hyväksymispäivä arvosana

arvostelija

Kanavatyyppien dynaamisen muuntelun tukeminen Apache ServiceMix -palveluväylässä

Tom Bertell

Helsinki 20.4.2013

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta – Fakultet – Faculty	Laitos – Institution – Department
Matemaattis-luonnontieteellinen tiedekunta	Tietojenkäsittelytieteen laitos

Tekijä – Författare – Author

Tom Bertell

Työn nimi – Arbetets titel – Title

Kanavatyyppien dynaamisen muuntelun tukeminen Apache ServiceMix -palveluväylässä

Oppiaine – Läroämne – Subject

Tietojenkäsittelytiede

Työn laji – Arbetets art – Level	Aika – Datum – Month and year	Sivumäärä – Sidoantal – Number of pages
Pro gradu -tutkielma	20.4.2013	62 sivua + 1 liitesivu

Tiivistelmä – Referat – Abstract

Nykyaikaiset ketterät liiketoimintamallit ovat tuoneet mukanaan uusia vaatimuksia järjestelmien dynaamisuudelle ja ajonaikaiselle mukautumiselle. Palvelukeskeinen arkkitehtuuriin (Service Oriented Architecture, SOA) pohjautuvilla järjestelmillä pyritään vastaamaan näihin haasteisiin. Palveluväylä on väliohjelmisto, joka tarjoaa työkalut ja ajonaikaisen infrastruktuurin palvelukeskeisen arkkitehtuurin toteuttamiselle. Kanavatyypit ovat ajonaikaisten kanavien alustariippumattomia kuvauksia, jotka määrittelevät palvelukeskeisen arkkitehtuurin palveluiden välisen sidoksen.

Palveluiden välisen sidoksen elinkaaren aikana voi tapahtua ympäristön ja vaatimusten muutoksia, joihin kanavatyypin pitää pystyä reagoimaan. Perinteisesti palveluväylien tarjoamien kanavatyyppien rajapinnat ja ominaisuudet asetetaan pysyvästi rakentamisvaiheessa ja ne tarjoavat hyvin rajalliset mahdollisuudet ominaisuuksien valinnalle sen jälkeen kun kanavatyypit on asennettu ajoympäristöön. Ongelman ratkaisuna työssä suunniteltiin ja toteutettiin OpenChannel -kehys. OpenChannel -kehys toteuttaa rajapinnan, jonka kautta päästään käsiksi kanavatyyppien sisäiseen rakenteeseen ja sitä kautta kehys mahdollistaa kanavatyyppien dynaamisen muuntelun ja elinkaaren hallinnan.

OpenChannel -kehys käyttää reflektiota kanavatyyppien rakenteen tutkimiseen ja se hyödyntää mallipohjaista lähestymistapaa muunneltavuuden hallintaan. Kehyksen kohdealustana toimii Apache ServiceMix -palveluväylä, jonka käyttämä ajoympäristö mahdollistaa komponenttien lisäämisen, muokkaamisen ja poistamisen ajonaikaisesti ja tekee siitä siten kehykselle sopivan kohdealustan. Kehystä arvioitiin toteuttamalla dynaamista muuntelua tukeva kanavatyyppi, jota ajettiin oikeaa käyttöympäristöä simuloivassa testiympäristössä. Tulosten perusteella kehys soveltuu hyvin käytettäväksi kaikissa ympäristöissä, joissa suorituskykyvaatimukset eivät ole erityisen korkeat.

ACM Computing Classification System 2012 (CCS):
Software and its engineering → Reflective middleware
Applied computing → Service-oriented architectures
Software and its engineering → Model-driven software engineering

Avainsanat – Nyckelord – Keywords

SOA, palveluväylä, reflektiivinen väliohjelmisto, MDE

Säilytyspaikka – Förvaringställe – Where deposited

Muita tietoja – Övriga uppgifter – Additional information

Sisältö

Liite 1. OpenChannel -kehyksen toteutus

1 Johdanto

Nykyään organisaatioiden liiketoimintaprosesseilta vaaditaan kykyä kehittyä ja reagoida nopeasti, jotta ne voivat vastata liiketoiminnan tarpeisiin [Pap03]. Tukeakseen liiketoimintaprosessien nopeaa muutosta organisaatioiden järjestelmäarkkitehtuureilta vaaditaan avoimuutta, laajennettavuutta ja mukautuvuutta. Palvelukeskeinen arkkitehtuuri on viimeisen kymmenen vuoden aikana kehittynyt järjestelmien suunnittelu- ja toteutusmalli, joka on syntynyt tukemaan liiketoiminnan nopeasti muuttuvia tarpeita. Palvelut ovat itsenäisiä, alustariippumattomia ja itsensä kuvaavia ohjelmistokomponentteja, jotka tukevat nopeaa ja helppoa hajautettujen sovellusten rakentamista [Pap03]. Palvelut voivat olla yksinkertaisia kuten tilin saldon tarkistus tai ne voivat olla monimutkaisia useista palveluista koostettuja liiketoimintaprosesseja kuten vakuutuskorvaushakemuksen käsittely.

