakioajassa tapahtuvia toimenpiteitä.

Suurissa tila-automaateissa hajautustaulukoodauksella saavutetaan siis potentiaalista suorituskykyetua matriisikoodaukseen nähden. Yleisesti käytetyt liiketoimintaprotokollat



Kuva 3.8: Kahdenvälisen neuvottelukoreografian tilasiirtymät (hajautustauluesitys).

muodostavat kuitenkin oletettavasti niin suppeita tila-automaatteja ettei käytetyllä koodauksella ole juurikaan merkitystä suorituskyvyn kannalta. Matriisikoodauksella on kuitenkin puolellaan se etu, että matriisikoodatut tila-automaatit on helppo kuljettaa minkä tahansa etärajapinnan (esimerkiksi Web-palvelurajapinnan) läpi. Hajautustaulumuotoisten tila-automaattien tapauksessa joudutaan potentiaalisesti suorittamaan monimutkainen tietomuunnos paikallisten esitystapojen ja etärajapinnan käyttämän esitystavan välillä.

3.3 Tehtävämallitason valvontafunktio

Tehtävämallitasolla toimivalla valvontakerroksella on ennenkaikkea rooliepookin elinkaaren seurantaan liittyvä tehtävä. Tehtävämalli kuvaa tietyllä ajanhetkellä aktiivisena olevan rooliepookin tilaa. Tehtävämalli onkin pohjimmiltaan tila-automaatti. Tehtävämallikerros ottaa vastaan tehtävävaikutuksia koreografiaa valvovalta kerrokselta ja päättelee näiden perusteella miten tehtävämallin tilaa pitää muuttaa.

Tehtävämallin vaikutusfunktio on hyvin yksinkertainen. Vaikutusfunktio palauttaa tyhjän arvon muulloin kuin silloin, kuin suoritettava tilasiirtymä siirtää automaatin lopputilaansa. Jälkimmäisessä tapauksessa vaikutusfunktio palauttaa epookin loppumista ilmaisevan arvon. Tehtävämalli voidaankin kuvata relaationa **Y = R(lähtötila, tehtävävaikutus)**, jossa **Y**:n alkiot koostuvat järjestetyistä pareista **(q,a)** siten, että kukin **q**:n arvo kuuluu tehtävämallin kohdetilojen **kohdetila = tilat / alkutilat** joukkoon ja **a** saa epookin loppumista ilmaisevan arvon silloin, kun **q** on tehtävämallin lopputila ja erityisen **tyhjän arvon** muutoin.

Varsinaisen tila-automaatin lisäksi tehtävämalliin lasketaan sisältyväksi kyky pitää yllä tila-automaatin **suoritusjälkeä** (trace). Suoritusjälkeä ei tarvita suoritusaikaisen valvontakoneiston käyttöön, vaan sen avulla on mahdollista jälkikäteen visualisoida epookin elinkaaren vaiheiden temporaalinen eteneminen. Tämä ominaisuus vastaa erityisesti Web-Pilarcos projektissa tuotettavan hallintapaneelin tarpeisiin. Tehtävämallin suoritus-

Algoritmi **Verifi oi**(Viestin mukana kulkeneet metatiedot M):

1. Kirjoita lokimerkintä johon sisältyy metatiedon M mukana kulkevat tiedot sekä aikaleima.
2. Eristä viestin mukana kulkeneista metatiedoista M session tunniste S.
3. Hae paikallisesta välimuistista sessioon S liittyvä hajautustaulumuotoinen tila-automaatti T.
   (a) Jos ei löydy niin tee sama haku metatietoa välittävältä rajapinnalta ja talleta T paikalliseen välimuistiin assosioituna sessioon S.
   (b) Jos kelvollista tila-automaattia T ei edelleenkään löydy paikallisesta välimuistista niin
   - Viivästetyn valvonnan tapauksessa tee asiasta lokimerkintä ja palaa operaatiosta normaalisti.
   - Aktiivisen valvonnan tapauksessa lähetä lokimerkinnän lisäksi asiasta notifi kaatio tehtävämallitasolle.
   - Proaktiivisen valvonnan tapauksessa edellisten lisäksi palaa algoritmin suorituksesta poikkeuksellisesti virheenä verifi oinnin epäonnistuminen ja virheen syynä virheellinen session tunniste (metadataa ei löydetty).
4. Muodosta hajautusavain A kentistä $T.nykytila$, $M.lahettaja$, $M.vastaanottaja$, $M.keskusteluyhteys$ ja $M.viestityyppi$
5. Hae hajautustaulusta avainta A vastaava (kohdetila K, tehtävävaikutus V) pari.
   (a) Jos sopivaa (K, V) paria ei löytynyt, niin algoritmin suoritus päättyy ilman, että sopivaa tilasiirtymää löytyi. Seurauksena on verifi oinnin epäonnistuminen eli algoritmin suoritus lopetetaan epäonnistuneesti virheenä verifi oinnin epäonnistuminen ja syynä se, ettei sopivaa tilasiirtymää löydetty.
   (b) Jos sopiva (K, V) pari löytyi, niin kelvollinen tilasiirtymä löytyi joten,
      i. Tee tilasiirtymä päivittämällä $T.nykytila$ kenttään kohdetilan K arvo.
      ii. Jos tehtävävaikutus V sisältää kelvollisen arvon, niin tee asiasta lokimerkintä ja lähetä kentän arvo tehtävämallitasolle ja käsittele paluuarvona saatu ohje O.
         A. Jos O käskee invalidoimaan automaatin T niin poista se välimuistista.
         B. Jos O käskee säilyttämään olemassaolevan automaatin niin talleta T, jonka nykytila on päivitetty, paikalliseen välimuistiin assosioituna sessioon S.
         C. Jos virheensietoprotokolla on käytössä niin talleta automaatin nykytila $T.nykytila$ persistenttiin muotoon myöhempää toipumisprotokollaa varten.
         D. Lopeta algoritmin suorittaminen onnistuneesti.
6. Jos hajautusavainta vastaavaa paria (K, V) ei löytynyt seurauksena on verifi oinnin epäonnistuminen.
   - Viivästetyn valvonnan tapauksessa tee asiasta lokimerkintä ja palaa operaatiosta normaalisti.
   - Aktiivisen valvonnan tapauksessa lähetä tehtävämallitasolle tieto virhetilanteesta ennen normaalia operaatiosta palaamista.
   - Proaktiivisen valvonnan tapauksessa lokimerkinnän ja virhetiedon lähetyksen lisäksi palaa algoritmin suorituksesta poikkeuksellisesti virheenä verifi oinnin epäonnistuminen ja syynä se, ettei sopivaa tilasiirtymää löydetty.

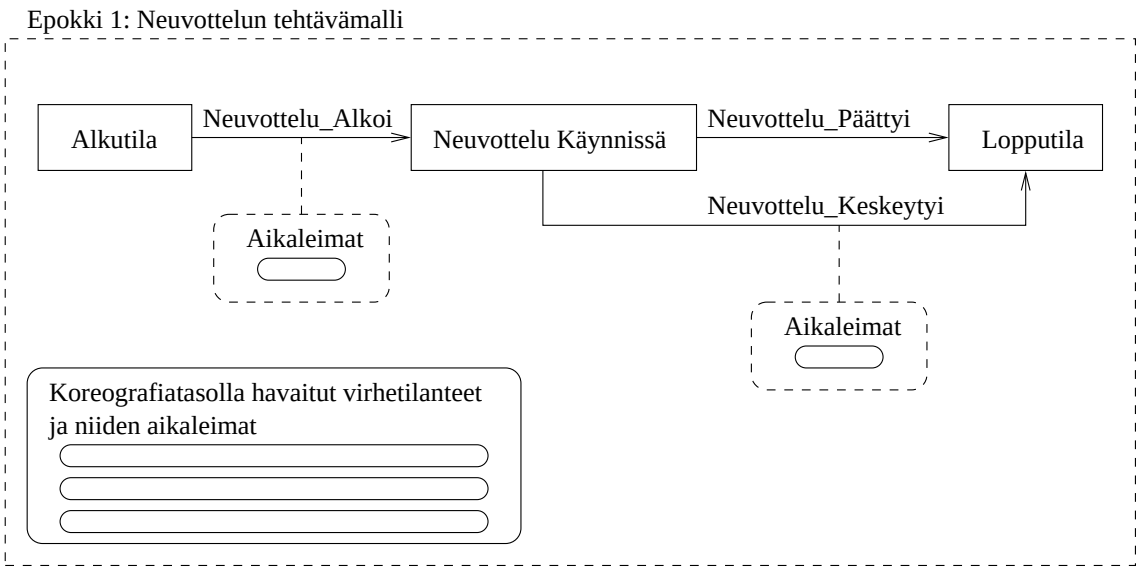
Kuva 3.9: Hajautustaulumuotoon koodattua tila-automaattia hyödyntävä valvonta-algoritmi.

jälki koostuu tehdyn tilasiirtymän identifioivasta tunnisteesta sekä aikaleimasta. Suoritusjälki on siis jono **(aikaleima, lähtötila, tehtävävaikutus, kohdetila)** nelikoita.

Suoritusjäljen ohella tehtävämalli pitää kirjaa koreografiatasolta saapuneista virhetilanneilmoituksista. Virhetilanteiden suoritusjälkeä voidaan kutsua **virhejäljeksi**. Virhejälki koostuu jonosta **(aikaleima, koreografiatasolla havaittu virhetilanne)** pareja. Virhejäljenkään tarkoitus ei ole tukea varsinaista valvontatehtävää, vaan vastata Web-Pilarcos projektissa tuotettavan hallintapaneelin tarpeisiin.

Tehtävämallitaso muodostaa tilatiedon abstraktiorajan, jota konkreettisemmat tilatiedot ovat puhtaasti organisaation sisäisiä ja luonteeltaan teknisiä. Tehtävämallitason ja sitä abstraktimman tason tilatiedot voidaan halutessa lähettää suoritusaikaisesti muille liiketoimintaverkoston osapuolille tai niitä voidaan käyttää organisaation sisäisesti liiketoimintaverkoston nykyisen tilan ja tilahistorian visualisointiin.

Kuvassa 3.10 on visualisoitu neuvottelukoreografian tehtävämalli. Tehtävämalli on alustuksensa jälkeen alkutilassaan **Alkutila**, josta se siirtyy **Neuvottelu_Alkoi** tehtävä-



Kuva 3.10: Kahdenvälisen neuvottelukoreografian tehtävämalli.

vaikutuksen seurauksena **Neuvottelu Käynnissä** vaiheeseen. Tämän jälkeen tehtävämalli voi päätyä **Neuvottelu_Päättyi** tai **Neuvottelu_Keskeytyi** tehtävävaikutusten seurauksena **Lopputila** vaiheeseen, joka käytännössä myös päättää rooliepookin. Tehtävämallin virittävä tila-automaatti voidaan koodata matriisi- tai hajautustaulumuotoon samoilla periaatteilla kuin koreografiatason tila-automaattikin.

Tilatiedon ja historiatiedon ylläpitämisen ohella tehtävämallikerroksella on valvontatehtävä, joka liittyy rooliepookkien päättymisen havaitsemiseen. Tehtävämallikerros havaitsee rooliepookin päättymisen siitä, että tehtävämalli päätyy lopputilaansa. Tehtävämallikerros ilmoittaa havainnostaan roolikerrokselle.

Algoritmi **Käsittele**(koreografi atason lähettämä tehtävävaikutus V):

1. Eristä tehtävävaikutuksesta V session tunniste S ja roolin tunniste R.
2. Pyydä roolitasolta session S roolin R aktiivisen rooliepookin tunniste E.
3. Tee rooliepookkiin E liittyvään tehtävämalliin M tehtävävaikutusta V vastaavan tilasiirtymä T.
4. Liitä tehtävämalliin M merkintä T:n tapahtumisajasta.
5. Tutki päätyikö tehtävämalli M lopputilaansa,
   - Jos kyllä, niin ilmoita roolitasolle rooliepookin E päättymisestä ja palaa operaatiosta paluuarvolla, joka pyytää koreografi atasoa invalidoimaan välimuistissaan olevan, sessioon S liittyvän, metatiedon.
   - Jos ei, niin ilmoita roolitasolle rooliepookin elinkaaren vaihesiirtymästä ja palaa operaatiosta paluuarvolla, joka pyytää koreografi atasoa säilyttämään välimuistissaan olevan, sessioon S liittyvän, metatiedon.

Kuva 3.11: Tehtävämallitason algoritmi tehtävämallivaikutusten käsittelyyn (geneerinen).

Tehtävämallin valvontatehtäväksi voidaan laskea myös koreografiatason välimuistin käytön ohjaaminen. Epookin vaihtuessa koreografiatason käyttämä metatieto poistetaan ja

korvataan uudella. Valvontakerrosten vertikaaliseen hajautusarkkitehtuuriin pureudutaan vasta toteutusta käsittelevässä pääkappaleessa, mutta jo tässä yhteydessä voidaan asiaa käsitellä jonkin verran.

Koreografiatason ja tehtävämallitason rajapinta on hallinnallisista syistä asymmetrinen ja aloitteentekijänä kommunikaatioon toimii aina koreografiataso. Tehtävämallitasolla ei näin ollen ole mahdollisuutta proaktiivisesti ilmoittaa koreografiatasolle epookin loppumisesta tai toimittaa sille uuteen epookkiin liittyvää metatietoa. Asymmetrisessä rajapintatoteutuksessa metatiedon päivitykset hoidetaan aina siten, että koreografiatason täytyy tehdä aloite ja pyytää tehtävämallitasolta tiettyyn sessioon liittyvää metatietoa. Tehtävämallitaso voi kuitenkin liittää ("piggy pack") saapuvien tehtävävaikutusten ja virheiden notifikaatioihin paluuarvona tiedon siitä, voiko koreografiataso edelleen käyttää aiemmin noutamaansa metatietoa vai pitääkö se invalidoida.

Koreografiatason metatiedon invalidoiminen tulee kyseeseen lähinnä epookin loppumisen yhteydessä sekä silloin, jos jokin koreografiatason havaitsema ja lähettämä virhetilanne johtaa muutoksiin verkoston konfiguraatiossa. Virhetilanteiden korkeammilla abstraktiotasoilla tapahtuvaa **automatisoitua tulkintaa** ei kuitenkaan tässä työssä käsitellä.

Kuvassa 3.11 on esitetty geneerinen algoritmi tehtävämallikerroksen toiminnalle silloin, kun se vastaanottaa koreografiatasolta tehtävävaikutuksen. Vastaava tehtävävaikutus voi tulla myös roolitasolta. Tällöin kyseessä on tilanne, jossa jokin toinen organisaatio päivittää muille organisaatioille epookin elinkaaren vaihesiirtymiä. Roolitasolta saapuva ilmoitus käsitellään periaatteessa samalla tavoin kuin koreografiatasoltakin saapuva ilmoitus, mutta roolitason ilmoitukset eivät johda mihinkään jatkotoimenpiteisiin. Tehtävämallitaso valvoo siis vain oman organisaationsa rooliepookkeja loppumista. Muiden organisaatioiden alaisuuteen kuuluvien roolien tapauksessa pidetään vain yllä niihin liittyvien rooliepookkien tilaa. Kuvassa 3.12 on kuvattu vastaava algoritmi tehtävämallitason toiminnalle, silloin kun se vastaanottaa koreografiatasolla havaittuun virheeseen liittyvän ilmoituksen.

Algoritmit ovat geneerisiä siinä mielessä, että niissä ei tehdä eroa käytetyn tila-automaattikoodausten välille (esimerkiksi matriisi- ja hajautustaulukoodaukset). Algoritmien toteutuksissa tila-automaatin läpikäynti voidaan tehdä esimerkiksi koreografiatason algoritmikuvauksissa esitetyllä tavalla.

3.4 Roolitason valvontafunktio

Roolitason valvontafunktion päätehtävä on olla tietoinen valvomansa roolin rooliepookista ja kommunikoida sessiotason ja tehtävämallitason kanssa rooliepookin tilassa tapahtuvista muutoksista. Roolitason valvontafunktio vastaanottaa tehtävämallitasolta ilmoituksia rooliepookin elinkaaren vaihesiirtymistä ja rooliepookkiin liittyvän koreografian suorituksessa tapahtuneista virhetilanteista.

Epookin vaihesiirtymiin liittyvät ilmoitukset toimitetaan suoraan sessiotasolle muihin organisaatioihin lähetettäviksi. Erityisesti rooliepookin päättymisestä tiedotetaan sessiotasoa. Rooliepookin päättyminen aiheuttaa sessiotason notifioinnin lisäksi roolin **deaktivoinnin**. Deaktivoinnin seurauksena tehtävämallitaso ja sitä kautta myöskään koreogra-

Algoritmi **Käsittele**(koreografi atason lähettämä virhetilanne V):

1. Eristä virhetilanteesta V session tunniste S ja roolin tunniste R
2. Pyydä roolitasolta session S roolin R aktiivisen rooliepookin tunniste E.
3. Tee rooliepookkiin E liittyvään tehtävämalliin M merkintä virheestä V ja virheen tapahtumisajasta.
4. Ilmoita virheestä roolitasolle ja tutki paluuarvona saatua ohjetta O,
   - Jos ohje O kertoo virhetilanteen johtaneen rooliepookin ennenaikaiseen loppumiseen, niin palaa operaatiosta paluuarvolla, joka pyytää koreografi atasoa invalidoimaan session S ja poistamaan välimuististaan siihen liittyvän metatiedon,
   - Jos komento K kertoo, ettei virhetilanne johda mihinkään toimenpiteisiin, niin palaa operaatiosta paluuarvolla, joka pyytää koreografi atasoa säilyttämään välimuistissaan olevan sessioon S liittyvän metatiedon.

Kuva 3.12: Tehtävämalitason algoritmi koreografiatasolla havaittujen virheiden käsittelyyn (geneerinen).

fiataso eivät voi jatkaa suoritustaan ennenkuin epookki on vaihtunut. Epookki vaihtuu kun kaikkien epookkiin liittyvän yhteisöarkkitehtuurin roolien rooliepookit ovat päättyneet. Rooliepookkitaso saa tiedon sessioepookin vaihtumisesta sessiotasolta ja tämän seurauksena vaihtaa aktiiviseen epookkiin liittyvät tehtävämallitason ja koreografiatason tilaautomaatit uuden epookin vastaaviksi. Tämän jälkeen roolitaso aktivoi tehtävämallitason ja sitä kautta epäsuorasti myöskin koreografiatason.

On syytää ottaa huomioon, että rooli ei välttämättä ole mukana kaikissa epookeissa. Tällöin se voi pysyä deaktivoituneena pitkäänkin. Edellisestä seuraa tarve pitää kaikkia rooliin liittyviä metatietoja yllä persistentisti tietokantaan talletettuna.

Tehtävämallitasolta saapuvien, epookin elinkaareen liittyvien, ilmoitusten lisäksi roolitaso voi saada vastaavia ilmoituksia sessiotasolta koskien jonkin muun organisaation rooliepookeissa tapahtuvia vaihesiirtymiä. Tällöin ei kyse ole valvonnasta vaan puhtaasta tilaseurannasta. Muiden organisaatioiden rooliepookkien tilaa seurattaessa roolitason ainoa funktio on toimittaa tieto roolin elinkaaren muutoksesta tehtävämallitasolle, joka ylläpitää rooliepookkiin liittyvää tehtävämallia.