Oleellisena osana nykyaikaisia palvelukeskeisiä arkkitehtuureja toimii palveluväylä. Palveluväylä on väliohjelmisto, joka tarjoaa työkalut ja ajonaikaisen infrastruktuurin palvelukeskeisen arkkitehtuurin toteuttamiselle [SHL05]. Kanavatyypit ovat ajonaikaisten kanavien alustariippumattomia kuvauksia, jotka määrittelevät palvelukeskeisen arkkitehtuurin palveluiden välisen sidoksen ja kokoavat palveluväylän toiminnallisuudet loogisiksi kokonaisuuksiksi. Kanavatyypin kuvaus pitää sisällään tiedon palvelun laatuvaatimuksista, monitoroinnista, virhetilanteiden käsittelystä ja tietoturvaominaisuuksista.

Perinteisesti palveluväylien tarjoamien kanavatyyppien rajapinnat ja ominaisuudet asetetaan pysyvästi rakentamisvaiheessa ja ne tarjoavat hyvin rajalliset mahdollisuudet ominaisuuksien muokkaamiselle sen jälkeen kun kanavatyypit on asennettu ajoympäristöön. Vaatimusten muutokset, palveluiden päivitykset, uuden tyyppiset käyttäjät ja uudet säännökset ovat esimerkkejä muutoksista, jotka voivat aiheuttaa kanavatyyppien muutostarpeita, joihin edellisen kaltainen malli ei pysty vastaamaan. Jotta kanavatyyppien ominaisuuksia voitaisiin muokata dynaamisesti ajonaikana, pitäisi kanavatyyppien tarjota kuvaus sisäisestä rakenteestaan ja mekanismit rakenteen muokkaamiseen. Reflektiota käytetään dynaamisten ominaisuuksien lisäämiseen väliohjelmistoihin, jolloin niiden ominaisuuksiin ja käyttäytymiseen voidaan helposti ja

tehokkaasti vaikuttaa ajonaikaisesti [BBC05]. Samoja komponentin rakenteeseen ja toiminnallisuuteen vaikuttavia mekanismeja voidaan käyttää myös palveluväylän kanavatyyppien yhteydessä tuomaan niihin kaivattua joustavuutta ja dynaamisuutta, jotta ne kykenevät mukautumaan paremmin muuttuviin vaatimuksiin ja erilaisiin ajoympäristöihin. Muutoksien ja niiden aiheuttamien seurauksien suhteet voivat olla monimutkaisia. Muutoksien hallinnassa voidaan hyödyntää muunneltavuuden mallintamista, jota on käytetty menestyksellisesti tuotelinjojen yhteydessä [Bos04]. Yhdistämällä reflektiiviset väliohjelmistot ja muunneltavuuden hallinnan voidaan toteuttaa joustava väliohjelmisto, jota on helppo muokata ja jonka muutokset ovat turvallisia ja todennettavissa [BBF08].

Työn yhteydessä suunniteltiin ja toteutettiin yksinkertainen ja helppokäyttöinen OpenChannel -kehys. OpenChannel -kehys toteuttaa rajapinnan, jonka kautta päästään käsiksi kanavatyyppien sisäiseen rakenteeseen ja sitä kautta kehys mahdollistaa kanavatyyppien dynaamisen muuntelun. Kehys tukee vastaavia dynaamisen muuntelun mahdollistavia mekanismeja, joita on kuvattu OpenORB -väliohjelmiston [BAB01] ja Genie -työkalun [BeB09] yhteydessä. Kanavatyyppien muunneltavien ominaisuuksien esittämiseen käytetään mallipohjaista lähestymistapaa [BBD08]. Kehyksen kohdealustana toimii avoimeen lähdekoodiin perustuva Apache ServiceMix -palveluväylä. Apache ServiceMix -palveluväylän käyttämä ajoympäristö mahdollistaa komponenttien lisäämisen, muokkaamisen ja poistamisen ajonaikaisesti, joten se soveltuu hyvin dynaamisesti muunneltavien kanavatyyppien alustaksi. Jotta kehyksen soveltuvuus käyttötarkoitukseen pystyttiin todentamaan, toteutettiin konkreettisen ongelma-alueen ratkaiseva kanavatyyppi, jonka ominaisuuksia ja käyttäytymistä voidaan tutkia ja muokata ajonaikaisesti. Tutkielmassa arvioitiin millaisiin muutoksiin kehyksen avulla pystytään mukautumaan ja pyrittiin tuomaan esille mitä haasteita ja rajoituksia ratkaisulla on.

Luvussa 2 esitellään aluksi palvelukeskeisen järjestelmän yleiset piirteet ja toimintaperiaatteet, sekä selvitetään mikä rooli palveluväylällä on palvelukeskeisessä järjestelmässä. Tämän jälkeen käydään läpi minkälaisiin ajonaikaisiin muutoksiin palvelukeskeisen järjestelmän toivotaan pystyvän mukautumaan ja minkälaisia vaati-

muksia ajonaikainen mukautuminen asettaa palveluväylälle. Seuraavaksi kuvataan miten palveluväylän dynaamisesti muunneltavat kanavatyypit pyrkivät toteuttamaan dynaamisuuden tuomat vaatimukset. Lopuksi esitellään Apache ServiceMix -palveluväylän yhteyteen toteutettava OpenChannel -kehys, joka toteuttaa avoimen rajapinnan kanavatyyppien sisäiseen rakenteeseen ja sitä kautta mahdollisuuden ominaisuuksien valinnalle ja toiminnallisuuden ajonaikaiselle muokkaamiselle. Luvussa 3 käsitellään tarkemmalla tasolla minkälaisia mekanismeja dynaamisen muuntelun toteuttamiseen voidaan käyttää ja kuinka mallipohjaista lähestymistapaa voidaan hyödyntää dynaamisen muuntelun yhteydessä. Lisäksi tarkastellaan miten mekanismeja käytetään hyväksi OpenORB -väliohjelmiston ja Genie -työkalun yhteydessä. Luvun 4 alussa perustellaan mitkä Apache ServiceMix -palveluväylän ominaisuuksista tekevät siitä sopivan OpenChannel -kehyksen alustaksi. Tämän jälkeen käydään läpi OpenChannel -kehyksen rakennetta ja toiminnallisuutta. Luvussa 5 selvitetään miten OpenChannel -kehys käyttää reflektiota dynaamisen muuntelun toteuttamiseen ja kuinka se hyödyntää mallipohjaista lähestymistapaa. Seuraavaksi esitellään dynaamisesti muunneltavan kanavatyypin kehityksen vaiheet ja esimerkkinä käytetään suorituskykyvaatimuksiin mukautuvan kanavatyypin toteutusta. Lopuksi luvussa 6 arvioidaan kuinka hyvin kehys toteuttaa sille asetetut vaatimukset ajamalla suorituskykyvaatimuksiin mukautuvaa kanavatyyppiä oikeaa ajoympäristöä simuloivassa testiympäristössä.

2 Dynaamisuus palvelukeskeisessä arkkitehtuurissa

Palvelukeskeinen arkkitehtuuri on hajautettujen järjestelmien suunnittelu- ja toteutusmalli, joka on kehittynyt nykyaikaisten ketterien liiketoimintaprosessien tarpeisiin. Palvelukeskeisen arkkitehtuurin perusvaatimuksena on toteuttaa palveluiden löyhä kytkentä ja yhteentoimivuus eri alustojen ja teknologioiden välillä. Palveluväylä on väliohjelmisto, joka pyrkii toteuttamaan palvelukeskeisen arkkitehtuurin vaatimukset. Ongelmana palvelukeskeisen arkkitehtuurin vaatimusten toteuttamisessa nykyisissä palveluväyläratkaisuissa on liiallinen jäykkyys ja avoimuuden puute. Palvelukeskeisen arkkitehtuurin dynaamisuuden mahdollistamiseksi tarvittaisiin uusia mekanismeja, joilla palveluväylän tarjoamien kanavatyyppien ominaisuuksia ja toi-

minnallisuutta voitaisiin muokata ajonaikaisesti.