Roolin elinkaaren muutosten lisäksi roolitaso voi saada tehtävämallitasolta virheilmoituksia koreografian suorituksessa tapahtuneista virheistä. Roolitasolle on myöskin mahdollista konfiguroida roolitason **politiikkoja**, joiden avulla roolitaso voi tehdä päätöksiä esimerkiksi eri virhetilanteiden merkityksestä. Tämän työn puitteissa virhetilanteiden evaluointia ja niihin reagointia ei kuitenkaan enempää käsitellä. Tämän hetken prototyyppitoteutuksessa roolitaso toimii optimistisesti ja ainoastaan toimittaa virhetilanteista tiedon sessiotasolle. Roolitaso voidaan vaihtoehtoisesti konfiguroida toimimaan pessimistisesti siten, että jokainen koreografiatason virhe aiheuttaa rooliepookin ennenaikaisen keskeyttämisen.

Valvonta- ja seurantatoimintojensa lisäksi roolitaso tarjoaa rajapinnan rooliin liittyvien tilatietojen eli käytännössä rooliepookkeihin liittyvien tehtävämallien pyytämiseksi.

3.5 Sessiotason valvontafunktio

Sessiotason valvontafunktion päätehtävänä on tietää, missä epookissa koko liiketoimintaverkosto kulloinkin on. Lisäksi sessiotason tulee tietää, mitkä aktiivisena olevan epookin yhteisöarkkitehtuurin rooleista kuuluvat sen oman organisaation hallintoalueelle ja mitkä muihin hallintoalueisiin.

Sessiotaso saa oman hallintoalueensa rooleja valvovalta roolitason valvontakerrokselta rooliepookin elinkaaren vaiheisiin liittyviä ilmoituksia. Rooliepookin päättymisilmoitusten perusteella sessiotaso päättelee milloin organisaation kaikki rooliepookit ovat loppuneet ja tiedottaa tästä sopimustasoa. Muut rooliepookin elinkaaren muutoksiin liittyvät ilmoitukset välitetään suoraan sopimustasolle reagoimatta niihin muuten mitenkään.

Kun sopimustaso on saanut tiedon kaikkien yhteisöarkkitehtuuriin liittyvien rooliepookkien loppumisesta (jokaiselta yhteisöarkkitehtuuriin liittyvältä organisaatiolta), saa sessiotaso pyynnön vaihtaa epookkia omassa organisaatiossaan. Sessiotaso siirtää uuteen epookkiin liittyvien roolien kohdalla kunkin roolin seuraavaan rooliepookkiin. Tämän seurauksena roolitaso, tehtävämallitaso ja koreografiataso jatkavat käytännössä suoritustaan uudessa epookissa käyttäen siihen liittyvää metatietoa.

Sessiotaso saa roolitasolta tiedon myös alemmilla valvonnan tasoilla havaituista virhetilanteista. Sessiotasolle on mahdollista konfiguroida politiikkoja, joiden perusteella sessiotaso voi tehdä päätöksiä virhetilanteiden seurauksista. Tämän hetken prototyyppitoteutuksessa sessiotaso kuitenkin vain toimittaa virheilmoitukset sopimustasolle ja jättää pohdinnan virhetilanteiden merkityksestä kokonaan sopimustasolle.

Seuraavassa luvussa (luku neljä) siirrytään tutkielman käytännönläheisempään osioon ja tehdään katsaus toteutusteknologiana käytettävään Web-palveluympäristöön. Viidennessä luvussa puolestaan kuvataan valvontajärjestelmän eri abstraktiotasot toteuttava toteutusarkkitehtuuri neljännessä luvussa esitellyn Web-palveluympäristön puitteissa.

Luku 4

Web-palveluarkkitehtuuri ja Web-palveluiden toteuttaminen

Web-Pilarcos projektin ja tämän tutkielman puitteissa valittuna teknologisena ympäristönä on Web-palveluarkkitehtuuri [34]. Teknisenä alustana toimii Web-palveluarkkitehtuurin suosittu Java-toteutus Apache Axis [41]. Toteutettavat Web-palvelut ovat joko itsenäisiä Apache Axis-alustaa hyödyntäviä Java-ohjelmia tai Apache Axis yhteensopivan Java 2 Enterprise Edition (J2EE) [29] sovelluspalvelimen mukaisia Enterprise Java Bean (EJB) komponentteja.

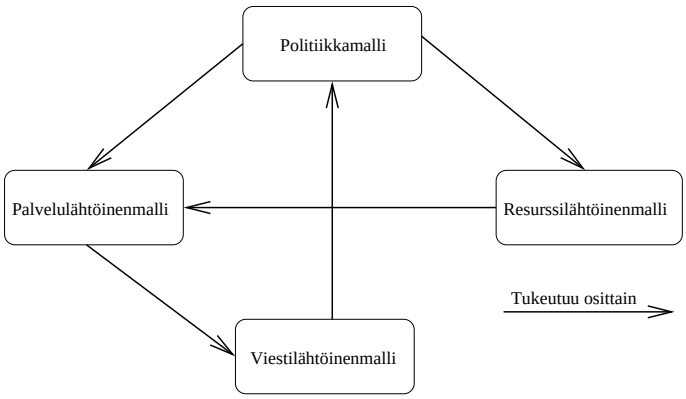
Tässä luvussa tutustutaan aluksi Web-palveluarkkitehtuurin tämän tutkielman kannalta keskeisimpiin osiin. Sen jälkeen luodaan katsaus Apache Axis-alustan asiakas- ja palvelinpään rajapinta-arkkitehtuuriin. Lopuksi luodaan katsaus Web-palvelun toteuttamisen periaatteisiin sekä itsenäisenä (standalone) Axis-palveluna että EJB-komponenttina. Pääkappaleen tarkoituksena on tutustuttaa lukija Web-palveluiden käsitemaailmaan ja valittujen alustojen toimintaperiaatteisiin riittävällä tarkkuudella seuraavan pääkappaleen kuvaaman valvonnan toteutusarkkitehtuurin ymmärtämiseksi.

4.1 Web-palveluarkkitehtuuri

Web-palveluarkkitehtuuri [34] määrittelee neliosaisen metamallin Web-palveluiden toteuttamiseen. Metamallin neljä osaa rakentuvat osittain toistensa varaan ja ovat nimeltään viestilähtöinen malli, palvelulähtöinen malli, resurssilähtöinen malli ja politiikkamalli.

Viestilähtöinen malli määrittelee viesteihin, viestien rakenteeseen ja viestitykseen liittyvät käsitteet. Palvelulähtöinen malli määrittelee varsinaisen Web-palvelun ja siihen läheisesti liittyvät muut käsitteet. Resurssilähtöinen malli kuvaa Web-palveluihin liittyvän maailmankuvan resurssinäkökulmasta. Politiikkamalli kuvaa Web-palveluihin ja niitä toteuttaviin toimijoihin liitettävien käyttäytymistä ohjaavien politiikkojen käsitteistön. Metamallin osat ja osien suhteet on esitetty kuvassa 4.1.

Tämän tutkielman puitteissa ei ole tarpeen tuntea koko Web-palveluihin liittyvää käsitteistöä. Kuva 4.2 esittääkin koko metamallin eri osista kootun tiivistetyn metamallin, joka kuvaa tutkielman kannalta olennaiset Web-palvelumaailman käsitteet ja niiden suh-

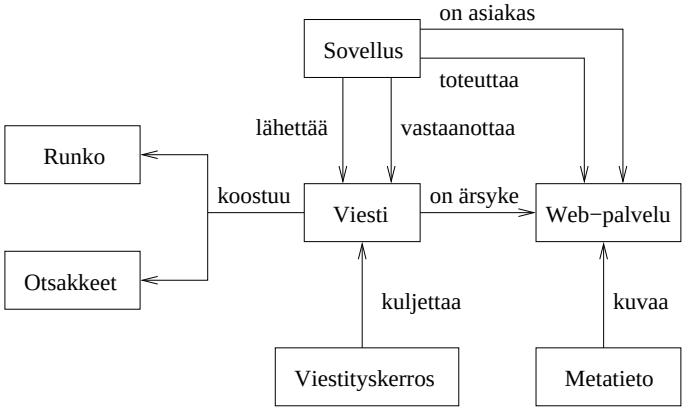


Kuva 4.1: Web-palveluarkkitehtuurin neliosaisen metamallin osat.

teet.

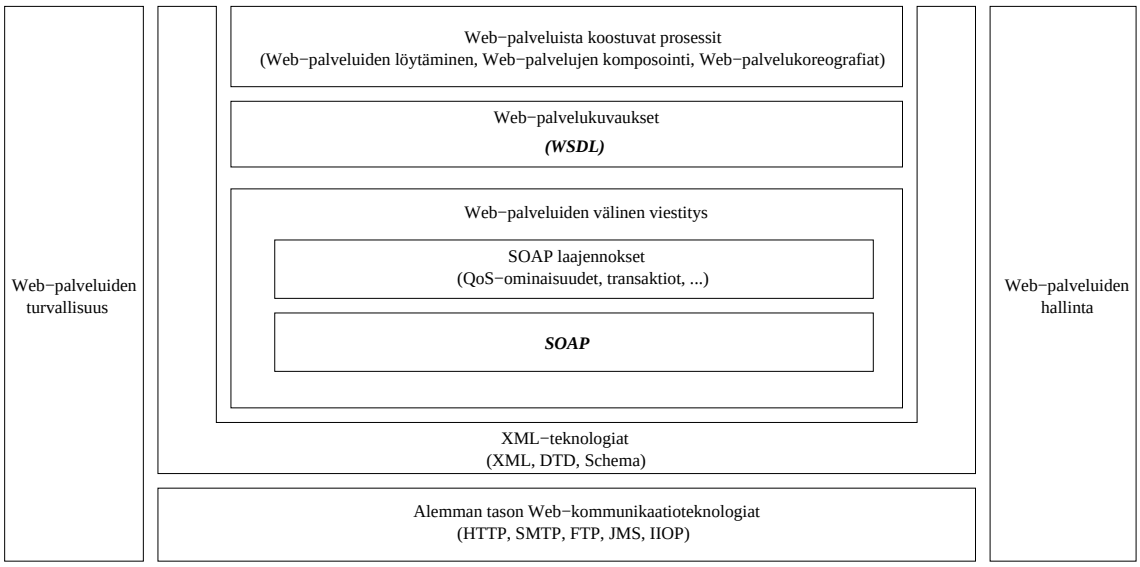
Web-palveluarkkitehtuuri määrittelee itse Web-palvelun seuraavasti: *"A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (spesifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."*.

Tämän tutkielman puitteissa Web-palveluista puhuttaessa tarkoitetaan ennenkaikkea sellaisia Web-palveluja, jotka on kuvattu WSDL kielellä [36] (**Metatieto** kuvassa 4.2), kommunikoivat suoraan SOAP-viesteillä [32] tai käyttäen hyväkseen SOAP-pohjaisia [32] etäproseduurikutsuja (**Viesti** ja **Viestityskerros** kuvassa 4.2) ja on toteutettu Javalla hyödyntäen Apache Axis Web-palvelu -alustaa [41] (**Sovellus** kuvassa 4.2).



Kuva 4.2: Web-palveluarkkitehtuurin keskeisiä käsitteitä.

Kuvassa 4.3 on esitetty Web-palveluarkkitehtuurin määrittelemä teknologiapino. Pinon pohjalla ovat yleiset Web-palveluiden kommunikoinnissa käytettävät protokollat. Pinon toisella tasolla on XML-teknologiaperhe [28], jonka varaan kolme ylintä tasoa raken-



Kuva 4.3: Web-palveluarkkitehtuurin määrittelemä teknologiapino [34].

tuvat. Pinon kolmannella tasolla on Web-palveluiden väliseen viestintään liittyvä SOAP-protokolla ja sen laajennokset. Pinon neljännellä tasolla on Web-palveluiden kuvaamiseen käytettävät teknologiat (ennenkaikkea WSDL). Pinon ylimmällä tasolla on Web-palveluista koostuvien (liiketoiminta)prosessien ja niiden vuorovaikutusten kuvaamiseen liittyvät teknologiat. Kuvan 4.3 reunoilla on koko teknologiapinon läpi ulottuvat Web-palveluiden turvallisuuteen ja hallintaan liittyvät teknologiat.

Teknologiapinon eri osiin liittyvien yksittäisten spesifikaatioiden kirjo on laaja ja yhä kasvava. Monikaan teknologia ei ole vielä saavuttanut vakiintunutta asemaa. Juuri tämän vuoksi kuvassa 4.3 esiintyvät vain ne teknologia-akronyymit, joiden voidaan katsoa vakiinnuttaneen asemansa “de-facto”-standardeiksi. Parhaiten vakiintuneet teknologiat ovat jo aiemmin mainitut SOAP ja WSDL.

SOAP kuvaa laajennettavissa olevan kehyksen (framework) XML-pohjaisten viestien rakenteen ja ominaisuuksien kuvaamiseen. SOAP-viesti rakentuu viestin **rungosta** (body), viestiin liittyvistä **otsakkeista** (headers) sekä optionaalisista **liitteistä** (attachments). Viesti kapseloidaan **sähköiseen kirjekuoreen** (envelope), jonka SOAP-viestejä ymmärtävä viestityskerros osaa toimittaa annettuun osoitteeseen.

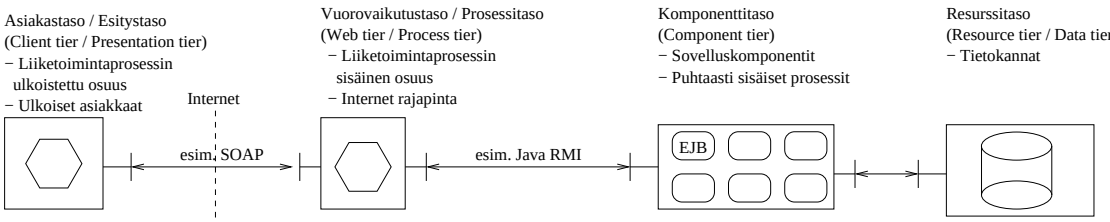
SOAP-protokollan (tai oikeammin SOAP-kehyksen) kehityshistoriallisista syistä johtuen SOAP-akronyymi ei enää esiinny varsinaisessa SOAP-spesifikaatiossa. Sen sijaan käytössä on kaksi akronyymista avattua versiota: “Service Oriented Architecture Protocol” ja “Simple Object Access Protocol”. Edellistä käytetään puhuttaessa SOAP:sta laajemmassa kontekstissa ja jälkimmäisellä viitataan SOAP:n yleisimpään käyttötapaan XML-pohjaisten, synkronisten tai asynkronisten etäproseduurikutsujen (XML-RPC) realisointiteknologiana (SOAP-RPC).

WSDL puolestaan on kieli, jolla voi kuvata Web-palvelun abstraktin rajapinnan ja sen sidonnan konkreettiseen viestityskerrokseen. WSDL-kuvaus sisältää ennen kaikkea määrittelyn Web-palvelun tarjoamista abstrakteista porteista (rajapinnoista), porttien sisältämistä operaatioista ja operaatioissa välitettävistä abstrakteista viestityypeistä. Lisäksi WSDL-kuvaus sisältää mahdollisuuden määritellä, miten abstraktit kuvaukset sidotaan konkreettiseen viestitysteknologiaan (esimerkiksi SOAP). WSDL kuvaus voi sisältää myös Web-palvelun nimen/tunnisteen ja verkko-osoitteen (URL).

4.2 Nelitasoiset yritysjärjestelmät ja J2EE

Nykyisin tuotettavista uusista sähköisen liiketoiminnan yrityssovelluksista (enterprise applications) jo yli puolet tuotetaan perustuen nelitasoiseen yritysjärjestelmäarkkitehtuuriin [14, 20]. Nelitasoisesta yritysjärjestelmäarkkitehtuurista on olemassa kaksi markkinoita dominoivaa kilpailevaa versiota. Toinen on Sun-yhtymän Java 2 Enterprise Edition (J2EE) ja toinen on Microsoftin .NET. Erään ennusteen mukaan vuoteen 2007 mennessä J2EE ja .NET pohjaisten toteutusten osuus uusista yrityssovelluksista tulee nousemaan jo yli 80 prosentin [14, 20].

Nelitasoinen yritysjärjestelmä jonka rakenne on visualisoitu kuvassa 4.4 jakaantuu neljään erilliseen horisontaaliseen tasoon. Ensimmäinen taso, jota kutsutaan asiakastasoksi tai esitystasoksi (client / presentation tier), kuvaa liitynnän yrityksen sisäiseen tietojärjestelmään yrityksen ulkopuolella olevan tahon näkökulmasta. Taso voi olla toteutettu vaikkapa Web-selaimeen ladattavana HTML-sivuna tai organisaation ulkopuolella toimivana erillisenä sovelluksena. Jos esitystasolla automaattisen sovellusohjelman sijaan toimii ihminen on kyseessä ihmisen ja koneen välinen vuorovaikutus (human-to-machine). Muutoin puhutaan koneiden välisestä vuorovaikutuksesta (machine-to-machine). Tämän tutkielman puitteissa kiinnostavaa on ennenkaikkea sovellusten välillä tapahtuva erilliseen sopimukseen pohjautuva viestinvaihto, eikä ihmisen ja koneen välisen viestinvaihdon erityispiirteisiin jatkossa puututa.



Kuva 4.4: Nelitasoisen yritysjärjestelmän horisontaalirakenne.

Toinen horisontaalitaso, jota kutsutaan vuorovaikutustasoksi tai joskus prosessitasoksi (Web tier / Process tier), määrittelee yrityksen ulkoisen liittymän yrityksen itsensä näkökulmasta. Ulkoinen liittymä voi teknisesti olla esimerkiksi Web-palvelu. Vuorovaikutustaso toimii paitsi rajapintana organisaatiosta ulospäin myös rajapintana organisaation sisäisiin järjestelmiin. Vuorovaikutustaso toteuttaa siis ulkoisesti näkyvän prosessin ohella myös prosessin, jolla yhdistetään ulospäin näkyvä viestityskoreografia ja sisäisten jär-

jestelmien kommunikointiin liittyvä viestityskoreografia. Ulkoisen ja sisäisen toiminnan yhdistävä prosessi voidaan toteuttaa erillisenä sovelluksena (esimerkiksi J2EE maailmassa Java Servlet [31] tyyppisenä toteutuksena) tai jo aiemmin mainittujen tarkoitukseen erityisesti suunniteltujen prosessikielten ja -alustojen [22, 23, 26, 37, 40, 44] avulla.

Kolmas horisontaalitaso on nimeltään komponenttitaso (component tier). Komponenttitasolla sijaitsevat yrityksen sisäiset sovelluskomponentit ja niihin liittyvät sisäiset prosessit. Esimerkiksi J2EE-arkkitehtuurissa tällä tasolla sijaitsevat EJB-komponentit (Enterprise Java Beans) [27]. Sisäiset prosessit on usein upotettu osaksi komponenttien sovelluslogiikkaa. Jos komponenttitaso ja vuorovaikutustaso on fyysisesti hajautettu ne tyypillisesti kommunikoivat jollain olio-orientoituneella tiukasti sitovalla (tightly coupling) etäkommunikointiteknologialla (kuten J2EE-arkkitehtuurin tapauksessa JavaRMI [30] tai CORBA/IIOP [24]).

Komponenttitasolla on omia viestinvaihdon valvonnasta erillään olevia valvontatarpeita muun muassa sisäisten sovellusten toimintaa ohjaavien politiikkojen valvomiseen liittyen. Näiden valvontatarpeiden toteuttamiseen vaadittavat mekanismit ovat luonteeltaan sellaisia, että ne liittyvät kiinteästi komponenttien sisäiseen toimintalogiikkaan. Valvontamekanismit voidaan standardilla tavalla toteuttaa esimerkiksi rakentamalla ne osaksi komponenttien toteuttamiseen liittyvää sovelluskehikkoa, josta komponentit voisivat ne periä. Tämänkaltaiset valvontamekanismit ovat tämän tutkielman tavoitteen asettelun ulkopuolella.

Neljäs horisontaalitaso on nimeltään resurssitaso (resource tier / data tier). Resurssitasolla sijaitsevat esimerkiksi tietokannat. Resurssitasolla on omia resurssien (esimerkiksi tietokantojen) käyttöön liittyviä valvontatarpeita. Nämä valvontatarpeet voidaan osittain ottaa huomioon jo komponenttitason valvonnassa ja ovat myös tämän tutkielman tavoitteen asettelun ulkopuolella.

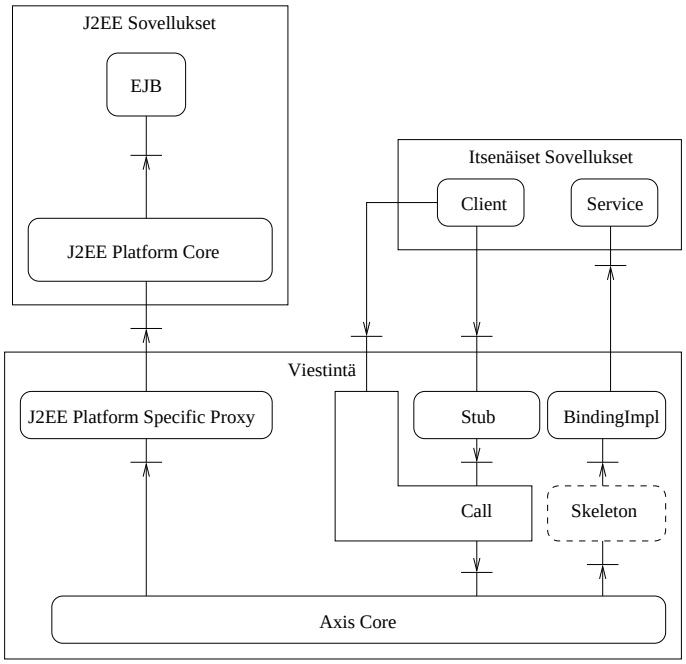
Komponenttitaso ja resurssitaso voivat liittyä toisiinsa monilla vaihtoehtoisilla tavoilla. Ne voivat olla fyysisesti erillään toisistaan tai niitä voidaan suorittaa samassa koneessa. Komponenttitason komponentit on usein eristetty resurssitasosta siten, että niitä suoritusaikana hallinnoiva sovelluspalvelin hoitaa resurssitason kanssa kommunikoinnin komponenteille tuntumattomasti.

4.3 Apache Axis Web-palvelualustana

Apache Axis [41] on avoimeen lähdekoodiin perustuva Java-pohjainen SOAP-toteutus. Axis mahdollistaa monenlaisten Web-palveluiden ja Web-palveluasiakkaiden toteuttamisen Java-ohjelmointikielellä. Apache Axis on jatko aiempaan Apache SOAP-projektiin, joka puolestaan perustuu IBM:n alkuperäisen SOAP4J-toteutuksen avattuun lähdekoodiin.

Apache Axis-alustan rakenne voidaan sovelluksen näkökulmasta jakaa varsinaiseen ytimeen, joka toteuttaa SOAP-yhteensopivan viestityskerroksen sekä asiakas- ja palvelinpään sovellusrajapintoihin. Axis-ytimen ominaisuuksiin ei jatkossa puututa, vaan keskitytään sovellusrajapintojen (tutkielman kannalta) olennaisiin osiin.

Axis- ja Axis/J2EE-alustojen yleisarkkitehtuuri sovellusten näkökulmasta tarkastelu-



Kuva 4.5: Axis ja Axis/J2EE alustojen yleisarkkitehtuuri sovellusten näkökulmasta.

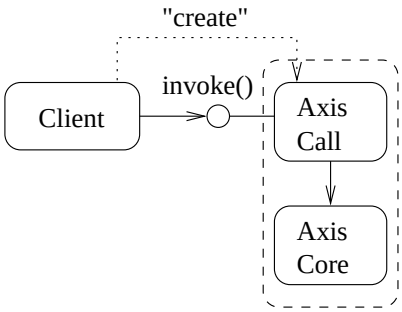
na on esitetty kuvassa 4.5. Seuraavissa alikappaleissa käydään läpi arkkitehtuurin eri osien rooli sovellusskenaarioissa.

4.3.1 Apache Axis -asiakasrajapinta

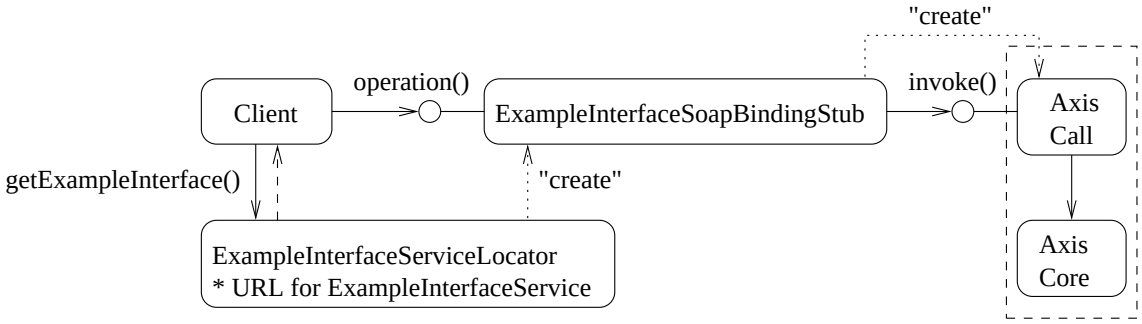
Axis-alustan geneerisenä asiakasrajapintana toimii **Call**-luokka. Kyseinen luokka tarjoaa Axis-asiakkaalle mahdollisuuden rakentaa dynaamisesti Java-pohjainen SOAP-viestiä esittävä olio ja lähettää se. Halutessaan lähettää SOAP-viestin Axis-asiakas luo Call-olion ja määrittelee olion tilan käyttäen olion tarjoamia operaatioita. Tämän jälkeen asiakas voi lähettää luomansa SOAP-viestin **Call.invoke()** operaatiota kutsumalla. Operaatiosta on olemassa synkroninen ja asynkroninen versio. Call-olio käyttää Axis-ytimen tarjoamia palveluja varsinaisen XML-pohjaisen SOAP-viestin muodostamiseen. Kuvassa 4.6 on esitetty dynaamista kutsurajapintaa käyttävä asiakas.

Koska geneerisen asiakasrajapinnan käyttäminen on usein ohjelmoijan kannalta työlästä, tarjoaa Axis-mahdollisuuden myös staattisen Web-palvelukohtaisen kutsurajapinnan käyttöön. Käytettäessä staattista kutsurajapintaa asiakkaalla on sovellusta laadittaessa oltava käytössään kutsutun Web-palvelun WSDL-kuvaus. WSDL-kuvauksen pohjalta generoidaan Web-palvelua asiakaspäässä edustava staattinen tynkätoteutus (stub), joka tarjoaa samat operaatiot kuin varsinainen Web-palvelukin. Tynkätoteutusta käytettäessä asiakas käyttää geneerisen rajapinnan sijasta tynkätoteutuksen rajapintaa. Tynkätoteutus puolestaan käyttää Call-olion rajapintaa ja osaa suoritusaikana luoda asiakkaan palvelupyyntöjä vastaavia SOAP-kutsuja Call-oliota käyttäen.

Tynkätoteutuksen generoinnin yhteydessä tuotetaan myös muun muassa sen elinkaa-



Kuva 4.6: Dynaamista kutsurajapintaa käyttävä Axis-asiakas.



Kuva 4.7: Staattista, tynkätoteutus-pohjaista, kutsurajapintaa käyttävä Axis-asiakas.

ren hallintaan ja dynaamiseen sidontaan tarkoitettu apuluokka **ServiceLocator**. Halutessaan tehdä Web-palvelukutsun asiakas pyytää ServiceLocator-luokalta tarvitsemaansa tynkätoteutusta. ServiceLocator sisältää WSDL-kuvauksesta poimitun Web-palvelun URL-osoitteen (Universal Resource Locator) ja osaa pyydettäessä luoda kyseistä Web-palvelua vastaavan ja URL-osoitteeseen sidotun tynkätoteutuksen. Asiakas voi halutessaan suoritusaikana uudelleen konfiguroida ServiceLocator-luokan sisältämän osoitteen.

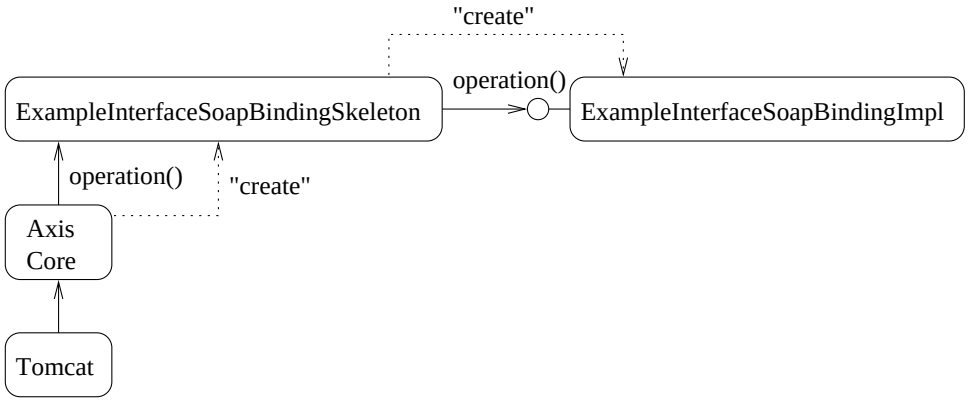
Kuvassa 4.7 on esitetty staattista kutsurajapintaa käyttävään Axis-asiakkaseen liittyvät vuorovaikutukset **ExampleInterface** -rajapinnan tarjoavaa Web-palvelua kutsuttaessa.

4.3.2 Apache Axis -palvelurajapinta

Palvelinpäässä toimiva Apache Axis on toteutettu Java-Servlet (käytetään myös termiä Java-palvelma) toteutuksena [31]. Java-pohjaisista, Web-ympäristöön tarkoitetuista, palvelmapalvelimista (servlet server) tunnetuin lienee Apache Tomcat [42]. Tomcat onkin yleisin Axis-toteutuksen toimintaympäristö. Palvelmapalvelimen tehtävänä on vastaanottaa esimerkiksi HTTP-protokollan välityksellä tehty WWW-kutsu ja reitittää sen sisältö oikealle palvelmalle (servlet). Axis-palvelman tehtävä on löytää viestille sopiva vastaanottava sovellus (Web-palvelun toteutus) ja muokata vastaanotettu SOAP-viesti sovelluksen ymmärtämään Java-pohjaiseen muotoon.

Jotta Web-palveluiden toteutukset voidaan suoritusaikana löytää, täytyy ne ensin **re-**

kisteröidä Axis-palvelmalle. Tähän rekisteröintiin (deployment) Axis tarjoaa yksinkertaisen kuvauskielen (deployment descriptor) sekä valmiita apuluokkia. Sovellusten rekisteröintiin on kaksi eri mahdollista käyttötapaa. Yleisimmin käytetty palvelinpään tynkätoteutuksia eli **Skeleton**-toteutuksia hyödyntävä lähestymistapa sopii hyvin monille Web-palveluille. Palvelinpään tynkätoteutus generoidaan jokaiselle Web-palvelulle erikseen Axisin tarjoamilla välineillä ja rekisteröidään Axis-palvelmalle kyseisen Web-palvelun toteuttajaksi. Palvelinpään tynkätoteutus sisältää esimerkiksi säännöt SOAP-viestissä esitettyjen tietotyyppien muuntamiseksi toteutuksen käyttämiksi tietotyypeiksi. Skeleton-toteutusta käytettäessä rekisteröintiin tarvittavat kuvaajat muodostuvat hyvin yksinkertaisiksi.

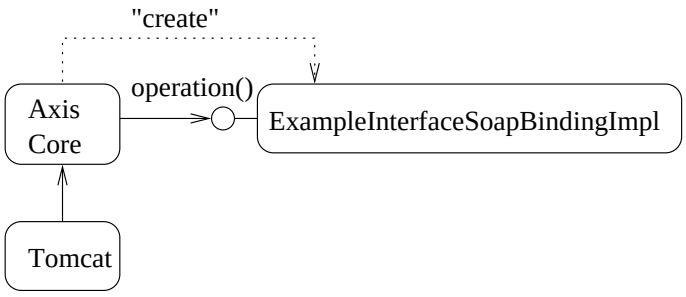


Kuva 4.8: Palvelinpään tynkätoteutusta käyttävä Axis-palvelu.

SOAP-viestin saapuessa suoritusaikana palvelinpäähän Axis luo kutsuttua Web-palvelua edustavan palvelinpään tynkätoteutuksen. Tämän jälkeen Axis kutsuu kutsuu SOAP-viestiä vastaavaa palveluoperaatiota viestistä poimituin parametrein. Tynkätoteutus edelleen luo erityisen aiemmin automaattisesti tuotetun sidontaolion (**BindingImpl**). Sidontaolio on tuotettu samassa yhteydessä kuin tynkätoteutuskin. Web-palvelun toteuttaja voi toteuttaa Web-palvelun yksinkertaisesti täyttämällä sidontaoliossa olevat tyhjät operaatiot sopivalla toiminnallisuudella. Vaihtoehtoisesti Web-palvelu voi olla toteutettu kokonaan erillään sidontaoliosta jolloin sidontaolio toimii vain varsinaisen Web-palvelun välittimenä (proxy). Jälkimmäisessä tapauksessa ohjelmoija täydentää sidontaolioon välitintoiminnallisuuden, joka kutsuu varsinaista Web-palvelutoteutusta. Palvelinpään tynkätoteuksen toimintaan liittyvät vuorovaikutukset on esitetty kuvassa 4.8.

Vaihtoehtoinen tapa palvelinpään tynkätoteutuksen käytölle on rekisteröidä sidontaolio (tai Web-palvelun erillinen Java-toteutus) suoraan Axis-palvelmalle. Tällöin kaikki normaalisti palvelinpään tynkätoteutuksessa olevat muuntosäännöstöt joudutaan kuvaamaan rekisteröitymiseen käytettävässä kuvaajassa. Kuvaajasta voi tällöin tulla hyvinkin monimutkainen ja pitkä.

Tätä palvelinpään geneerisempää liittymää on hyvä käyttää esimerkiksi silloin, kun Web-palvelu on sellainen jota muutetaan usein (eikä tynkätoteutuksia haluta jatkuvasti uudelleen tuottaa ja linkittää toteutuksiin). Muuttunut Web-palvelu voidaan tällöin saattaa toimivaksi pelkästää rekisteröintikuvaajaa muuttamalla. Ilman tynkätoteutusta toimi-

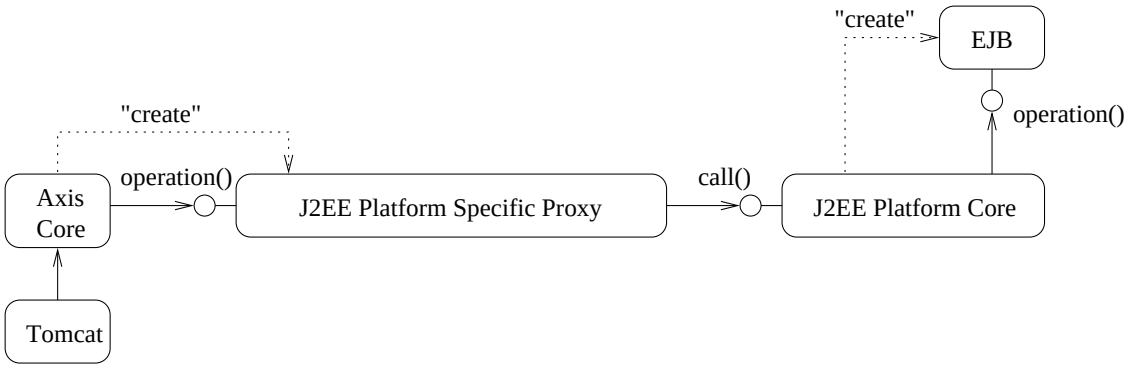


Kuva 4.9: Ilman palvelinpään tynkätoteutusta toimiva Axis-palvelu.

vaa palvelinpään liittymää käytetään usein myös silloin jos Axis-sovelmaa ei käytetä itsenäisesti, vaan se on integroitu osaksi laajempaa sovelluspalvelinympäristöä. Näin on usein esimerkiksi J2EE-yhteensopivien palvelinten tapauksessa. Niissä Axis ei suoraan näy lainkaan sovelluskehittäjille, vaan sitä ohjataan J2EE-alustan omien kuvaajien kautta. Kuvassa 4.9 on esitetty ilman palvelinpään tynkätoteutusta toimivan Web-palvelun vuorovaikutukset Axis-ympäristön kanssa.

4.3.3 Apache Axis-palvelun toteuttaminen EJB-komponenttina

J2EE-yhteensopivissa sovelluspalvelimissa toimivat Web-palvelut toteutetaan yleensä EJB-komponentteina. EJB-komponentit tarjoavat useita valmiita kehikkoja erilaisten palvelusovellusten toteuttamiseen. EJB-spesifikaatio jakaa sovellukset tilattomiin (stateless session bean, message driven bean), väliaikaista tilaa ylläpitäviin (stateful session bean) ja pysyvää tilaa ylläpitäviin (entity bean) sovelluksiin. EJB-spesifikaation mukaisia sovelluksia voidaan suomeksi kutsua yrityspavuiksi. Sovelluksen laatija päättää minkälaisen kehikon mukaisen pavun hän tarvitsee ja tämän jälkeen täydentää valitsemaansa kehikkoon varsinaisten palveluoperaatioden toteutukset.



Kuva 4.10: J2EE-alustan toteuttama Axis-palvelu.

Kuvassa 4.10 on periaatteellinen esimerkki Axis/J2EE-alustalle EJB-komponenttina toteutetusta Web-palvelusta. Esimerkki ei kuvaa missään tietyssä J2EE-alustassa toteutet-

tua arkkitehtuuria, vaan antaa esimerkin yhdestä suunnittelumallista. Suunnittelumallissa J2EE-alusta rekisteröi Axis-sovelmalle välitin-olion jokaista tukemaansa Web-palvelua kohden. Välitin-olio edelleen toimittaa saamansa Web-palvelukutsut J2EE-alustalle alustakohtaista mekanismia käyttäen. J2EE-alusta käsittelee Web-palvelukutsun ja reitittää kutsun oikealle pavulle.

Seuraavassa pääkappaleessa käsitellään tarkemmin valvontafunktion eri kerrosten toteuttamista puhtaasti Axis-pohjaisissa sekä Axis/J2EE-ympäristöissä.