2.1 Palvelukeskeisen arkkitehtuurin yleiskuvaus

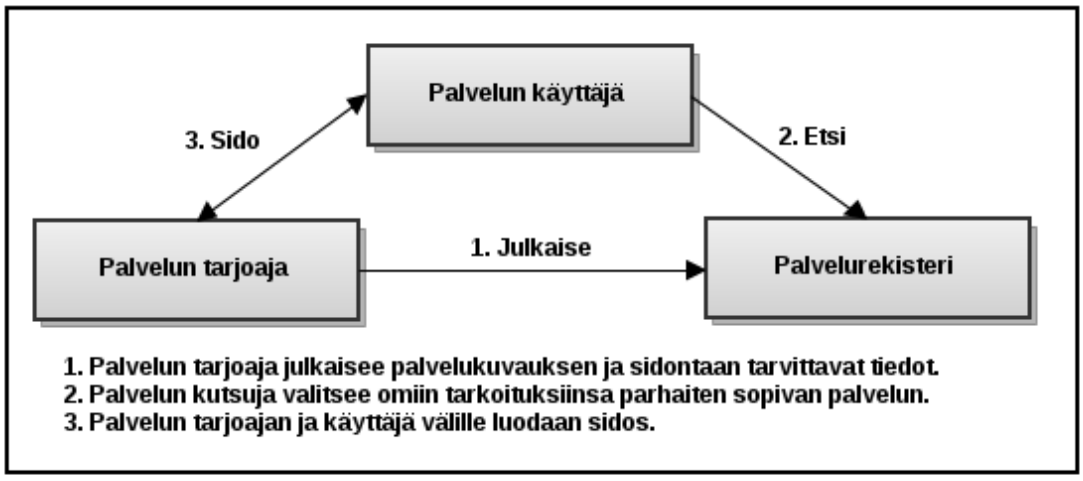
Organisaatioiden liiketoimintaprosesseilta vaaditaan kykyä kehittyä ja reagoida nopeasti, jotta ne voivat vastata liiketoiminnan tarpeisiin [Pap03, Liq12]. Tukeakseen liiketoimintaprosessien nopeaa muutosta vaaditaan organisaatioiden järjestelmäarkkitehtuureilta avoimuutta, laajennettavuutta ja mukautuvuutta. Palvelukeskeinen arkkitehtuuri on viimeisen kymmenen vuoden aikana kehittynyt järjestelmien suunnittelu- ja toteutusmalli, joka on syntynyt tukemaan liiketoiminnan nopeasti muuttuvia tarpeita.

Palvelut ovat itsenäisiä, alustariippumattomia ja itsensä kuvaavia ohjelmistokomponentteja, jotka tukevat nopeaa ja helppoa hajautettujen sovellusten rakentamista [Pap03]. Palvelut voivat olla yksinkertaisia kuten tilin saldon tarkistus tai ne voivat olla monimutkaisia useista palveluista koostettuja liiketoimintaprosesseja kuten vakuutuskorvaushakemuksen käsittely. Palveluiden avulla organisaatiot pystyvät tarjoamaan ydinosaamistaan laajalle käyttäjäkunnalle avoimen standardin rajapinnan ja standardeihin perustuvien teknologioiden avulla. Toimiakseen osana palvelukeskeistä arkkitehtuuria palvelulla tulee olla tiettyjä yhteentoimivuuden mahdollistavia ominaisuuksia. Palveluiden täytyy pystyä toimimaan keskenään saumattomasti riippumatta palveluiden varsinaisesta sijainnista, kommunikointiin käytetystä tietoliikenneprotokollasta tai palveluiden toteutusteknologiasta [Pap03].

Palveluiden löyhä kytkentä on välttämätön vaatimus palveluiden joustavalle yhteentoimivuudelle. Löyhä kytkentä mahdollistaa palveluiden korvaamisen ja muuttamiseen ilman, että muutokset heijastuvat muihin palveluihin tai järjestelmän osiin. Löyhä kytkentä toteutuu palvelun kutsujan ja palvelun dynaamisella sidonnalla. Sidos määrittelee tiedot, joita palvelun kutsuja tarvitsee palvelun kutsumiseen, kuten palvelun osoitteen, tietoliikenneprotokollan ja sanomaformaatin. Mitä myöhempään sidos määritellään sen paremmin vaatimusten ja ympäristön muutoksiin pystytään reagoimaan. Dynaamisella sidonnalla tarkoitetaan sitä, että palvelun kutsujan ja palvelun tarjoajan välinen sidonta muodostetaan viimeisellä mahdollisella hetkellä eli silloin kun palvelua kutsutaan.

Palvelun sijaintiriippumattomuuden vaatimuksena on se, että palvelukuvaus on tallennettu sellaiseen paikkaan, josta se on palvelun kutsujan löydettävissä. Palvelukuvaus pitää sisällään kaiken tiedon siitä mitä palvelun mahdollinen kutsuja tarvitsee palvelun kutsumiseen ja käyttämiseen. Tietoihin kuuluu tieto tuetuista tietoliikenneprotokollista, tietotyypeistä ja tietoturvavaatimuksista [Wee06]. Palveluun liitetään usein myös palvelun laatuun liittyviä ominaisuuksia, kuten palvelun käytön kustannukset, luotettavuus, saatavuus, skaalautuvuus ja suorituskyky.