Luku 5

Automaattisen valvontafunktion toteuttaminen Web-palveluympäristössä

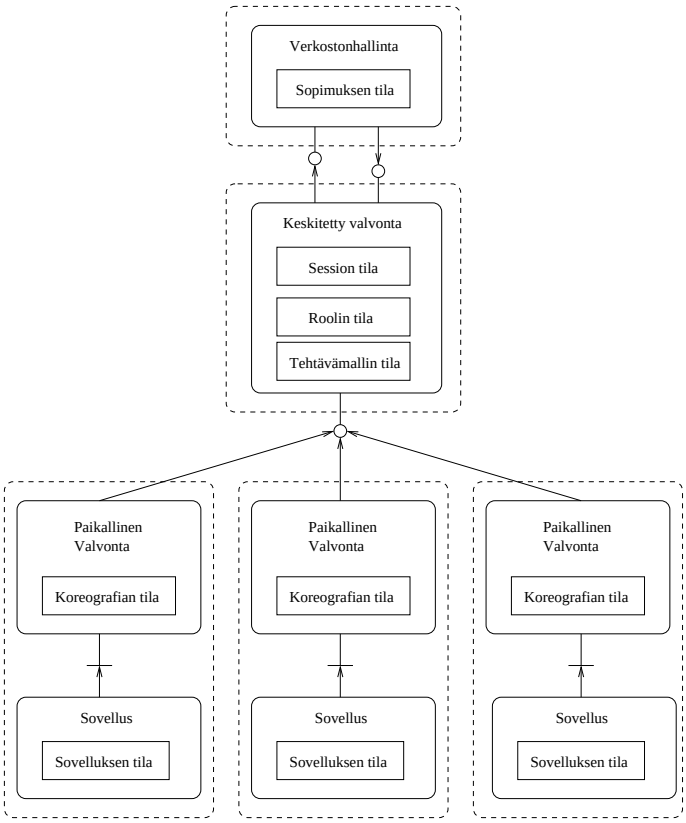
Tässä luvussa esitetään valvonta-arkkitehtuurille Web-palveluympäristöön sopiva Java-pohjainen toteutusarkkitehtuuri. Asiakaspään tapauksessa esitetään malli valvotun Web-palveluasiakkaan toteuttamiseksi Apache Axis-alustan päälle. Palvelinpäässä esitetään erikseen kaksi toteutusmallia: valvotun Web-palvelun toteuttaminen itsenäisenä (standalone) Axis-palveluna sekä EJB-komponenttina.

5.1 Valvonnan tilatiedon ja valvontakerrosten fyysinen hajautus

Valvontakerrokset ja niiden ylläpitämä tila voidaan periaatteessa jakaa neljään erilliseen **toteutusyksikköön** kuten kuvassa 5.1 on esitetty. Toteutusyksiköistä korkeimmalla tasolla on **verkostonhallinta**, joka pitää yllä ja valvoo sopimuksen tilaa. Seuraavalla tasolla alaspäin mentäessä on **keskitetty valvonta**, joka sisältää sessiotason, roolitason ja tehtävämallitason valvontalogiikan ja tilan.

Kuvassa 5.1 on myös esitetty toteutusyksiköiden jakaminen **hajautusyksiköihin**. Hajautusyksiköitä on kolme kappaletta: verkostonhallinta omassa yksikössään, keskitetty hallinta omassa yksikössään ja paikallinen valvonta sekä sovellustason valvonta omassa yksikössään. Käytännössä hajautusyksiköt voivat olla fyysisesti sijoittuneet siten, että verkostonhallinta ja keskitetty valvonta sijaitsevat samassa koneessa.

Hajautusyksikkö ei tässä yhteydessä näin ollen kuvaa niinkään toteutusyksiköiden fyysistä sijoittumista samaan tai eri koneeseen vaan **potentiaalia** siihen. Toteutusyksikkö voidaan ulkopuolelta nähdä “mustana laatikkona”, jolla on **rajapinta**. Hajautusyksikkö puolestaan nähdään “mustana laatikkona”, jolla on **etärajapinta**. Rajapinnan ja etärajapinnan ero on siinä, että rajapintaa voi käyttää vain sen tarjoajan kanssa samassa (virtuaali)koneessa sijaitseva toimija. Etärajapintaa sen sijaan on mahdollista kutsua sekä samassa että eri (virtuaali)koneessa sijaitseva toimija.



Kuva 5.1: Liiketoimintaverkoston tilatiedon fyysinen hajautus ja rajapinnat.

Kaksi toteutusyksikköä siis kommunikoivat käyttäen paikallista rajapintaa (esimerkiksi Java-kutsu), mutta kaksi hajautusyksikköä kommunikoivat käyttäen etärajapintaa (esimerkiksi Web-palvelu-, JavaRMI-, tai CORBA/IIOP-kutsu). Hajautusyksiköt on mahdollista sijoittaa fyysisesti samaan koneeseen, mutta tällöinkin ne tukeutuvat etärajapintaan kommunikoidessaan. Hajautusyksiköt eivät siis koskaan ole suoraan tietoisia esimerkiksi muiden hajautusyksiköiden Java-rajapinnasta, vaan näkevät toisensa esimerkiksi WSDL- tai IDL-kuvauksen toteuttavana “mustana laatikkona”.

Paikallisen ja keskitetyn valvonnan erottaminen erillisiksi toteutus- ja hajautusyksiköiksi perustuu näiden toiminnallisuuksien ja ympäröivän järjestelmän suhteeseen. Paikallinen valvontakerros sisältää rajapinnan sovellusten sekä sovellusten käyttämän viestintäkerroksen kanssa ja näin ollen sen on luonnollista sijaita samassa hajautusyksikössä valvomansa sovelluksen kanssa.

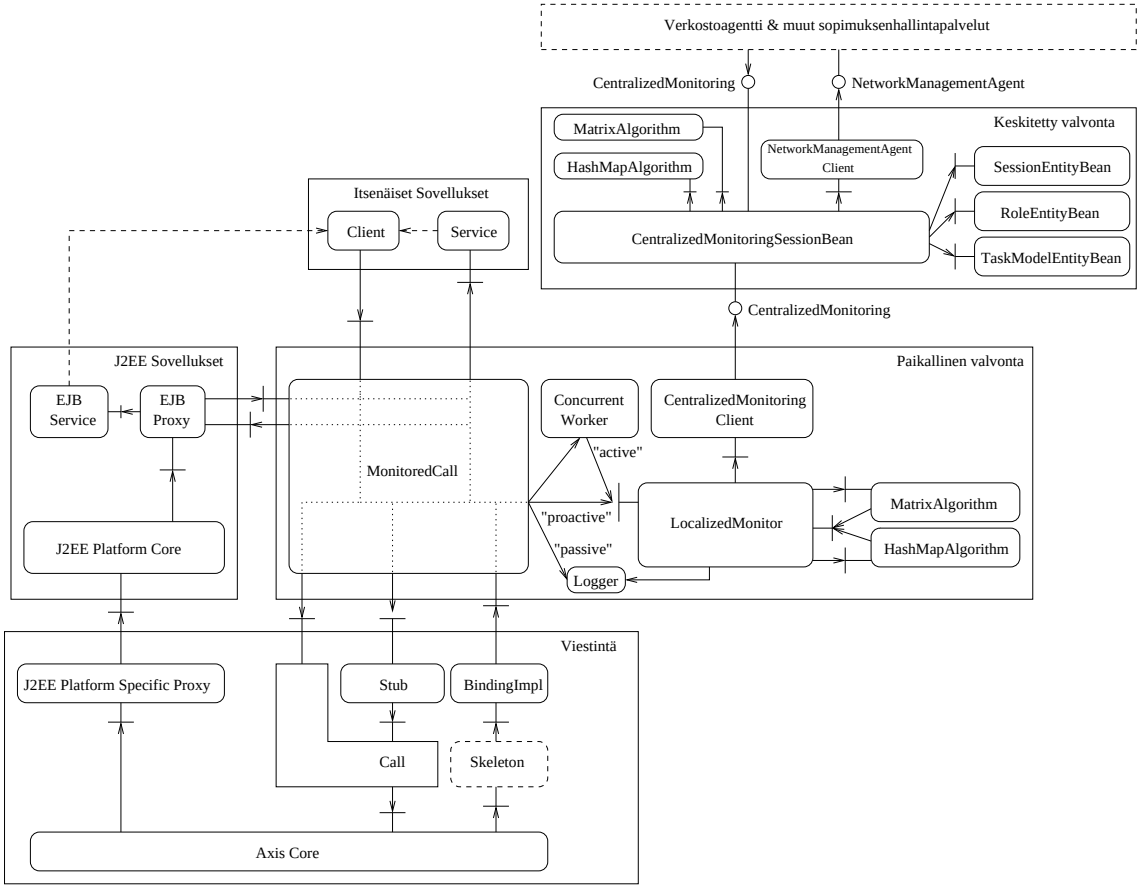
Toinen syy paikallisen valvonnan olemassaoloon on suorituskyky. Paikallinen valvontayksikkö sisältää suorituskyvyn kannalta vaativan koreografiatason valvontakerroksen. Koreografiataso kommunikoi usein sekä sovellusten että näiden käyttämän viestikerroksen kanssa. Koreografiatason ja tehtävämallitason keskinäinen kommunikaatio sen sijaan on vähäisempää. Näin ollen luonnollinen hajautuspiste muodostuu koreografiatason ja tehtävämallitason välille.

Koreografiatasolla on myös mahdollisuus ylläpitää omaa välimuistia, jonne session

aktiivisiin rooliepookkeihin liittyvät koreografiat voidaan tallentaa. Näin keskitetyn ja paikallisen valvonnan hajautusyksikköjen välinen liikenne saadaan minimoitua. Paikallinen ja keskitetty valvonta vuorovaikuttavat keskenään ainoastaan, kun tehtävävaikutuksia tai virheilmoituksia välitetään koreografiatasolta tehtävämallitasolle.

5.2 Valvonnan toteutuksen yleisarkkitehtuuri

Kuvassa 5.2 on esitetty olennaiset osat valvonnan toteuksen yleisarkkitehtuurista. Kuvassa valvontaan välittömästi tai välillisesti liittyvät järjestelmän osat on jaettu kuuteen eri arkkitehturaaliseen komponenttiin.



Kuva 5.2: Valvonta-arkkitehtuuri: toteutusluokat ja etärajapinnat.

Viestintä sisältää käytetyn viestintäkerroksen arkkitehturaaliset osat. Viestinnässä käytetään aiemmin esiteltyä Apache Axis-alustaa, joka tarjoaa dynaamisen ja staattisen kutsurajapinnan Web-palveluasiakkaille. Lisäksi Axis tarjoaa mahdollisuuden tehdä tynkätoteutuksen kanssa tai ilman sitä toimivia Web-palveluja.

Paikallinen valvonta sisältää valvonnan sovellus- ja viestintäkerrosläheiset osat. Paikallinen valvontakerros liittyy sekä sovelluksiin että viestintäkerrokseen paikallisen (Java-)rajapinnan kautta. Lisäksi paikallinen valvontakerros liittyy keskitettyyn valvontakerrokseen Web-palvelurajapinnan kautta. Paikallinen valvonta muodostaa käytännössä Java-luokkakirjaston, joka sijoittuu sovellus- ja viestintäkerroksen väliin. Paikallisen valvonnan välimuistitoiminnallisuuden ollessa pois käytöstä kerros toimii tilattoman funktiokirjaston tavoin. Välimuistinsa ollessa käytössä paikallinen valvonta ylläpitää lokaalia tilaa tarjoamiensa kirjasto-operaatioiden lisäksi. Paikallinen valvonta toteuttaa koreografiatason valvontakerroksen.

Keskitetty valvonta sisältää toteutuksen tehtävämallitason, roolitason ja sessiotason valvontakerroksille. Keskitetty valvonta tarjoaa Web-palvelurajapinnan sekä verkostonhallintapalveluille että paikalliselle valvontakerrokselle. Keskitetty valvonta on toteutettu J2EE-spesifikaation mukaisina EJB-papuina siten, että kerroksen tilaa ylläpitävät osat on toteutettu tietokantaan tilansa tallettavina yksilöpapuina (entity bean) ja muut osat tilattomana sessiopapuna (stateless session bean). Pavut tukeutuvat toteutuksessaan useaan tavalliseen Java-luokkaan.

Verkostoagentti ja muut sopimuksenhallintapalvelut sisältää yleiset verkostonhallintaan liittyvät palvelut sekä sopimustason valvontakerroksen toteutuksen. Verkostoagentti liittyy keskitettyyn valvontaan kaksisuuntaisen Web-palvelurajapinnan kautta. Tämä kerros on toteutettu erillisenä Web-Pilarcos projektissa, eikä sen sisäiseen arkkitehtuuriin enempää puututa.

J2EE-sovellukset edustavat JBoss-nimiselle [43] J2EE-sovelluspalvelimelle toteutettuja sovelluksia. Sovellukset on tällöin toteutettu EJB-spesifikaation mukaisina papuina. EJB-sovellus sisältää ensisijaisesti johonkin Web-palveluun liittyvän liiketoimintalogiikan toteutuksen. EJB-sovellus voi toimia myös asiakkaana, jos sen liiketoimintalogiikka on riippuvainen muista sovelluksista tai Web-palveluista. EJB-sovellus ei kuitenkaan koskaan ole ensisijaisesti asiakas, vaan sen mahdolliset asiakasominaisuudet aktivoituvat vasta sen palvelulogiikan suorituksen aikana.

Itsenäiset sovellukset edustavat itsenäisiä (standalone) Axis-alustan päälle toteutettuja Axis-pohjaisia Web-palveluita tai Web-palveluasiakkaita. Itsenäisellä tarkoitetaan sitä, että sovellus kykenee toimimaan ilman ulkopuolista sovelluspalvelinta tai vastaavaa suoritusympäristöä. Esimerkiksi Java-kielen tapauksessa käytetään usein englanninkielistä termiä **standalone Java application** itsenäisistä Java-ohjelmista. Esimerkkejä epäitsenäisistä Java-ohjelmista ovat Java-sovelmat (Java applet), Java-palvelmat (Java servlet), tai aiemmin mainitut Java yrityspavut (Enterprise Java Beans). Itsenäinen Axis-asiakas voi kuitenkin olla osa epäitsenäistä Java-ohjelmaa. Esimerkiksi yrityspapuna eli EJB-sovelluksena toteutettu Java-luokka voi käyttää itsenäistä Axis-asiakasta Web-palvelukutsujensa tekemiseen.

Seuraavissa kappaleissa tutustutaan tarkemmin paikallisen ja keskitetyn valvontakerroksen sisäiseen arkkitehtuuriin ja toimintaan. Paikallisen valvontakerroksen tapauksessa käydään lisäksi läpi paikallisen valvonnan tarjoama sovellusrajapinta (API), rajapinta Axis-pohjaisen viestintäkerroksen kanssa sekä rajapinta keskitetyn valvontakerroksen kanssa. Keskitetyn valvonnan tapauksessa luodaan katsaus keskitetyn valvonnan sisäiseen toteutusarkkitehtuuriin ja toimintaan sekä määritellään keskitetyn valvonnan ja verkostonhallintapalveluiden välinen kaksisuuntainen Web-palvelurajapinta.

5.3 Paikallisen valvontakerroksen rakenne, rajapinnat ja toiminta

Kuvan 5.2 keskellä on näkyvissä paikallisen valvontakerroksen tärkeimmät sisäiset osat. Olennaisimpia kuvassa näkyvistä sisäisistä komponenteista ovat **MonitoredCall**- ja **LocalizedMonitor**-luokat. **MonitoredCall** toteuttaa paikallisen valvontakerroksen rajapinnan sekä sovellusten että viestintäkerroksen kanssa. **LocalizedMonitor** puolestaan koordinoi valvontakerroksen toimintaa.

Kuvassa 5.2 paikallisessa valvontakerroksessa on näkyvillä myös muutamia apuluokkia (**ConcurrentWorker**, **CentralizedMonitoringClient**, **Logger**, **MatrixAlgorithm**, **HashMapAlgorithm**), jotka suorittavat kukin omaa hyvin määriteltyä tehtäväänsä ja ovat olennaisia paikallisen valvontakerroksen toiminnan ymmärtämiseksi. Muut vähemmän tärkeät apuluokat on redusoitu kuvasta pois.

Seuraavaksi luodaan katsaus paikallisen valvontakerroksen sisäisten komponenttien toimintaan.

5.3.1 Paikallisen valvontakerroksen perusrakenne

Paikallisen valvontakerroksen ytimen muodostaa **LocalizedMonitor**-luokka. Kyseinen luokka sisältää paikallisen valvontakerroksen toimintaa koordinoivan toiminnallisuuden. **LocalizedMonitor** muodostaa ympärilleen tähtimäisen rakenteen siten, että kaikki muut kerroksen luokat ovat yhteydessä siihen. Yleinen periaate on, että kaikki muut valvontakerroksen toimijat kommunikoivat ainoastaan **LocalizedMonitor**-luokan kanssa, joka koordinoi näiden välistä kommunikointia. Tähän sääntöön on kuitenkin poikkeus, josta puhutaan lisää myöhemmin.

Paikallisella valvontakerroksella on periaatteessa kolme rajapintaa joka ovat sovellus- ja viestintäkerrosrajapinta sekä keskitetyn valvontakerroksen rajapinta. **MonitoredCall**-luokka toteuttaa rajapinnat sovellusten ja viestintäkerroksen kanssa. Sovellusrajapinta (eli API) koostuu paikallisista staattisista Java-kutsuista. **MonitoredCall**-luokasta ei luoda lainkaan instansseja, vaan luokan operaatioita käytetään kuten perinteisen funktiokirjaston operaatioita.

5.3.2 Paikallisen valvontakerroksen asiakaspuolen sovellusrajapinta

Ilman valvontaa toimivassa järjestelmässä asiakassovellus kytkeytyy suoraan viestintäkerrokseen käyttäen sen tarjoamaa kutsurajapintaa. Kuten edellisessä luvussa esitettiin, Axis-pohjainen viestintäkerros tarjoaa kaksi erilaista liityntää: dynaamisen kutsurajapinnan ja staattisen kutsurajapinnan. Tämä periaate pätee useimmissa muissakin olemassa olevissa viestintäkerroksissa.

Sekä dynaamisen että staattisen kutsurajapinnan tapauksessa asiakkaan kannalta perustoiminnot ovat samat. Asiakas luo paikallisen instanssin edustamaan kutsuttavaa palvelua ja suorittaa etäkutsun käyttäen tätä instanssia. Sovellusrajapinnan toteuttavan **MonitoredCall**-luokan perusajatus on tukeutua tämän yleisen käytännön varaan.

Kun asiakas haluaa tehdä valvotun kutsun, se käyttää normaaliin tapaan viestityskerroksen kutsurajapintaa luomaan instanssin joko dynaamisesta kutsuoliosta tai tynkätoteutuksesta. Asiakas ei kuitenkaan suorita varsinaista kutsua itse, vaan kutsuu **MonitoredCall**-luokan tarjoamaa “invoke()”-operaatiota ja antaa sille parametrina luodun olion, kutsuttavan operaation nimen, ja operaation parametrit. **MonitoredCall**-luokan “invoke()”-operaatio verifioi kutsun , kutsuu parametrina annettua oliota, verifioi mahdollisen paluuarvon ja palauttaa sen asiakkaalle.

MonitoredCall-luokalle ei ole väliä onko sille annettava etäpalvelua edustava olio dynaaminen kutsuolio vai tynkätoteutus vai kenties paikallinen Java-olio. Se hyödyntää Java:n tarjoamaa geneeristä reflektiorajapintaa kutsun tekemiseen siten, että annetun olion tyyppi jää sille itselleen tuntumattomaksi. **MonitoredCall** luokka tarjoaa operaatiot sekä asynkronisen yksisuuntaisen (oneway), että synkronisen kaksisuuntaisen (twoway) kutsun tekemiseen.

5.3.3 Paikallisen valvontakerroksen palvelinpuolen sovellusrajapinta

Palvelinpuolella valvotun kutsun tekemiseen käytettävä rajapinta on identtinen asiakaspuolen kanssa. Ero tulee näkyviin lähinnä siinä, miten palvelinsovellus liitetään valvontajärjestelmään. Perustilanteessa viestintäkerros toimittaa viestin “suoraan” kutsutulle palvelulle (välissä voi toki olla erilaisia palvelinpään tynkätoteutuksia ja sovelluspalvelimen sisäisiä toimintoja). Palvelinpään valvontakoneiston kytkemiseksi tarvitaan jokaisen palvelinsovelluksen ympärille erityinen kääresovellus (wrapper) tai palvelinsovelluksen ja viesintäkerroksen väliin erityinen välitinsovellus (proxy). Käytännössä tämä kääre-/välitinsovellus rekisteröidään viestintäkerokselle/sovelluspalvelimelle edustamaan varsinaista palvelusovellusta.

Kääre-/välitinsovellus saa palvelulle menevän kutsun, se käyttää **MonitoredCall**-luokkaa samalla periaatteella kuin asiakassovelluskin sitä käyttää. Kääre-/välitinsovellus antaa **MonitoredCall**-luokalle parametrina kutsuttavan palvelusovellusinstanssin ja siihen liittyvät parametrit. Tämän jälkeen **MonitoredCall**-luokka verifioi kutsun ja sen paluuarvon samalla tavoin kuin asiakaspäässäkin. Verifiointien välissä se suorittaa varsinaisen palveluinstanssille menevän kutsun Javan reflektiomekanismeja käyttäen.

Kuvassa 5.2 tilanne on esitetty sekä itsenäisen Axis-palvelun, että J2EE-alustalle to-

teutetun EJB-palvelun näkökulmasta. Itsenäisen Axis-palvelun tapauksessa Axis-alustan tuottama **BindingImpl**-luokka toimii periaatteessa kääresovelluksena (tai toteutustavasta riippuen joskus välitinsovelluksena) varsinaiselle palvelusovellukselle. Kaikki tietylle Web-palvelulle saapuvat Axis-kutsut toimitetaan siis sitä vastaavalle **BindingImpl**-instanssille. Ilman valvontaa toimivassa tilanteessa **BindingImpl** joko itsessään sisältäisi Web-palveluoperaatioiden toteutuksen tai useimmiten ohjaisi saapuneen kutsun erilliselle palvelun toteuttavalle objektille (jonka se kääresovelluksena toimiessaan itse instantioisi). Valvotussa tapauksessa **BindingImpl** sen sijaan toimiikin siis siten, että se välittää palvelun toteuttavan objektin viitteen ja kutsuttavan operaation parametreineen **MonitoredCall**-luokalle aivan kuten asiakkaana toimiva sovelluskin tekee.

EJB-toteutusten tapauksessa palvelinpään valvontaintegraatiossa on yksinkertaisinta käyttää apuna erillistä proxy-papua. Proxy-papu rekisteröidään tietyn Web-palvelun toteuttajaksi, joten kaikki Web-palvelulle saapuvat palvelukutsut kulkeutuvat sille. Proxy-papu kutsuu varsinaisen liiketoimintalogiikan sisältävää erillistä papua (tai erillistä luomaansa Java-objektia jolloin se toimii kääresovelluksen tavoin) **MonitoredCall**-luokan kautta. Näin proxy-papu toimii periaattessa samalla tavoin, kuin **BindingImpl**-luokka itsenäisten palvelusovellusten tapauksessa. Proxy-papu kannattaa yleensä toteuttaa tilattomana sessiopapuna (Stateless Session Bean) tai viestiorientoituneena papuna (Message Driven Bean).

5.3.4 Sovellusrajapinnassa kuljetettava valvontametatieto

Jotta valvontainfrastruktuuri voisi verifioida suoritettavan palvelukutsun oikeellisuuden, täytyy kutsun sisältää valvontaan liittyvää lisätietoa. Päätelläkseen onko suoritettava palvelukutsu oikeellinen, tarvitsee valvontakerros käyttöönsä erityisen (jokaisen palvelukutsun mukana kulkevan) **MonitoringData**-tietueen sisällön.

MonitoringData sisältää jokaista tehtävää palvelukutsua kohden sopimussession tunnisteen, keskusteluyhteyden tunnisteen, lähettävän roolin tunnisteen, vastaanottavan roolin tunnisteen, lähetettävän viestin viestityypin sekä synkronisessa kommunikaatiossa lisäksi vastaanotettavan viestin viestityypin. Näiden tietojen perusteella paikallinen valvontakerros pystyy löytämään välimuististaan tai keskitetyltä valvontakerrokselta oikean koreografiatason valvontametatiedon ja verifioimaan palvelukutsun suhteessa valvontametatietoon. Huomattavaa on, että epookin tunnistetta ei tarvitse sisällyttää palvelukutsuun, koska sopimussessio on kerrallaan vain yhdessä epookissa. Kaikki tietyn session tunnisteen sisältävät viestit tulkitaan aina session aktiivisen epookin kontekstissa.

MonitoringData-tietue tulee valvontakoneiston näkökulmasta kutsun tekevältä asiakkaalta yhtenä kutsun parametrina. Tietue kulkeutuu Web-palvelun toteutuspuolelle lähetetyn viestin (esimerkiksi SOAP-viestin) mukana. Nykyinen prototyyppitoteutus hyväksyy sekä erityisen **MonitoringData** Java-tyypin instanssin että merkkijonoksi (string) pakatun kuvan 5.3 skeeman mukaisen XML-dokumentin.

Koska nykyinen paikallinen valvontakerros toimii aidosti viestikerroksen päällä, ei se näe varsinaisia SOAP-viestejä missään vaiheessa. Näin ollen on paikallisen valvonnan kannalta tuntumatonta pakataanko **MonitoringData** SOAP-viestien runkoon, otsakkeeseen vai liitteeseen. Pakkausmenetelmä määritellään Web-palvelun WSDL-rajapinnassa,

```
Skeema:
<xs:schema targetNamespace="http://www.cs.helsinki.fi/group/web-pil/schema/MonitoringData.xsd"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.cs.helsinki.fi/group/web-pil/schema/MonitoringData.xsd">

           <xs:element name="MonitoringData">
             <xs:complexType>
                <xs:sequence>
                  <xs:element name="sessionID" nillable="false" type="xsd:string"/>
                  <xs:element name="senderID"  nillable="false" type="xsd:string"/>
                  <xs:element name="receiverID" nillable="false" type="xsd:string"/>
                  <xs:element name="conversationID" nillable="false" type="xsd:string"/>
                  <!-- Poikkeus alla olevaan: MonitoredCall-luokka ei hyväksy null-arvoja
                       requestMSGType kentässä ja responseMSGType kentässäkin vain, kun
                       kyseessä on asynkroninen (oneway) kutsu. Edellisestä erikoistettu
                       MonitoredAxisCall-luokka hyväksyy null-arvot kaikissa tapauksissa.
                       Jos requestMSGType kenttä on null niin MonitoredAxisCall asettaa
                       siihen Axis:n vakionimeämiskäytäntöä vastaavan ‘‘palveluoperaatio’’+’’Request’’
                       viestityypin. Jos responseMSGType kenttä on null niin MonitoredAxisCall
                       asettaa siihen Axis:n vakionimeämiskäytäntöä vastaavan ‘‘palveluoperaatio’’+’’Response’’
                       viestityypin.->
                  <xs:element name="requestMSGType" nillable="true" type="xsd:string"/>
                  <xs:element name="responseMSGType" nillable="true" type="xsd:string"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
</xs:schema>

Esimerkki:
<MonitoringData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:="http://www.cs.helsinki.fi/group/web-pil/schema/MonitoringData.xsd>

                <sessionID>PurchaseSession</sessionID>
                <senderID>Buyer</senderID>
                <receiverID>Seller</receiverID>
                <conversationID>PurchaseConversation</conversationID>
                <requestMSGType>PurchaseOrder</requestMSGType>
                <responseMSGType>Invoice</responseMSGType>
</MonitoringData>
```

Kuva 5.3: **MonitoringData**-tietueen XML-esitystapaa vastaava skeema ja esimerkki XML-dokumentti.

joka ei siis suoraan näy valvontakerrokselle lainkaan.

5.3.5 Keskustelua sovellusrajapinnasta

Sovellusrajapinta toimii nykyisellään Javan reflektiomekanismeja käyttäen sovelluskerroksen ja viestintäkerroksen välissä. Rajapinta olisi mahdollista integroida myös osaksi viestintäkerrosta. Tämän lähestymistavan järkeväksi toteuttamiseksi viestintäkerroksen tulisi kuitenkin tarjota standardirajapinta omien keskeytinsovellusten (interceptor) liittämiseksi viestivirtaan. Web-palvelumaailmassa standardeja keskeytinrajapintoja ei kuitenkaan vielä ole. Jotkut viestintäympäristöt tarjoavat jonkinlaisen keskeytinrajapinnan (esimerkiksi “Axis Handlers”), mutta mitään tuoterajat ylittävää ratkaisua ei ole olemassa. Tästä syystä geneerisimmäksi ratkaisuksi osoittautui oma valvonnan rajapintakerros, joka toisaalta lukkiuttaa sovellukset tähän rajapintakerrokseen, mutta samalla mahdollistaa periaatteessa minkä tahansa olemassa olevan viestintäkerroksen käytön.

Sovellusrajapinnan toteutuksen yksityiskohtiin ei mennä tässä työssä. Web-Pilarcos projektin prototyyppi ja siihen liittyvä dokumentaatio kuvaa rajapintojen rakenteen teknisellä tasolla. Mainittavaa kuitenkin on se, että **MonitoredCall**-luokasta on olemassa myös erikoistettu **MonitoredAxisCall**-luokka, joka tarjoaa Axis-spesifisen kutsurajapinnan. Periaatteessa luokka on toiminnallisuudeltaan sama kuin alkuperäinen **MonitoredCall**, mutta sen operaatiot eivät ole täysin geneerisiä, vaan ne on tyypitetty siten, että niitä on

Axis-pohjaista viestityskerrosta käyttävien sovellusten helpompi ja turvallisempi käyttää.

5.3.6 Paikallisen valvontakerroksen rajapinta keskitetyn valvontakerroksen kanssa

Paikallisen valvontakerroksen rajapinta keskitetyn valvontakerroksen kanssa on toteutettu **CentralizedMonitoring** Web-palvelurajapintana kuten kuvasta 5.2 voidaan nähdä. **CentralizedMonitoring** Web-palvelurajapintaan kuuluu paitsi paikallisen valvontakerroksen käyttämät operaatiot, myös sopimustasolla toimivan verkostonhalinta-agentin käyttämät operaatiot. Tässä kappaleessa keskitytään **CentralizedMonitoring**-rajapinnan siihen osaan, jota paikallinen valvontakerros käyttää ja rajapinnan muista osista puhutaan myöhemmin keskitetyn valvontakerroksen ja verkostonhallinta-agentin rajapintaa käsittelevässä luvussa.

Paikallisen valvonnan näkökulmasta **CentralizedMonitoring** tarjoaa operaatiot koreografiatason metatiedon pyytämiseen, koreografiatason tilamuutosten persistenttiin rekisteröimiseen sekä tehtävävaikutusten ja virheilmoitusten eteenpäin toimittamiseen.

Koreografiatason metatiedon pyytämiseen tarkoitetun rajapinnan avulla paikallinen valvontakerros voi pyytää session tunnistetta ja roolin tunnistetta vasten aktiiviseen rooliepookkiin liittyvän koreografiametatiedon. Välimuistia käyttävä koreografiataso tyypillisesti pyytää metatiedon jokaisen epookin alussa ja poistaa vanhan metatiedon epookin lopussa. Ilman välimuistia toimiva koreografiataso pyytää metatiedon erikseen jokaista valvontarutiinien suoritusta varten.

Koreografiatason tilamuutosten persistenttiin rekisteröintiin tarkoitetun rajapinnan avulla ilman välimuistia toimiva tai karkeaa virheensietoa käyttävä paikallinen valvontakerros voi rekisteröidä koreografiatason tilamuutokset keskitetylle valvontakerrokselle. Keskitetty valvontakerros pitää tällöin persistentissä muodossa yllä koreografian nykytilaa. Nykytila esitetään yhtenä merkkijonona (string). Jos tilojen tunnisteet on tuotettu automaattisesti voi tilan tunniste olla esimerkiksi sitä vastaava numeroarvo (kuten tila numero “5”).

Suurin syy siihen miksi koreografiatason tilatietoa ylläpidetään persistentisti keskitetyn valvontakerroksen puolella on, että ilman välimuistia toimiva paikallinen valvontakerros voi mahdollisesti toimia myös ilman pysyväismuistia (esimerkiksi ilman kovalevyä). Tällöin paikallisia persistenttejä tietoja on mahdotonta ylläpitää. Edellisen syyn ohella keskitettyä valvontakerrosta on luontevaa käyttää persistentin tiedon tallentamispaikkana, koska se on toteutettu J2EE-alustalle, joka tarjoaa helpon ja skaalautuvan ympäristön persistenttien tietojen ylläpitämiseksi.

Tehtävävaikutusten ja virheilmoitusten toimittamiseen tarkoitettu osa **CentralizedMonitoring**-rajapintaa koostuu synkronisista Web-palvelukutsuista. Kutsuissa kulkee parametreina session tunnisteen ja lähettävän roolin tunnisteen lisäksi joko aktivoituneen tehtävävaikutuksen tunniste tai havaitun virhetilanteen tunniste.

Koreografiatasolla on mahdollista havaita kahdenlaisia virhetilanteita. Ensimmäinen virhetilanne jonka paikallinen valvontakerros voi havaita on sellainen, jossa vastaanotettu viesti sisältää sellaisen session tunnisteen, joka ei vastaa mitään tunnettua sessiota. Täs-

tä seurauksena on **UnknownSessionException**-virhetilanne (Java-toteutuksessa virhetilanteet on toteutettu Java-poikkeuksina ja siksi ne ovat toteutuksessa **Exception**-luokan alaluokkia ja nimetty poikkeuksiksi eikä virheiksi). Toinen havaittava virhetilanne on sellainen, jossa sessio on tunnettu, mutta viesti ei vastaa sovittua koreografiaa. Tästä seuraa **MonitoringException**-virhetilanne. MonitoringException-virhetilanne seuraa siis tilanteista, joissa viestille pystytään löytämään sitä vastaava koreografiatason tila-automaatti, mutta viestiä ei pystytä tulkitsemaan yhdeksikään sallituksi tilasiirtymäksi tila-automaatin sen hetkisessä tilassa.

Toteutettujen virhetilanteiden lisäksi olisi periaatteessa mahdollista käyttää ajastimia havaitsemaan muuten oikeellisten, mutta väärään aikaan (liian aikaisin / liian myöhään) tapahtuvia viestien lähettämisiä/vastaanottamisia. Tällaisia reaaliaika- ja palvelunlaatuominaisuuksia ei nykyisellään kuitenkaan ole ajateltu tässä tutkielmassa esitettyjen valvontakerrosten valvottaviksi, vaan niitä on ajateltu valvottavan joko sovelluksissa (jos aikarajat liittyvät liiketoimintalogiikkaan) tai viestikerroksessa (jos aikarajat liittyvät tekniseen kommunikointiin).

Tehtävämallivaikutusten ja virheilmoitusten välittämiseen tarkoitettujen palvelukutsujen paluuarvoja käytetään paikallisen valvontakerroksen välimuistin käytön ohjaamiseen. Tästä puhutaan lisää myöhemmässä vaiheessa paikallisen valvontakerroksen sisäistä toimintaa käsittelevässä alikappaleessa.

5.3.7 Valvonnan erilaisten moodien (proaktiivinen, aktiivinen, passiivinen) toteutus ja toiminta

Paikallisen valvonnan ytimen muodostaa kuvassa 5.2 näkyvä **LocalizedMonitor**-luokka. Kuten aiemmin jo kerrottiin, kyseinen luokka toimii eräänlaisena paikallisen valvontakerroksen koordinaattorina, jonka kanssa kerroksen muut luokat keskustelevat ja joka koordinoi koko kerroksen toimintaa. Perusperiaate on, että muut luokat ovat yhteydessä ainoastaan **LocalizedMonitor**-luokkaan. Ainoan poikkeuksen edelliseen sääntöön tekee liittymä **MonitoredCall**-luokan ja **LocalizedMonitor**-luokan välillä.

Valvonta voi toimia kolmessa eri moodissa: passiivisesti, aktiivisesti tai proaktiivisesti. Käytännön ohjelmointiteknisistä syistä johtuen passiivisen, aktiivisen ja proaktiivisen moodin välisten toiminnallisuuksien ero on toteutettu **LocalizedMonitor**-luokan sijaan **MonitoredCall**-luokassa. Valinnalle on myös arkkitehturaaliset perusteet, koska näin **LocalizedMonitor**-luokan rakenne saadaan pysymään yksinkertaisempana.
LocalizedMonitor-luokan ei tarvitse olla lainkaan tietoinen erilaisista valvontamoodeista, vaan se toimii käytännössä aina proaktiivisessa moodissa. Proaktiivinen valvonta on siis valittu perustilanteeksi, jonka johdannaisia/yksinkertaistuksia muut moodit ovat.

Normaalissa proaktiivisessa moodissa toimittaessa **MonitoredCall** toimittaa verifiointitiedon synkronisella operaatiokutsulla luomalleen **LocalizedMonitor**-oliolle.
LocalizedMonitor suorittaa verifioinnin ja muut siihen liittyvät toimenpiteet ja joko palaa operaatiosta normaalisti tai poikkeuksella. Jos operaatio palaa normaalisti, niin **MonitoredCall** päättelee tästä verifioinnin onnistuneen ja jatkaa siitä mihin jäi. Jos taas operaatio palaa poikkeuksella (**UnknownSessionException** tai **MonitoringException** tai jokin Ja-

va:n yleinen poikkeus), niin **MonitoredCall** päättelee verifioinnin epäonnistuneen syystä tai toisesta ja toimittaa poikkeuksen sitä kutsuneelle sovellukselle tai kääre-/välitinsovellukselle.

Aktiivisessa moodissa toimittaessa **MonitoredCall** ei suorita blokkaavaa verifiointikutsua, vaan sen sijaan luo **LocalizedMonitor**-olion normaaliin tapaan ja tämän jälkeen antaa sen suoritettavaksi kuvassa 5.2 näkyvälle **ConcurrentWorker**-oliolle. **ConcurrentWorker** on oma säikeensä, joka pitää yllä aktiivisen valvonnan suoritusjonoa. Se odottaa kunnes jonoon ilmestyy uusi **LocalizedMonitor**-olio, minkä jälkeen se toimii kuten **MonitoredCall**-luokka proaktiivisessa tapauksessa ja kutsuu **LocalizedMonitor**-olion verifiointioperaatiota. Näin **MonitoredCall**-luokka ja siten myös sitä kutsuvat sovellukset voivat jatkaa suoritustaan ilman, että niiden tarvitsee jäädä odottamaan verifioinnin lopputulosta. Jos verifioinnin aikana havaitaan virhetilanne, niin **ConcurrentWorker** ei reagoi siihen liittyvään poikkeukseen mitenkään, vaan normaalisti poistaa verifioinnin suorittaneen **LocalizedMonitor**-olion jonosta ja siirtyy seuraavaan olioon. Näin ollen aktiivisen valvonnan tapauksessa sovellukset ja paikallinen valvontakerros toimivat rinnakkain ja sovellusten sallitaan toimivan myös koreografian vastaisesti.