Kuvassa 2.1 on kuvattu palvelukeskeiseen arkkitehtuuriin kuuluvat osapuolet ja niiden väliset vuorovaikutukset. Tyypillisessä tapauksessa palvelun tarjoaja tarjoaa palvelun verkon yli muiden käytettäväksi ja määrittelee palvelukuvauksen, joka julkaistaan palvelurekisterissä. Palvelun käyttäjä pyytää palvelurekisteristä hakukriteerien mukaista palvelua. Jos palvelurekisteri löytää hakukriteereihin sopivan palvelun, niin se palauttaa kutsujalle palvelukuvauksen ja tiedon palvelun osoitteesta. Näiden tietojen pohjalta palvelun käyttäjä luo palvelun käyttämisessä vaadittavan sidoksen [Pap03]. Uudet palvelut rekisteröidään palvelurekisteriin, josta palvelun käyttäjä voi löytää ne seuraavalla kutsukerralla.



Kuva 2.1: Palvelukeskeisen arkkitehtuurin osapuolet.

Palvelut itsessään voivat olla toteutettu millä teknologialla hyvänsä, mutta varmistaakseen yhteentoimivuuden, palvelut tulee olla julkaistavissa, löydettävissä ja käytettävissä avoimien standardien teknologioiden avulla. Palveluiden väliseen kom-

munikointiin käytetään yleisesti XML perustaista SOAP -protokollaa (Simple Object Access Protocol). Palvelut voidaan kuvata esimerkiksi käyttämällä WSDL -kuvauskieltä (Web Services Description Language) [CCM01]. Lisäksi palvelurekisteriä varten on kehitetty standardeja kuten UDDI (Universal Description, Discovery and Integration) [CHR04].

Tukeakseen palvelukeskeisen arkkitehtuurin melko monimutkaisia vaatimuksia täytyy infrastruktuurin olla monipuolinen ja joustava. Väliohjelmistoja on yleisesti käytetty tukemaan monimutkaisten heterogeenisten hajautettujen järjestelmien toteutusta [CGI07]. Palveluväylä on väliohjelmisto, joka tarjoaa työkalut ja ajonaikaisen infrastruktuurin palvelukeskeisen arkkitehtuurin toteuttamiselle [SHL05].

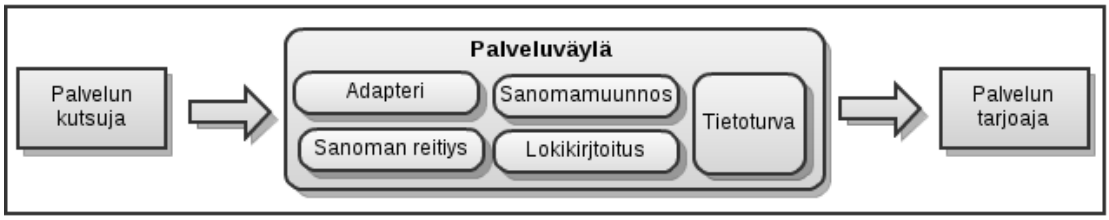
Ennen vuosituhannen alkupuolella tapahtunutta palveluihin perustuvan arkkitehtuurityylin yleistymistä hajautettujen järjestelmien toteuttamiseen käytettiin yleisesti ORB-pohjaisia väliohjelmistoja [Wee06]. Eniten käytettyjä ORB-pohjaisia väliohjelmistoja ovat CORBA (Common Object Request Broker Architecture), Java RMI (Remote Method Invocation) ja Microsoft DCOM (Distributed Component Object Model). Eri valmistajien väliohjelmistotototeutukset perustuvat erilaisiin oliomalleihin ja alustojen tarjoamat palvelut kuten transaktiot, tietoturva ja sanomanvälitys eivät ole keskenään yhteensopivia [Wee06]. Nämä eroavaisuudet tekevät eri alustoilla olevien palveluiden välisen yhteentoimivuuden haastavaksi ja jossain tilanteissa jopa mahdottomaksi, joten ne eivät sovellu kovin hyvin palvelukeskeisen arkkitehtuurin pohjaksi.

Palveluväylä on sanomapohjainen avoimiin standardeihin perustuva väliohjelmisto, joka toimii palveluiden välisenä integrointikerroksena (kuva 2.2). Palveluväylän tärkeimpiä toiminnallisuuksia ovat:

- turvallinen ja luotettava sanomanvälitys eli palveluväylän pitää tukea sanomien salausta, sekä palvelukutsujen todennusta ja pääsynhallintaa,
- sanomien reititys on palveluväylän kyky päättää sanoman kohdeosoite sanoman välityksen aikana,
- sanomamuunnokset lähdeformaatista sanoman vastaanottajan käyttämään formaattiin,
- protokollamuunnokset mahdollistavat eri teknologioilla toteutettujen palvelui-

den yhteentoimivuuden ja

- auditointi ja lokikirjoitus [Men07].