Karkean vikasieto-ominaisuuden ollessa käytössä, **ConcurrentWorker** pitää aktiivisen valvonnan suoritusjonoa yllä persistentissä muodossa paikallisessa tiedostossa. Näin ollen karkeaa vikasieto-ominaisuutta ei voida käyttää kuin niissä tapauksissa, että sovelluskoneessa on paikallinen pysyväismuisti. Käytännössä tiedostoon tallennetaan sarjallistettu **java.util.Vector**-luokan ilmentymä, joka sisältää suoritusjonossa olevat oliot. Uusi versio vektorista talletetaan tiedostoon aina aktiivisen valvontajonon kasvaessa tai lyhentyessä. Koneen mahdollisen kaatumisen ja uudelleen käynnistyksen jälkeen **ConcurrentWorker** jatkaa valvontatapahtumien läpikäymistä siitä jonon kohdasta, johon se ennen kaatumistaan jäi.

Toteutetun vikasiedon karkeus näkyy käytännössä siinä, että parhaillaan suorituksessa olevan verifiointitapahtuman atomisuutta ei taata. Jos siis koneen kaatuminen tapahtuu sopivassa kohdassa verifiointia suorittavaa algoritmia voi käydä vaikkapa niin, että koreografiatason tila-automaattia ehditään päivittää, mutta keskitetylle valvonnan tasolle ei ehditä toimittaa tilasiirtymän tuottamaa tehtävävaikutusta. Tällöin tehtävämallin ja koreografiatason tila-automaatin yhtenäisyys särkyy. Kunnollisen vikasietoisuuden rakentaminen ei ole ollut tämän tutkielman eikä laaditun tutkimusprototyypin tavoitteena, mutta tutkielman lopussa käsitellään kuitenkin hieman vikasietoisuusominaisuuksien jatkokehityssuuntia.

Passiivisen valvonnan tapauksessa varsinaista suoritusaikaista valvontaa ei suoriteta lainkaan. Käytännössä tämä siis tarkoittaa sitä, että **MonitoredCall**-luokka ei luo lainkaan **LocalizedMonitor**-luokan ilmentymää, vaan sen sijaan tekee lokimerkinnän tapahtumasta kuvassa 5.2 näkyvää **Logger**-luokkaa käyttäen. Näin ollen passiivisen valvonnan tapauksessa suoritus ei kulkeudu lainkaan suoritusaikaista valvontaa tekeviin paikallisen valvontakerroksen osiin.

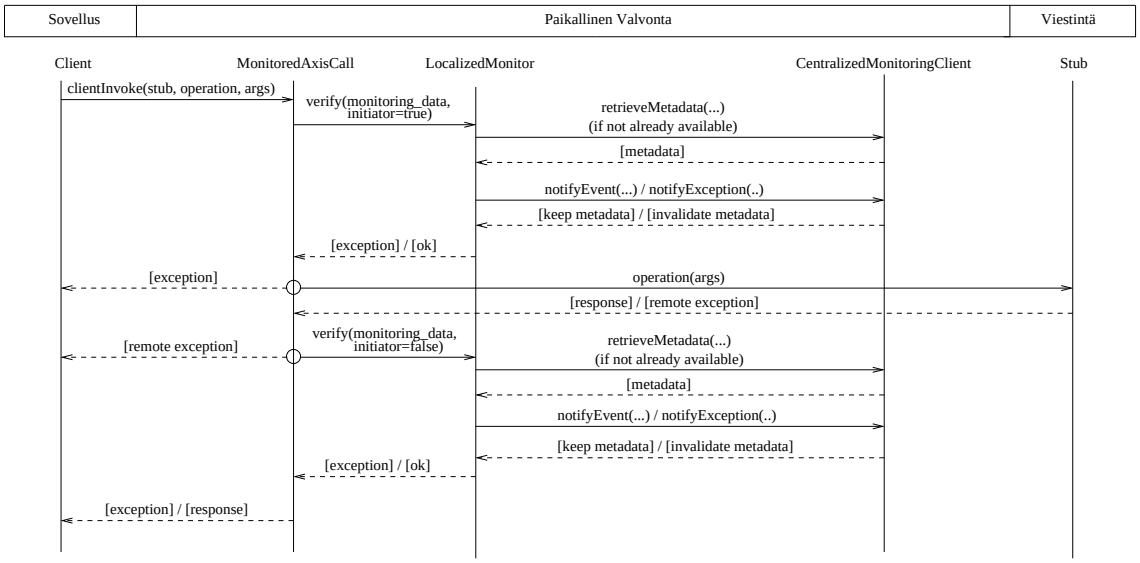
MonitoredCall-luokasta tiettyä viestintäkerroksta varten erikoistetut rajapintatoteutukset (kuten **MonitoredAxisCall** perivät valvontamoodeja käsittelevän osion **MonitoredCall**-luokalta. Näin ollen valvontamoodeja käsittelevä osuus on aito osa paikallisen valvonnan muodostamaa sovelluskehystä eikä sitä tarvitse erikseen toteuttaa jokaiseen eril-

liseen rajapintatoteutukseen.

5.3.8 Paikallisen valvontakerroksen suoritusaikaiset vuorovaikutukset

Kuten edellä on esitetty, paikallinen valvontakerros sisältää muun muassa rajapintana toimivan **MonitoredCall**-luokan, **ConcurrentWorker**-luokan, joka mahdollistaa aktiivisen valvontamoodin käytön sekä **Logger**-luokan, jonka avulla voidaan kirjoittaa valvontalokia. Paikallisen valvonnan ytimestä puhuttaessa tarkoitetaan kuitenkin ennenkaikkea koordinointia hoitavaa **LocalizedMonitor**-luokkaa sekä sen kanssa vuorovaikuttavia niin sanottuja algoritmiluokkia.

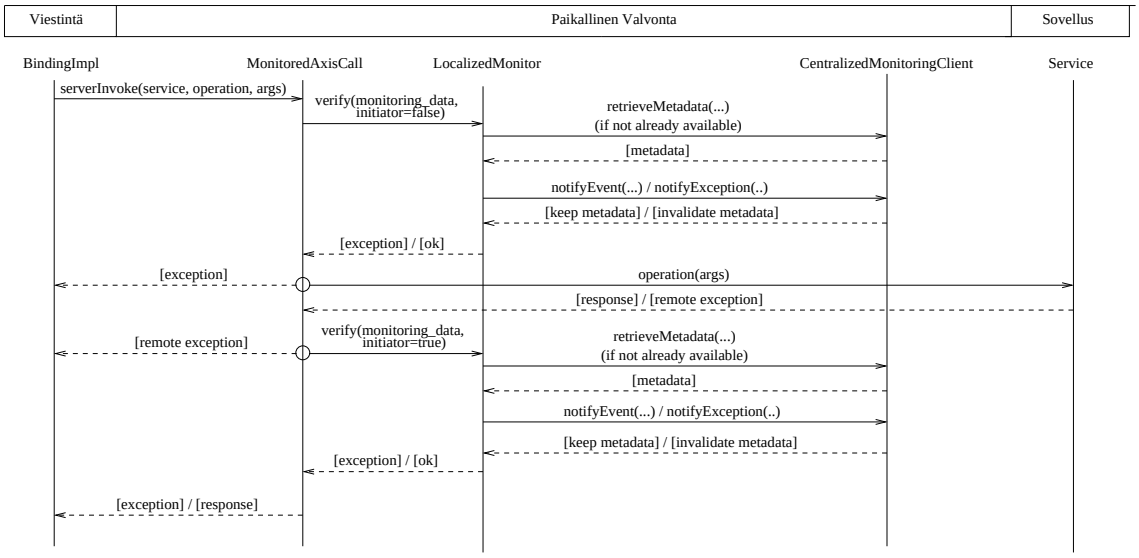
Kuvassa 5.2 on paikallisessa valvontakerroksessa näkyvissä kaksi algoritmiluokkaa: **MatrixAlgorithm** ja **HashMapAlgorithm**. Algoritmiluokat sisältävät koreografiatason tila-automaattimallin läpikäyntiin tarkoitetut algoritmit. Algoritmiluokkien toiminnallisuuden voisi periaatteessa upottaa osaksi **LocalizedMonitor**-luokkaa, jos koreografiatason valvontametatiedolle olisi olemassa standardin mukainen koodaustapa. Algoritmiluokkien erottamisella omiksi erillisiksi luokikseen mahdollistetaan kuitenkin uusien koreografiatason algoritmien helppo lisääminen.



Kuva 5.4: Paikalliseen proaktiiviseen valvontakerrokseen liittyvät vuorovaikutukset asiakassovelluksen puolella synkronisessa kommunikaatiossa.

Paikallisen valvontakerroksen eri komponenttien toimintaa proaktiivisessa valvontamoodissa on visualisoitu kahdessa erillisessä kuvassa. Kuvassa 5.4 on kuvattu sekvenssikaaviomaisesti asiakassovelluksen tekemä tynkätoteutusta hyödyntävä synkroninen Axis-kutsu. Kuvassa 5.5 on puolestaan kuvattu vastaava kutsu palvelinpäässä. Kuvissa on ymmärtämisen helpottamiseksi kuvattu vain olennaisin toiminnallisuus. Seuraavaksi pureudutaan tarkemmin (kuvissakin näkyviin) paikallisen valvonnan suoritusaikaisiin vuoro-

vaikutuksiin.



Kuva 5.5: Paikalliseen, proaktiiviseen, valvontakerrokseen liittyvät vuorovaikutukset palvelusovelluksen puolella synkronisessa kommunikaatiossa.

Asiakaspäässä valvontarutiinien suoritus käynnistyy, kun asiakas tekee valvotun kutsun **MonitoredCall**-luokkaa (tai jotain sen johdannaista kuten **MonitoredAxisCall**-luokkaa) käyttäen. Kutsun tekemiseen käytetään **clientInvoke** operaatiota, jonka avulla valvontakoneisto tunnistaa, että kyseessä on viestin lähettäjän suorittama pyyntö.

Asiakas antaa **MonitoredCall**-luokalle parametrina kutsuttavan olion, kutsuttavan operaation nimen ja operaation parametrit (sisältäen **MonitoringData**-tietueen). Vastaanotettuaan asiakkaan pyynnön, **MonitoredCall**-luokka tutkii Javan virtuaalikoneen käynnistymisen aikana ladattuja konfigurointitietojaan ja päättelee näiden perusteella onko käytössä passiivinen, aktiivinen vai proaktiivinen valvontamoodi.

Passiivisen valvonnan tapauksessa **MonitoredCall** kirjoittaa ainoastaan valvontalokia. Aktiivisen valvonnan tapauksessa **MonitoredCall** luo **LocalizedMonitor**-olion ja antaa sen **ConcurrentWorker**-olion suoritettavaksi. Proaktiivisen valvonnan tapauksessa **MonitoredCall** kutsuu itse (blokkaavasti) luomaansa **LocalizedMonitor**-oliota ja jää odottamaan operaation palauttamaa valvontatulosta.

Huomattavaa on, että nykyisessä toteutuksessa valvontamoodi on sama kaikille tietyssä virtuaalikoneessa toimiville sovelluksille. **MonitoredCall**-luokkaa olisi periaatteessa mahdollista päivittää siten, että valvontamoodi voisi olla esimerkiksi sopimus tai sopimussessio kohtainen.

Kun **LocalizedMonitor**-olio alkaa suorittaa pyydettyä verifiointia sille toimitetun **MonitoringData**-olion pohjalta, tutkii se ensin välimuististaan onko **MonitoringData**-olion sisältämää session ja roolin tunnistetta vastaava metatieto jo olemassa. Roolin tunnisteena käytetään sen organisaation roolia, jonka alaisuudessa paikallinen valvontakerros toimii. Liittyykö valvontapyyntö lähettävään vai vastaanottavaan rooliin päätellään jo rajapinnassa (asiakas- ja palvelinpuolella toimiville sovelluksille on rajapinnassa erikseen

nimetyt, mutta muuten identtiset operaatiot).

Jos metatietoa ei ole paikallisessa välimuistissa saatavilla (tai jos välimuisti on kokonaan poissa käytöstä), tekee **LocalizedMonitor**-olio kutsun keskitetylle valvontakerrokselle luomalla **CentralizedMonitoringClient**-olion ja käyttämällä sitä kutsun tekemiseen. **CentralizedMonitoringClient**-olio tekee Web-palvelukutsun tunnetussa osoitteessa (URL) olevaan **CentralizedMonitoring** Web-palveluun ja saa siltä vastineena joko session aktiiviseen rooliepookkiin liittyvän koreografiatila-automaatin tai **null**-arvon ilmaisemaan sitä, että annetut tunnisteet eivät kuvaudu yhdeksikään tunnetuksi automaatiksi.

Jos metatietoa ei löydetty, niin valvontarutiinin suoritus keskeytetään ja **MonitoredCall**-luokalle toimitetaan **UnknownSessionException**-poikkeus, joka proaktiivisen valvonnan tapauksessa edelleen toimitetaan kutsujalle. Jos keskitetyltä valvontakerrokselta saatiin sopiva metatieto, niin se talletetaan välimuistiin ja jatketaan valvontarutiinin suoritusta. Välimuisti on nykyisessä prototyypissä toteutettu **java.util.HashMap**-olion ylläpitämänä hajautustauluna, johon talletetaan katenoitua "**session tunniste** + **roolin tunniste**"-hajautusavainta käyttäen kukin saatu koreografiatila-automaatti Java-olioksi muunnettuna.

Kun koreografiatason metatieto on talletettu välimuistiin (tai jos se löytyi sieltä jo ennestään), jatketaan varsinaisen valvontatoiminnallisuuden suoritusta aktivoimalla sopiva valvonta-algoritmi. Koreografiametatieto kantaa mukanaan tietoa siitä, minkä algoritmin mukaista koodausta se käyttää. Vaihtoehtoina nykyisellään ovat matriisimuotoinen (**MATRIX_ENCODING**) ja hajautustaulumuotoinen (**HASMAP_ENCODING**) koodaus.

LocalizedMonitor valitsee käytettävän algoritmin metatiedossa olevan koodauksen mukaan, instantioi oikean algoritmiluokan ja pyytää sitä verifioimaan **MonitoringData**-olion suhteessa siihen liittyvään koreografiametatietoon (tätä osaa ei ole piirretty näkyviin kuviin 5.4 ja 5.5). Jos koodaus on tuntematon lopetetaan operaation suoritus ja palautetaan **MonitoredCall**-luokalle **UnknownEncodingException**-poikkeus. Proaktiivisen valvonnan tapauksessa **MonitoredCall**-luokka edelleen toimittaa poikkeuksen kutsujalle.

Valittu valvonta-algoritmi verifioi ensin sen, onko **MonitoringData**-olion määrittelemää tilasiirtymää olemassa annetussa automaatissa. Jos tilasiirtymää ei löydy, niin valvonta-algoritmi lopettaa suorituksensa poikkeuksellisesti syynä **MonitoringException**. Proaktiivisen valvonnan tapauksessa poikkeus toimitetaan siis rajapintaluokan kautta valvontakutsun tekijälle asti.

Jos sopiva tilasiirtymä löytyy, niin valvonta-algoritmi pyytää sitä kutsunutta **LocalizedMonitor**-oliota siirtämään koreografiatila-automaatin uuteen tilaan. Jos välimuisti on käytössä talletetaan uusi tila siellä olevaan automaattiin. Jos välimuisti on poissa käytöstä lähettää **LocalizedMonitor** uuden tilan keskitetylle valvontakerrokselle **CentralizedMonitoringClient**-oliota apuna käyttäen. Jos välimuisti on käytössä ja karkea vikasietoisuus on päällä, laitetaan tieto sekä välimuistiin että replikoidaan se keskitetylle valvontakerrokselle.

Kaikkiin aiemmin mainittuihin virhetilanteisiin liittyen tehdään aina virhenotifikaatio keskitetylle valvontakerrokselle ennenkuin syntynyt poikkeus palautetaan kutsujalle. **LocalizedMonitor**-olio lähettää **CentralizedMonitoringClient**-oliota apunaan käyttäen

virhetilanteesta tiedon keskitetylle valvontakerrokselle ja saa paluuarvonaan totuusarvon (boolean) joka kertoo säilytetäänkö nykyinen metatieto välimuistissa ja jatketaan suoritusta normaalisti (**true**) vai johtaako virhetilanne metatiedon vanhenemiseen ja poistamiseen (**false**).

Suorituksensa lopuksi valvonta-algoritmi tutkii seuraako tehdystä tilasiirtymästä jokin tehtävävaikutus. Jos kyllä, niin valvonta-algoritmi notifioi **LocalizedMonitor**-oliota tästä. **LocalizedMonitor** lähettää tiedon tehtävävaikutuksesta **CentralizedMonitoringClient**-oliota käyttäen keskitetylle valvontakerrokselle ja saa paluuarvona totuusarvon (boolean). Edelleen saatu totuusarvo määrittelee sen, säilytetäänkö nykyinen metatieto välimuistissa (**true**) vai poistetaanko se (**false**). Poistamispyyntö tarkoittaa käytännössä sitä, että koreografiatila-automaatti (ja siis myös tehtävämalli) on päätynyt lopputilaansa ja rooliepookki on näin ollen päättynyt.

Koreografiatason metatiedon virtaus paikallisen ja keskitetyn valvontakerroksen välillä tapahtuu siis aina siten, että aloite tulee paikalliselta valvontakerrokselta. Jos välimuisti on pois käytöstä, metatieto pyydetään erikseen jokaista valvontarutiinin suoritusta varten. Jos välimuisti on pois käytöstä tai karkea vikasietoisuus on päällä, toimitetaan koreografiatason metatiedon muutokset keskitetylle valvontakerrokselle aina kun niitä tapahtuu.

Välimuistin ollessa käytössä, paikallinen valvontakerros pyytää uutta metatietoa vain silloin, kun se joutuu verifioimaan ennestään tuntemattomalta sessio/rooli-parilta lähtevän tai niille saapuvan palvelupyynnön. Välimuisti tyhjennetään, kun keskitetty valvontakerros sitä pyytää. Tyhjennys voi tapahtua virhetilanteen tai rooliepookin päättävän tehtävävaikutuksen seurauksena. Välimuistin toimintaan voisi liittää lisäksi ajastinjärjestelmän, jolloin metatiedot voisi poistaa välimuistista myös vanhentuneeseen aikaleimaan perustuen.

5.4 Keskitetyn valvontakerroksen rakenne, rajapinnat ja toiminta

Toisin kuin paikallinen valvontakerros, keskitetty valvontakerros on aito pysyvää tilaa ylläpitävä J2EE-sovellus. Keskitetty valvontakerros tarjoaa Web-palvelurajapinnan sekä paikalliselle valvonnalle että sopimustasolla toimivalle verkostonhallinta-agentille. Keskitetty valvontakerros koostuu kuvassa 5.2 näkyvällä tavalla yhdestä tilattomasta sessiopavusta (**CentralizedMonitoringSessionBean**), kolmesta yksilöpavusta (**SessionEntityBean**, **RoleEntityBean** ja **TaskModelEntityBean**), sekä algoritmiluokista (**MatrixAlgorithm** ja **HashMapAlgorithm**). Lisäksi kerrokseen kuuluu verkostonhallinnan tarjoamaa **NetworkManagementAgent** Web-palvelua käyttävä **NetworkManagementAgentClient**.

Aivan kuten **LocalizedMonitor**-toteutus paikallisessa valvontakerroksessa, myös **CentralizedMonitorSessionBean** muodostaa ympärilleen tähtimäisen rakenteen. Keskitetyn valvontakerroksen suunnittelun taustalla onkin sama perusperiaate kuin paikallisella valvontakerroksella: kaikki muut toimijat kommunikoivat tähden keskellä olevan **CentralizedMonitoringSessionBean**-luokan kanssa ja tämä koordinoi hajautustason sisäisiä vuorovaikutuksia. Koordinointiominaisuuksiensa lisäksi **CentralizedMonitoring-**

SessionBean toteuttaa **CentralizedMonitoring** Web-palvelun.