Kuva 2.2: Palveluväylän arkkitehtuuri

Käyttämällä hyväksi palveluväylän toiminnallisuuksia palvelun tarjoajan ja kutsujan ei tarvitse olla toteutettu samalla teknologialla, eikä niillä tarvitse olla yhteistä tietoliikenneprotokollaa tai edes sanomaformaattia. Keskeisenä konseptina ovat väylään toteutetut palvelun kutsujan ja palvelun tarjoajan välisen sidonnan toteuttavat kanavatyypit. Kanavatyyppien ominaisuudet vaihtelevat tuettavien tietoliikenneprotokollien, tietotyyppien, ajoympäristöjen ja ei-toiminnallisten vaatimusten osalta.

2.2 Dynaamisuuden tarve palvelukeskeisessä arkkitehtuurissa

Tarjotakseen järjestelmille niiden tarvitsemaa joustavuutta ja dynaamisuutta monia palvelukeskeisen arkkitehtuurin ominaisuuksia voi olla tarpeen muokata ajonaikaisesti [Clo08]. Muutostarpeet voivat johtua esimerkiksi ympäristön muutoksista, palvelutasosopimuksen muutoksista tai asiakkaan vaatimuksista. Lisähaasteena joissakin ympäristöissä muutoksiin mukautuminen pitäisi tapahtua ilman, että ajossa olevaa järjestelmää joudutaan sulkemaan muutoksien teon ajaksi.

Perinteisesti järjestelmään kohdistuvien muutosten hallinta on tehty järjestelmän rakentamisvaiheessa, joka saattaa toimia hyvin niin kauan kuin palveluiden tarpeet ja ympäristöt eivät muutu kovin usein. Useita palvelukeskeisen arkkitehtuuriin perustuvia järjestelmiä käytetään kriittisissä ympäristöissä, joissa esiintyvät käyttökatkot saattavat aiheuttaa organisaatiolle suuria rahallisia menetyksiä, joten muutokset ei-

vät saa aiheuttaa häiriöitä ajossa olevaan järjestelmään. Lisäksi monissa järjestelmissä pitäisi pystyä mukautumaan palvelun kutsujan tarpeisiin ja kontekstin muutoksiin. Palvelun kutsuja voi esimerkiksi haluta valita alemman turvatason ja saada sitä kautta paremman suorituskyvyn. Kontekstin muutokseen liittyvä muutostarve saattaa syntyä esimerkiksi, kun jokin palveluntarjoaja katoaa yllättäen. Toipuakseen järjestelmän pitää dynaamisesti korvata kadonnut palveluntarjoaja toisella palveluntarjoajalla, joka tarjoaa vastaavan toiminnallisuuden.

Taulukossa 2.1 on listattu palvelukeskeisen arkkitehtuurin ominaisuuksia, joita voi olla tarpeen muokata ajonaikaisesti ja esimerkkejä niiden käyttötarpeista. Näiden ominaisuuksien toteuttaminen monimutkaisiin hajautettuihin järjestelmiin, joita palvelukeskeiseen arkkitehtuuriin perustuvat järjestelmät pohjimmiltaan ovat, on erittäin haastavaa ja vaatii tukea järjestelmän alustana toimivalta palveluväyläratkaisulta.

Ominaisuus	**Esimerkki**
Tietoturva	Todennustavan vaihtaminen. (Saattaa vaihdella riippuen kutsujasta)
Suorituskyky	Suorituskyvyn monitorointi. Pakkausalgoritmin muuttaminen.
Palvelun valinta	Sopivan palvelun valinta esimerkiksi palvelun laadun (QoS) tai hinnan perusteella.
Poikkeustilanteiden käsittely	Virhe- ja poikkeustilanteiden käsittely ja virhetilanteista toipuminen.
Räätälöidyn liiketoimintalogiikan kutsuminen	Ennen palvelun kutsua tai kutsun jälkeen voidaan haluta suorittaa tilanteeseen räätälöityä liiketoimintalogiikkaa.
Versiointi	Jos palvelusta on tullut uusi versio, niin se voidaan haluta ottaa heti käyttöön tietyille asiakkaille.
Konfiguroinnin muuttaminen	Palveluiden konfiguroinnin muuttaminen ajonaikaisesti.
Monitorointi ja lokikirjoitus	Monitoroinnin ja lokikirjoituksen lisääminen suorituksen eri vaiheisiin.