Web-palvelun ja koordinointitoiminnallisuuden toteuttavan **CentralizedMonitoringSessionBean**-luokan ohella keskitetyssä valvontakerroksessa on siis kolme yksilöpapua. Yksilöpapujen tehtävänä on säilyttää persistentissa muodossa kukin itseensä liittyvän valvonnan tason metatietoa ja tilaa. **SessionEntityBean** pitää yllä sopimussessiotason meta- ja tilatietoa, **RoleEntityBean** pitää yllä roolitason meta- ja tilatietoa ja **TaskModelEntityBean** pitää yllä tehtävämallitason meta- ja tilatietoa. Koska tiettyyn tehtävämalliin liittyy nykytoteutuksessa aina yksi koreografia, pitää **TaskModelEntityBean** yllä myöskin koreografiatason metatietoa ja tietyissä (aiemmin esitellyissä) tapauksissa myöskin tilatietoa.

Keskitetyn valvontakerroksen algoritmiluokat (**MatrixAlgorithm** ja **HashMapAlgorithm**) puolestaan vastaavat tehtävämallin perustana olevan tila-automaatin läpikäynnistä ja tilan päivittämisestä. Keskitetyn valvontakerroksen tapauksissa algoritmiluokilla ei kuitenkaan ole samanlaista valvontatehtävää kuin paikallisen valvontakerroksen vastaavilla. Tehtävämallitasolla kun oletuksena on, että virheellisiä siirtymiä ei tapahdu, vaan virheellisten siirtymäyritysten havaitseminen tapahtuu jo koreografiatasolla ja tehtävämallitasolle välittyvät tehtävävaikutukset on jo kertaalleen verifioitu oikeellisiksi.

Paikallisen valvontakerroksen toiminnan yhteydessä määriteltiin myös paikallisen ja keskitetyn valvontakerroksen välinen rajapinta. Tämä rajapinta muodostaa osan **CentralizedMonitoring** Web-palvelun koko toiminnallisuudesta. Paikalliselle valvontakerrokselle tarjottujen operaatioiden lisäksi **CentralizedMonitoring** sisältää verkostonhallinta- ja sopimustasolle tarjotut operaatiot valvontametatiedon rekisteröintiin ja konfigurointiin, valvonnan tilatiedon kysymiseen ja muista organisaatiosta saapuvan valvonnan tilatiedon käsittelyyn. Keskitetyn valvontakerroksen ja verkostonhallintapalveluiden väliseen rajapintaan kuuluu myös keskitetyn valvontakerroksen käyttämä **NetworkManagementAgent** Web-palvelu, jota keskitetty valvontakerros käyttää tilatiedon muutoksista ja epookkien loppumisista notifioimiseen.

CentralizedMonitoring Web-palvelu tarjoaa erilliset operaatiot sessiotason, roolitason, tehtävämallitason ja koreografiatason valvontametatiedon rekisteröimiseen. Sessiotason valvontametatiedot talletetaan **SessionEntityBean**-pavun avulla, roolitason valvontametatiedot **RoleEntityBean**-pavun avulla, ja tehtävämallitason ja koreografiatason valvontametatiedot **TaskModelEntityBean**-pavun avulla J2EE-alustan käyttämään tietokantaan. Jokainen **CentralizedMonitoring** Web-palvelulle saapuva palvelukutsu käsitellään siis ensin **CentralizedMonitoringSessionBean**-pavun toimesta, joka puolestaan kutsuu yksilöpapujen operaatioita tilanteeseen sopivalla tavalla. **CentralizedMonitoringSessionBean** toimii siis eräänlaisena yksilöpapuja käyttävänä fakaadina (facade).

Koreografiatasolta saapuvat tehtävävaikutukset ohjataan käsiteltäväksi niitä läpikäyville algoritmitoteutuksille. Kun algoritmitoteutus on siirtänyt tehtävämallin uuteen tilaansa, palautuu suoritus **CentralizedMonitoringSessionBean**-pavulle. Papu tallentaa tehtävämallin uuden tilan **TaskModelEntityBean** yksilöpapua käyttäen. Jos tehtävämalli päätyy lopputilaansa, palautuu tieto tästä **CentralizedMonitoringSessionBean**-pavulle. Merkiksi rooliepookin loppumisesta papu merkitsee roolitason yksilöpapuun roolin deaktivoiduksi. Tämän jälkeen papu tarkastaa ovatko kaikki vallitsevalla ajan hetkellä aktiiviseen epookkiin liittyvät oman organisaation roolit deaktivoituja. Jos ovat, tiedottaa papu

asiasta verkostoagenttia **NetworkManagementAgentClient** Web-palveluasiakasta käyttäen. Verkostoagentti synkronoi liiketoimintaverkoston osapuolten välisen epookin vaihdon ja ilmoittaa tämän jälkeen uudesta aktiivisesta epookista **CentralizedMonitoring**-rajapintaa käyttäen.

Kun jokin tietty rooli on deaktivoitu vastataan kaikkiin siihen liittyviin paikallisen valvontakerroksen pyyntöihin ilmoittamalla epookin vaihdon olevan kesken roolin osalta. Paikallinen valvontakerros tyypillisesti pollaa tämän jälkeen jonkin aikaa keskitettyä valvontakerrosta ja jos se havaitsee epookin vaihdon kestävän “liian kauan” se ilmoittaa tästä palvelukutsun tehneelle sovellukselle **EpochChangeException**-poikkeuksella. Jos paikallinen valvontakerros ei toimi proaktiivisesti ei tieto koskaan välity sovellukselle. Näin ollen automatisoitu epookin vaihto vaatii yleisessä tapauksessa proaktiivisen valvontamoodin käyttöä.

Epookkia vaihdetaan tyypillisesti lineaarisesti, eli seuraavaksi suoritettava epookki on järjestyksessään edellistä seuraava. Joissain tilanteissa saatetaan suorittaa uudelleen sama epookki kuin aiemmin tai vaihtoehtoisesti hypätä jonkin epookin yli. Tämänkaltaiset päätökset tehdään kuitenkin verkostonhallintapalveluiden toimesta, eikä niihin enempää tässä puututa.

5.5 Valvontakoneiston testitapaukset suorituskykytestausta varten

Valvontakoneiston prototyyppi sisältää viisi erilaista sen sisäiseen toimintaan vaikuttavaa parametria. Jokainen parametri voi saada totuusarvon (true/false) eli määritellä onko joku toiminto käytössä vai ei. Nuo viisi parametria on esitelty alla.

Suoritusaikainen valvonta käytössä / pois käytöstä määrittelee tehdäänkö suoritusaikana proaktiivista tai aktiivista valvontaa lainkaan.

Proaktiivinen valvonta käytössä / pois käytöstä määrittelee onko suoritusaikaisen valvonnan toimintatapa proaktiivinen vai aktiivinen.

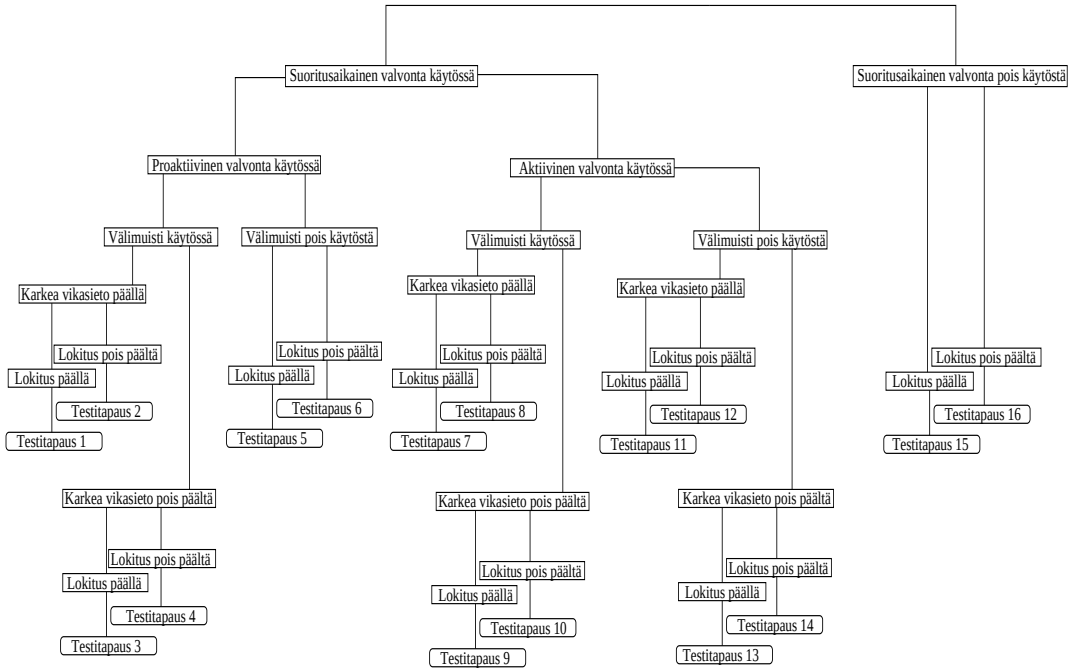
Paikallisen valvonnan välimuisti käytössä / pois käytöstä määrittelee pitääkö paikallinen valvonta yllä omaa välimuistia koreografiametatiedolle.

Karkea vikasieto päällä / pois päältä määrittelee tukeeko paikallinen valvonta heuristista/karkeaa vikasietoisuutta. Karkean vikasiedon ollessa päällä koreografiatason tila-automaattien tilaa pidetään tallessa persistentissä muodossa keskitetyn valvontakoneiston puolella. Karkea vikasieto on automaattisesti päällä ilman välimuistia toimivassa paikallisessa valvontayksikössä. Proaktiivisen valvonnan ollessa pois päältä karkea vikasietomekanismi pitää lisäksi paikallisessa valvontayksikössä persistentissä muodossa tallessa sovelluksen suorittamia vielä käsittelemättömiä valvottavia viestejä. Karkea vikasieto mahdollistaa valvontajärjestelmän toipumisen, jos valvottavat koneet kaatuvat kesken

suorituksen (useissa tapauksissa). Vikasiedon karkeus tarkoittaa sitä, että persistenttiysmekanismit eivät ole luonteeltaan atomisia. Persistenttiysmekanismit suorittavat peräkkäin ei-atomisia varmuustoimenpiteitä. Jos kone kaatuu sopivassa kohdassa kesken näiden toimenpiteiden suorituksen voi toipuminen saattaa järjestelmän epäkoherenttiin tilaan (järjestelmä toipuu vain osittain). Vikasietomekanismit nykymuodossaan ovat liian epäluotettavia teollisuustason vikasietoisuuden toteuttamiseen. Niiden merkitys on lähinnä siinä, että niiden avulla voi karkeasti arvioida vikasietoisuuden lisäämisen aiheuttamien suorituskykyhaittojen suuruusluokkaa.

Lokitus päällä / pois päältä määrittelee kirjoittaako paikallinen valvontayksikkö omaa lokia vai ei. Suoritusaikaisen valvonnan ollessa pois päältä on lokin kirjoittaminen ainoa valvontamekanismi ja mahdollistaa viivästetyn valvonnan. Paikallista lokia voi kirjoittaa myös silloin, kun suoritusaikainen valvontamekanismi (aktiivinen tai proaktiivinen) on käytössä.

Eri parametrien kombinaatioista voidaan johtaa 16 **testitapausta** valvontajärjestelmän suorituskyvyn testaamiseksi eri käyttöskenaarioissa. On syytä ottaa huomioon, että testitapauksia ei ole suunnattu valvonnan tasojen **yksikkötestauksen** tai **integraatiotestauksen** tarpeisiin vaan nimenomaan suorituskykytestauksen testitapauksiksi. Yksikkö- ja erityisesti integraatiotestausta tehdessä pitää testaus toki suorittaa erikseen kaikille suorituskykytestitapauksille. Suorituskykytestitapaukset antavat näin pohjan muulle testaukselle, mutta eivät ole riittävä testijoukko.



Kuva 5.6: Valvonnan suorituskykytestitapaukset.

Kuvassa 5.6 on eri testitapaukset ja niihin liittyvät testiparametrien arvot esitetty puumaisessa muodossa. Kuvasta voidaan myös huomata, että puu ei ole tasaisesti muodos-

tunut. Tasaisesti muodostunut puu sisältäisi **2*2*2*2*2** lehteä ja sitä kautta **32** eri testitapausta. Syy siihen, miksi puu on epätasapainoinen ja miksi testitapauksia on vain **16** erilaista johtuu siitä, että testiparametrit eivät ole keskenään riippumattomia. Joidenkin testiparametrien arvoilla ei ole merkitystä kuin silloin, kun jotkin muut testiparametrit on asetettu tietyllä tavoin.

Kuvassa 5.6 olevan kuvan ylimmällä tasolla puu haarautuu kahteen osaan sen mukaan, onko suoritusaikainen valvonta käytössä vai ei. Jos suoritusaikainen valvonta ei ole käytössä ei muilla kuin lokituksen määrittelevällä testiparametrilla ole merkitystä. Puun oikeassa laidassa onkin tämän takia vain kaksi lehteä: testitapaukset 15 ja 16. Testitapaus 16, jossa sekä suoritusaikainen valvonta että lokitus ovat pois päältä, on niin kutsuttu perustestitapaus (baseline). Perustestitapauksessa valvontakoneisto on kokonaisuudessaan kytketty pois päältä. Perustestitapaus antaa mittaustuloksen siitä, paljonko mitattavassa sovellusskenaariossa kuluu aikaa varsinaisen sovelluslogiikan suorittamiseen ja sovellusten väliseen viestiliikenteeseen. Kaikkien muiden testitapauksen mittaustuloksia verrataan tähän perustulokseen, jolloin nähdään kuinka paljon lisäkuormaa valvonnasta aiheutuu kussakin viidestätoista muusta skenaariosta.

Testitapaus 15 puolestaan kuvaa tilannetta, jossa suoritusaikainen valvonta on poissa päältä, mutta valvontakoneisto kirjoittaa koreografiatasolla lokia, jonka perusteella valvontarutiinit voitaisiin myöhemmässä vaiheessa ajaa. Kyseessä on siis viivästettyä valvonnan tasoa vastaava suorituskykytestitapaus.

Testitapaukset 1-14 liittyvät tilanteisiin, jossa suoritusaikainen valvonta on käytössä joko proaktiivisessa tai aktiivisessa muodossa. Sekä proaktiivisessa että aktiivisessa tapauksessa voidaan erikseen säätää paikallisen valvonnan välimuistin käyttöön, karkean vikasiedon käyttöön ja lokituksen käyttöön liittyviä parametreja. Kuitenkin huomattavaa on, että proaktiivisen valvonnan tapauksessa ja välimuistin ollessa pois käytöstä ei karkean vikasiedon määrittelevällä parametrilla ole vaikutusta järjestelmän toimintaan. Tämä johtuu siitä, että karkea vikasieto ei proaktiivisen valvonnan tapauksessa vaikuta muuhun kuin siihen, talletetaanko välimuistissa olevan koreografian muuttunut tila persistentisti vai ei.

Luku 6

Yhteenveto ja jatkokehityskohteet

Sähköisten liiketoimintaverkostojen muodostamiseen ja hallintaan liittyy monia hajautetun tietojenkäsittelyn peruskysymyksiä. Kuinka osapuolet löytävät toisensa? Kuinka osapuolet saadaan ymmärtämään toisiaan niin teknisesti kuin semanttisestikin? Kuinka taataan toiminnan oikeellisuus ja turvallisuus?

Web-Pilarcos projektissa lähestytään sähköisiin liiketoimintaverkostoihin liittyviä kysymyksiä useista eri näkökulmista ja kehitetään väliohjelmistopalveluja löydettyjen ongelmien ratkaisemiseksi. Tämän tutkielman tarkoituksena on analysoida sähköisen liiketoimintaverkoston valvontaan liittyviä kysymyksiä ja kehittää niihin ratkaisuja. Erityisenä painopisteenä on sähköisen liiketoimintaverkoston tilan käsite. Mitä sähköisen liiketoimintaverkoston tila tarkoittaa? Mitä eri abstraktiotasoja tilaan voidaan liittää? Miten tilaa voidaan seurata ja valvoa sen kullakin eri tasolla?

Tutkielmassa esitettiin malli sähköisen liiketoimintaverkoston tilan abstraktiotasoiksi. Malli jakaa liiketoimintaverkoston tilan kuuteen eri abstraktiotasoon. Nuo tasot ovat abstraktista konkreettiseen suuntaan sopimustaso, (sopimus)sessiotaso, roolitaso, tehtävämallitaso, koreografiataso ja sovellustaso. Väliohjelmiston tehtävänä on hallita viiden ylimmän abstraktiotason tilaa.

Sopimustasolla valvotaan koko liiketoimintaverkoston tilaa. Sopimustasolla tila tarkoittaa sopimuksen itsensä tilaa sekä tietoutta siitä, mitä sopimussessioita sopimukseen liittyy. Sopimustasoa ei käsitellä tässä tutkielmassa, vaan se muodostaa oman kokonaisuutensa Web-Pilarcos projektissa.

Sessiotasolla tila tarkoittaa yhden sopimuksen puitteissa käynnistetyn sopimussession suoritusaikaista tilaa. Sopimussessio perii alkutilansa sopimukselta, mutta omaa tämän jälkeen oma suoritusaikaisen tilansa. Sopimusessio voidaankin nähdä olio-ajattelun näkökulmasta tietyn sopimuksen määrittelemän luokan instanssina. Sessiotason tilaan sisältyy tieto siitä, missä epookissa sessio on kulloinkin menossa. Sessiotason tilaan sisältyy lisäksi siihen liittyvien roolitasojen tila.

Roolitasolla tila tarkoittaa session tiettyyn epookkiin liittyvän roolin tilaa. Roolit liittyvät epookkeihin erityisten yhteisöarkkitehtuurien kautta. Jokaiseen epookkiin liittyy yksi yhteisöarkkitehtuuri, joka kuvaa epookkiin liittyvän yhteisön. Valvonnan näkökulmasta yhteisö koostuu rooleista ja niiden välisistä keskusteluyhteyksistä. Roolitasolla epookki näkyy erityisenä rooliepookkina. Rooliepookkiin liittyy tehtävämalli ja koreografia, joilla

molemmilla on oma tilansa. Sessiotason epookkiin liittyy siis yhteisöarkkitehtuurin kautta joukko rooliepookkeja, joita valvotaan roolitasolla.

Tehtävämallitasolla tila tarkoittaa tietyn rooliepookin elinkaaren vaihetta. Rooliepookin elinkaareen liittyy vähintään kaksi tilaa: alkutila ja lopputila. Rooliepookki voi myös olla monimutkaisempi jolloin sen alku- ja lopputilan välissä voi olla yksi tai useampi välitila. Näin syntyvä tehtävämalli muodostaa tila-automaatin, jolla on alkutila, lopputila ja nolla tai useampi välitilaa. Tehtävämallitason tila-automaatin rakenne on ilmaistu liiketoimintaverkoston välisessä sopimuksessa joko suoraan tai epäsuorasti jonkin mallinnuskielen keinoin. Jälkimmäisessä tapauksessa tehtävämallin tila-automaatti voidaan tuottaa mallinnuskielen pohjalta. Tehtävämallitaso ylläpitää tietoutta tila-automaatin nykytilasta sekä suoritusjäljestä. Suoritusjälki koostuu tilasiirtymien tapahtumisajankohtien aikaleimoista, sekä rooliepookin suorituksessa tapahtuneista virhetilanteista ja niiden aikaleimoista. Tila-automaatin tilasiirtymien aktivoituminen sekä virhetilanteiden havaitseminen tapahtuvat koreografiatason toimesta.