Taulukko 2.1: Palvelukeskeisen arkkitehtuurin dynaamisia ominaisuuksia [BGR05].

Kaupallisessa käytössä olevat palveluväylät ovat tyypillisesti toteutettu niin, että ne piilottavat sisäisen rakenteensa käyttäjiltä. Ratkaisulle on monia hyviä syitä, kuten palveluväylän ylläpitäjän ja käyttäjän välinen luottamussuhde. Samalla ratkaisu kuitenkin rajoittaa palveluväylän avoimuutta ja sitä kautta muokattavuutta. Suuri osa palveluväylän toimintaa ja ominaisuuksia koskevista valinnoista ja asetuksista joudutaan suljetun rakenteen takia tekemään järjestelmän rakentamisvaiheessa tai ne vaativat palveluväylän uudelleen käynnistyksen tullakseen käyttöön.

Palveluväyläratkaisut sisältävät usein palveluväylän hallintaan tarkoitettuja työkaluja, mutta ne tarjoavat hyvin rajoittuneet mahdollisuuden toiminnallisuuden ajonaikaiseen muokkaamiseen. Ongelmana on myös se, että hallintatyökalut ovat vain palveluväylän omistavan osapuolen hallinnassa, eivätkä näin ole palveluväylää käyttävien tahojen käytettävissä. Dynaamisella muuntelulla tarkoitetaan kykyä tehdä ajonaikaisesti muutoksia komponentin rakenteeseen tai toiminnallisuuteen. Edellisessä luvussa esitettyjen järjestelmän ominaisuuksien dynaamisen muuntelun tukeminen palveluväylässä vaatii palveluväylän ominaisuuksien laajentamista monin tavoin. Palveluväylään tarvitaan mekanismeja, joilla päästään näkemään palveluväylän tarjoamien palveluiden sisäinen rakenne ja toiminnallisuudet. Lisäksi tarvitaan mekanismeja rakenteiden muokkaamiseen.

Dynaamisen muuntelun toteuttaminen tarjoamalla mekanismit järjestelmän sisäisen rakenteen muokkaamiselle tuo mukanaan paljon mahdollisuuksia dynaamisuuden toteuttamiseen, mutta samalla se tuo uusia haasteita ja sen tuomaa joustavuutta onkin usein tarvetta rajoittaa [Ven02]. Oleellisena osana rakenteen ja ominaisuuksien muutoksia tehtäessä on varmistaa, että järjestelmä toimii oikein ennen muutosta, muutoksen aikana ja muutoksen jälkeen. Tämä vaatii, että rakenteen ja toiminnallisuuden muutokset ovat kontrolloituja ja formaalisti määriteltyjä, jotta ne voidaan todentaa automaattisesti. Dynaamisen muuntelun mekanismeja käyttämällä pystytään häiritsemään järjestelmän toimintaa, joten mekanismeihin pääsyä pitää pystyä rajoittamaan mahdollisilta hyökkääjiltä [MSK04].

2.3 Kanavatyyppien dynaaminen muuntelu OpenChannel -kehyksen avulla

RM-ODP (Reference Model of Open Distributed Processing) on viitemalli, joka kuvaa arkkitehtuurin avoimien hajautettujen järjestelmien kehittämiselle. RM-ODP koostuu useista suosituksista ja standardeista, jotka yhdessä tarjoavat kehyksen avoimien hajautettujen järjestelmien suunnittelulle käyttäen hyväksi useita eri näkökulmia järjestelmään [LMT11]. Mallin mukaan kanava on sidoksen konkreettinen toteutus, joka mahdollistaa hajautettujen komponenttien välisen interaktion. Kanavatyyppi määrittelee kanavan toiminnallisuuden, joka pitää sisällään tiedon QoS-vaatimuksista, monitoroinnista, virhetilanteiden käsittelystä ja tietoturvaominaisuuksista.

Palveluväylien yhteydessä kanavatyypit kuvaavat palvelukeskeisen arkkitehtuurin palveluiden välisen sidoksen ja kokoavat palveluväylän toiminnallisuudet loogisiksi kokonaisuuksiksi. Jos kanavatyyppien ominaisuuksiin ja rakenteeseen pystyttäisiin vaikuttamaan ajonaikaisesti, niin se lisäisi oleellisesti palveluväylän joustavuutta, dynaamisuutta ja mukautuvuutta.

Muunneltavuuden mallintaminen (variability modeling) on tuotelinjojen (product line) kehityksessä käytetty menetelmä, jonka avulla määritellään miten tuotelinjan eri tuotteiden ominaisuudet eroavat toisistaan [Clo08]. Tuotelinja-ajattelun perustana on se, että tuotteella on yhteinen osa ja muunneltava osa, joka vaihtelee tuotekohtaisesti. Muunneltavuuden mallintamista käytetään lisäksi määrittelemään miten monimutkaiset järjestelmät mukautuvat erilaisiin ympäristön ja kontekstin muutoksiin [Clo08].