Koreografiatasolla tila tarkoittaa tiettyyn rooliepookkiin liittyvää ulkoisen viestinvaihdon tilaa. Rooliepookkiin liittyvä koreografia muodostaa käytännössä tila-automaatin, joka sisältää kaikki mahdolliset viestinvaihdon suorituspolut. Kuten tehtävämallin, myös koreografiatason tila-automaatin rakenne on ilmaistu liiketoimintaverkostojen välisessä sopimuksessa joko suoraan tai epäsuorasti (yhteisöarkkitehtuurien formaalissa käyttäytymiskuvauksessa). Koreografiatason tila-automaattiin sisältyy varsinaisen viestinvaihtokoreografian lisäksi viestinvaihdon sidonta tehtävämallitason tila-automaattiin. Koreografiatason tila-automaatin tietyt tilasiirtymät tuottavat niin sanotun tehtävävaikutuksen, joka kuljetetaan tehtävämallitasolle ja tulkitaan tehtävämallitason tila-automaatin siirtymänä.

Koreografiatasolla on viestinvaihdon tilan ylläpitämisen lisäksi käyttäytymisen dynaamiseen verifiointiin liittyvä vastuu. Yhteisöarkkitehtuurien formaaleissa käyttäytymiskuvauksissa on kuvattu yhteisöarkkitehtuurin toiminta. Kuvaukset on etukäteen oikeellisiksi verifioitu. Koreografiatason tehtävänä on valvoa, että suoritusaikaiset vuorovaikutukset tapahtuvat oikeelliseksi verifioidun käyttäytymiskuvauksen mukaisesti. Dynaamisen verifioinnin näkökulmasta koreografiatason valvontakoneisto voidaan asettaa toimimaan passiivisessa, aktiivisessa tai proaktiivisessa moodissa. Passiivisessa moodissa koreografiataso ainoastaan ylläpitää lokia viestinvaihdosta ja lokin perusteella voidaan myöhemmin suorittaa varsinaiset valvontarutiinit. Aktiivisessa moodissa koreografiataso toimii rinnakkain sovelluksen kanssa ja sovelluksen virheellinen viestinvaihto havaitaan suoritusaikaisesti (valvontajärjestelmä ei kuitenkaan puutu sovelluksen toimintaan). Proaktiivisessa moodissa jokainen sovelluksen lähettämä ja vastaanottama viesti verifioidaan synkronisesti sovelluksen toiminnan kanssa ja koreografisesti virheellisten viestien lähestys ja vastaanotto estetään.

Nykyisessä toteutusmallissa tila on replikoitu eri organisaatioihin eli jokainen organisaatio pitää yllä kaikkiin osapuoliin liittyvää tilaa. Tilaa voitaisiin pitää yllä myös yhdessä keskitetyssä paikassa, jolloin replikointia ei tarvittaisi. Web-Pilarcoksen lähestymistavan perusideana on kuitenkin aito vertaisverkko (peer-to-peer network), jossa ei erityistä kaikille yhteistä keskitettyä osapuolta käytetä muuten kuin palvelujen löytämisen yhteydessä.

Tilan ylläpitämisen lisäksi jokaisella väliohjelmistotasolla on erilaisia hallinnallisia

tehtäviä, joita on käsitelty tarkemmin tutkielman aiemmissa luvuissa ja ne voidaan yhteenvedossa pääosin sivuuttaa. Hallinnalliset tehtävät liittyvät ennenkaikkea tilatiedon välittämiseen abstraktiotasojen välillä ja lopulta organisaatioiden välillä. Lisäksi hallinnallisiin tehtäviin liittyy epookinvaihtotarpeen havaitseminen sekä sessio- että rooliepookkitasoilla ja itse epookin vaihtoon liittyvät toiminnot.

Väliohjelmistotasot on Web-Pilarcos prototyypissä toteutettu hajautetusti. Verkostonhallintapalvelut, jotka hallinnoivat sopimustason tilaa, ovat omassa hajautusyksikössään. Ne on usein kuitenkin fyysisesti sijoitettu keskitetyn valvonnan hajautusyksikön kanssa samaan paikkaan. Keskitetyn valvonnan hajautusyksikkö sisältää sessiotason, roolitason ja tehtävämallitason toiminnallisuuden. Keskitetyn valvonnan hajautusyksikkö tarjoaa Web-palvelurajapinnan sekä verkostonhallintapalveluille että paikallisen valvonnan hajautusyksiköllе. Paikallisen valvonnan hajautusyksikkö sisältää koreografiatason valvonta-algoritmit ja sen instanssit kommunikoivat keskitetyn valvonnan hajautusyksikön kanssa SOAP-kutsuilla.

Prototyypitetty valvontakoneisto sisältää neljä konfiguroitavaa koneiston toimintaan vaikuttavaa parametria. Parametreilla voidaan säädellä käytettyä valvontamoodia (proaktiivinen, aktiivinen passiivinen), paikallisen valvonnan välimuistia, vikasietoisuusominaisuuksia ja lokitusta. Parametreja voidaan säätää siten, että lopputuloksena on 16 järkevää testitapausta suorituskykytestausta varten. Yksi näistä testitapauksista on sellainen, jossa kaikki valvontatoiminnot on kytketty pois päältä. Tätä moodia voidaan käyttää suorituskykytestauksen perustapauksena (baseline), johon muiden moodien suorituskyvyllistä raskautta voidaan verrata.

Väliohjelmistoon kuuluvien abstraktiotasojen lisäksi liiketoimintaverkoston tilaa ylläpidetään sovelluksissa. Sovellukset sisältävät varsinaisen liiketoimintalogiikan ja näin ollen niissä ylläpidetään erityisesti verkoston liiketoiminnallista tilaa. Sovelluksia ja niihin liittyvää tilaa ei kuitenkaan käsitelty tässä tutkielmassa.

Eri abstraktiotasojen lisäksi voidaan puhua tasoriippumattomasta valvonnasta. Tasoriippumaton valvonta kuvaa sellaisia väliohjelmistollisia valvontatehtäviä, jotka eivät suoraan liity mihinkään liiketoimintaverkostoja kuvaavaan sopimukseen, vaan jotka liittyvät organisaation sisäisten ja sopimusriippumattomien politiikkojen valvontaan. Tasoriippumatonta valvontaa ei myöskään käsitelty tässä työssä.

Tutkimuksen jatkosuuntia ja jatkokehityskohteita voidaan nimetä useita. Ilmiselviä kohteita ovat tutkielmassa kokonaan tai lähes kokonaan sivuutetut asiat. Arkkitehturaalisesta näkökulmasta sivuutettuihin asioihin kuuluvat tasoriippumaton valvonta ja sovellustason valvonta. Myös sopimustason valvonta sivuutettiin, mutta sitä tutkitaan jo erikseen Web-Pilarcos projektissa. Asianäkökulmasta katsottuna sivuutettuja kohteita ovat politiikkojen valvontaan ja palvelun laadun (QoS) valvontaan liittyvät kysymykset. Nämä asiakokonaisuudet liittyvät osittain tutkielmassa käsiteltyihin ja osittain edellä mainittuihin, tutkielman ulkopuolelle jääviin, valvontakerroksiin.

Eräs mielenkiintoinen, mutta pragmaattisesti ajateltuna ehkä turha, jatkokehityskohde olisi semanttisen valvonnan nostaminen sovellustasolta väliohjelmistossa toteutuille valvonnan tasoille. Erityisesti tässä yhteydessä mielenkiintoinen on koreografioihin liittyvän sisältövirran semanttisen oikeellisuuden varmentaminen.

Suoraviivaisempia jatkokehityskohteita ovat olemassaolevien valvonnan tasojen edel-

leen kehittämiseen liittyvät aspektit. Erityisesti koreografiatasolle olisi kohtuullisella vaivalla tuotavissa uusia piirteitä. Koreografiatason tila-automaattien osittaminen keskusteluyhteyskohtaisiksi ja fyysisesti hajautetuiksi on esimerkki toteutusorientoituneesta laajennoskohteesta. Koreografiatason laajentaminen monenvälisiä keskusteluyhteyksiä tukevaksi on esimerkki käsitteellisestä laajennoskohteesta. Valvontamoodin (passiivinen, aktiivinen, proaktiivinen) sopimusperustainen valinta ja käyttöönotto on on esimerkki laajennoskohteesta, joka vaatisi lisäyksiä myös tämän työn ulkopuolelle jätettyyn sopimustason valvontakerrokseen. Avoimemman, ei suljettuihin koreografiakuvauksiin perustuvan, koreografiakerroksen rakentaminen on esimerkki, ei niinkään laajennoskohteesta, vaan potentiaalisesta muunnoskohteesta.

Hyvin toteutusläheisiä laajennoskohteita ovat vikasieto-ominaisuuksien parantaminen esimerkiksi replikoinnin ja hienojakoisten tallennuspisteiden (checkpoint) avulla. Paikallisen valvonnan toteutusyksikön välimuistiominaisuuksien edelleen kehitys lukeutuu myös potentiaalisiin toteutusläheisiin parannuksiin.

Käsitteellisten ja toiminnallisten muutosten ohella valvontakoneistolle olisi syytä tehdä suorituskykytestaus tutkielmassa esiteltyjä testitapauksia hyväksi käyttäen. Suorituskykytestauksen tulosten perusteella valvontakoneiston toteutusta voisi edelleen optimoida ja sen aiheuttamaa lisäkuormaa kvantitatiivisesti arvioida.

Kaikenkaikkiaan voidaan todeta automatisoidun valvonnan olevan aihe, jonka kattavaan läpikäymiseen vaadittaisiin paljon enemmän kuin yksi tutkielma. Tässä työssä on mahdollisuuksien rajoissa pyritty käsittelemään aihetta kattavasti kuitenkaan käytännönläheisyyttä unohtamatta. Työn ulkopuolelle on jätetty pohdinnat siitä, onko automatisoitu valvonta nykyisissä yrityssovelluksissa ja teknologisessa ympäristössä lainkaan järkevää. Konseptina se on kuitenkin mielenkiintoinen. Kysymys on loppujen lopuksi siitä milloin automatisoitua valvontaa on kustannustehokasta käyttää ja milloin on halvempaa antaa asioiden mennä pieleen ja korjata ne myöhemmin tarpeen mukaan. Tämän kysymyksen selvittämiseen vaadittaisiin kuitenkin tapauskohtaisia tutkimuksia (case study) automatisoidun valvonnan hyödyistä ja kustannuksista erilaisissa liiketoimintaskenaarioissa.

Kirjallisuutta

[1] COLE, J., DERRICK, J., MILOSEVIC, Z., AND RAYMOND, K. Policies in an Enterprise Specification. In *Policies for Distributed Systems and Networks* (2001), M. Sloman, J. Lobo, and E. Lupu, Eds., Lecture Notes in Computer Science, Springer-Verlag, pp. 1–17.

[2] DIMITRAKOS, T., DJORDJEVIC, I., MILOSEVIC, Z., JOSANG, A., AND PHILLIPS, C. Contract Performance Assessment for Secure and Dynamic Virtual Collaborations. In *EDOC'03* (Sept. 2003), IEEE Computer Society. 7th International Enterprise Distributed Object Computing Conference (EDOC 2003).

[3] FENSEL, D., HORROCKS, I., VAN HARMELEN, F., MCGUINNESS, D., AND PATEL-SCHNEIDER, P. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems 16*, 2 (Feb. 2001).

[4] GRUBER, T. What is an ontology? *Published on Web* (Dec. 2004). `http://www-ksl.stanford.edu/kst/what-is-an-ontology.html`.

[5] HAATAJA, J.-P., AND TERGUJEFF, R. Using BPEL4WS for Supply-Chain Integration - Experiences and Evaluation. Tech. Rep. C-2003-74, Department of Computer Science, University of Helsinki and VTT Information Technology, 2003.

[6] KUMARAN, S., AND NANDI, P. Conversational Support for Web Services: The next stage of Web services abstraction. *Published on Web* (Sept. 2002). `http://www-106.ibm.com/developerworks/webservices/library/ws-conver`.

[7] KUTVONEN, L. B2B middleware for managing process-aware ecommunities. Tech. Rep. C-2004-63, Department of Computer Science, University of Helsinki, 2004.

[8] KUTVONEN, L. Challenges for ODP-based infrastructure for managing dynamic B2B networks. In *Workshop on ODP for Enterprise Computing (WODPEC 2004)* (2004), A. Vallecillo, P. Linington, and B. Wood, Eds., pp. 57–64.

[9] KUTVONEN, L. Controlling dynamic eCommunities: Developing federated interoperability infrastructure. In *Workshop on Interoperability of Enterprise Systems (INTEREST 2004)* (2004).

[10] KUTVONEN, L. Managing collaboration and interoperability of dynamic business networks. Tech. Rep. C-2004-62, Department of Computer Science, University of Helsinki, 2004.

[11] KUTVONEN, L. Relating MDA and inter-enterprise collaboration management. In *Second European Workshop on Model Driven Architecture (MDA), EWMDA-2* (Sept. 2004), pp. 84–88. Published as technical report No. 17-04 in University of Kent.

[12] KUTVONEN, L. Using business network models in web-Pilarcos. In *CAISE'04 Workshops* (June 2004), J. Grundpenkis and M. Kirikova, Eds., Riga Technical University, pp. 284–287. Open INTEROP Workshop on Enterprise Modelling and Ontologies for Interoperability (EMOI-INTEROP 2004).

[13] KUTVONEN, L., RUOKOLAINEN, T., METSO, J., AND HAATAJA, J.-P. Interoperability middleware for federated enterprise applications in web-Pilarcos. In *INTEROP-ESA'05* (2005). To be published.

[14] KÄHKIPURO, P. J2EE vastaan .NET. *3S Sovelluskehitys Symposium* (Jan. 2004). Presentation.

[15] LININGTON, P., MILOSEVIC, Z., AND RAYMOND, K. Policies in communities: Extending the enterprise viewpoint. In *Proc. 2nd International Workshop on Enterprise Distributed Object Computing (EDOC'98), San Diego, USA* (November 1998), p. 11.

[16] LUPU, E., MILOSEVIC, Z., AND SLOMAN, M. Use of Roles and Policies for Specifying, Building and Managing Virtual Enterprises. In *RIDE'99 Workshop* (1999), IEEE Computer Society. 9th International Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises.

[17] MILNER, R. *Communicating and Mobile System : The Pi-Calculus*. May 1999.

[18] MILOSEVIC, Z., BERRY, A., BOND, A., AND RAYMOND, K. Supporting business contracts in open distributed systems. IEEE Computer Society. 2nd International Workshop on Services in Distributed and Networked Environments.

[19] MILOSEVIC, Z., JOSANG, A., AND PHILLIPS, C. Discretionary Enforcement of Electronic Contracts. In *EDOCt'02* (Sept. 2002), IEEE Computer Society. 6th International Enterprise Distributed Object Computing Conference (EDOC 2002).

[20] NATIS, Y. CIO Update: The Microsoft Application Platform and J2EE. *Gartner Publications* (Jan. 2004).

[21] PETERSON, L. *Petri Net Theory and The Modeling of Systems*. 1981.

[22] Business Process Execution Language for Web Services (BPEL4WS v.1.1). *Published on Web* (May 2003). `http://www-106.ibm.com/developerworks/library/ws-bpel`.

[23] Business Process Modeling Language (BPML). *Published on Web* (Mar. 2001). http://www.bpmi.org/bpml-spec.esp.

[24] Common Object Request Broker Architecture (CORBA). *Published on Web* (Dec. 2004). http://www.corba.org.

[25] ebXML. *Published on Web* (Dec. 2004). http://www.ebxml.org.

[26] ebXML Business Process Specification Schema. *Published on Web* (May 2001). http://www.ebxml.org/specs/ebBPSS.pdf.

[27] Enterprise Java Beans Technology. *Published on Web* (Dec. 2004). http://java.sun.com/products/ejb/.

[28] Extensible Markup Language (XML). *Published on Web* (Dec. 2004). http://www.w3.org/XML/.

[29] Java 2 Enterprise Edition. *Published on Web* (Dec. 2004). http://java.sun.com/j2ee.

[30] Java Remote Method Invocation Technology. *Published on Web* (Dec. 2004). http://java.sun.com/products/jdk/rmi/.

[31] Java Servlet Technology. *Published on Web* (Dec. 2004). http://java.sun.com/products/servlet/.

[32] Simple Object Access Protocol (SOAP v.1.2). *Published on Web* (2003). http://www.w3.org/TR/SOAP.

[33] Unified Modeling Language (UML). *Published on Web* (Dec. 2004). http://www.uml.org.

[34] Web Services Architecture. *Published on Web* (Feb. 2004). http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

[35] Web Services Choreography Description Language (WS-CDL v1.0). *Published on Web* (Dec. 2004). http://www.w3.org/TR/ws-cdl-10/.

[36] Web Service Definition Language (WSDL v.1.1). *Published on Web* (Mar. 2001). http://www.w3.org/TR/wsdl.

[37] Web Services Flow Language (WSFL v.1.0). *Published on Web* (May 2001). http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.

[38] Web Services Policy Framework (WS-PolicyFramework). *Published on Web* (May 2003). http://www-106.ibm.com/developerworks/library/ws-polfram/.

[39] Workflow Process Definition Interface - XML Process Definition Language (XPDL). *Published on Web* (Oct. 2002). http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf.

[40] XLANG: Web Services for Business Process Design. *Published on Web* (June 2001). http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm.

[41] Apache AXIS (v1.1). *Published on Web* (June 2003). http://ws.apache.org/axis.

[42] Apache Tomcat (v4.1). *Published on Web* (Jan. 2003). http://jakarta.apache.org/tomcat.

[43] JBOSS Application Server. *Published on Web* (Dec. 2004). http://www.jboss.org/.

[44] Open source workflow and BPM-market. *Published on Web* (Dec. 2004). http://jbpm.org/article.html.

[45] Pi-Calculus: Automata, State, Actions, and Interactions. *Published on Web* (Dec. 2004). http://www.ebpml.org/pi-calculus.htm.

[46] What is EDI? *Published on Web* (Dec. 2004). http://www.webopedia.com/TERM/E/EDI.html.

[47] OASIS Consortium. *Published on Web* (Dec. 2004). http://www.oasis-open.org.

[48] Object Management Group (OMG). *Published on Web* (Dec. 2004). http://www.omg.org.

[49] Ontology.org. *Published on Web* (Dec. 2004). http://www.ontology.org/.

[50] RosettaNet Consortium. *Published on Web* (Dec. 2004). http://www.rosettanet.org.

[51] Semantic Web. *Published on Web* (Dec. 2004). http://www.w3.org/2001/sw.

[52] The Foundation for Intelligent Physical Agents (FIPA). *Published on Web* (Dec. 2004). http://www.fipa.org.

[53] W3C - Web Ontology Working Group. *Published on Web* (Dec. 2004). http://www.w3.org/2001/sw/WebOnt/.

[54] Web-Pilarcos Project. *Published on Web* (Dec. 2004). http://www.cs.helsinki.fi/group/web-pil.

[55] Workflow Management Coalition (WFMC). *Published on Web* (Dec. 2004). http://www.wfmc.org.

[56] World Wide Web Consortium (W3C). *Published on Web* (Dec. 2004). `http://www.w3.org`.

[57] XML.org. *Published on Web* (Dec. 2004). `http://www.xml.org`.