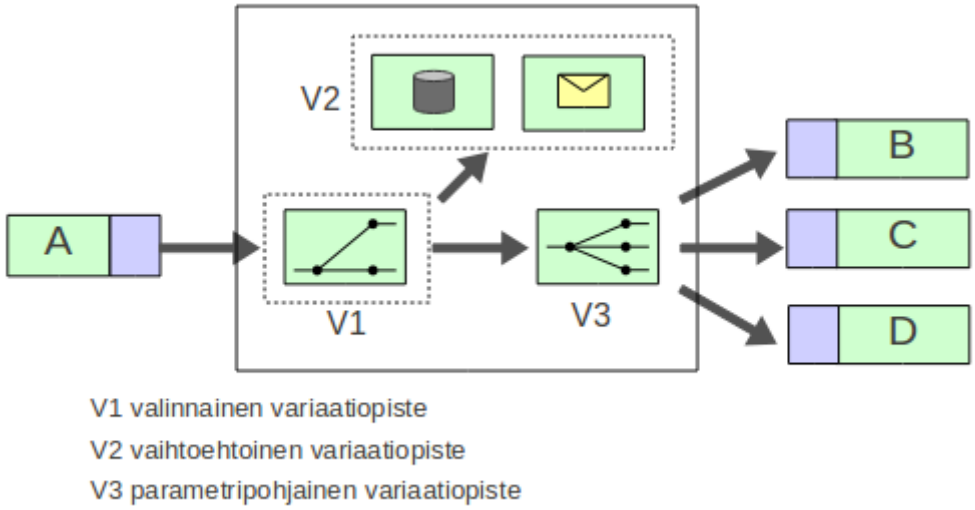
Muunneltavuuden mallintamisen perusperiaatteena on se, että ohjelmistokomponentille voidaan määritellä variaatiopisteitä eli kohtia joissa rakenteen tai ominaisuuksien muutokset ovat mahdollisia [Bos04]. Yksinkertainen variaatiopisteessä olevan parametrin muuntelu mahdollistaa melko rajoitetun mukautumisen. Sen sijaan variaatiopisteessä olevan komponentin rakenteeseen kohdistuva muuntelu mahdollistaa monipuolisemmat mahdollisuudet mukautumiselle.

Kanavatyypeistä voidaan myös tunnistaa kohtia, joissa sen rakennetta tai ominaisuuksia on tarpeen muunnella. Kanavatyypit voidaan siten rinnastaa tuotelinjoihin ja

kanavatyypistä johdettuja muunnoksia voidaan pitää tuotelinjan tuotteina, joten samoja muunneltavuuden mallintamisen menetelmiä, joita käytetään tuotelinjojen yhteydessä voidaan soveltaa myös kanavatyyppien muuntelun hallintaan. Kanavatyyppien yhteydessä voidaan tunnistaa kolme erilaista tapaa muuntelulle

- komponentti voi olla valinnainen,
- komponentin toteutuksella voi olla useita vaihtoehtoja tai
- komponenttiin liittyvää parametria voi muunnella.

Kuvassa 2.3 on esimerkki kanavatyypistä, jonka rakenteessa on kolme variaatiopistettä. Kanavatyypin tarkoituksena on välittää palvelun A lähettämä sanoma multicast-tyyppisesti usealle vastaanottajalle. Kanavatyypin variaatiopisteeseen V1 voidaan lisätä validointikomponentti, joka tarkistaa sanoman ja reitittää virheellisen sanoman virheenkäsittelykomponentille variaatiopisteeseen V2 ja tarkistuksen läpäisseen sanoman komponentille variaatiopisteessä V3. Variaatiopisteessä V2 voidaan valita kahdesta eri komponentista, joista toinen kirjoittaa virheen tietokantaan ja toinen lähettää käyttäjälle virheestä tiedon sähköpostilla. Variaatiopisteessä V3 on reitityskomponentti, joka lähettää sanoman parametrisoidulle listalle vastaanottajia.



Kuva 2.3: Esimerkki: kanavatyypin variaatiopisteet.

Palveluväylän alustalta ja ajoympäristöltä vaaditaan avoimuutta ja dynaamisuutta, jotta kanavatyyppien muuntelu olisi mahdollista toteuttaa. Käyttämällä hyväksi dynaamisesti muunneltavia kanavatyyppejä palveluväylä pystyy paremmin vastaamaan palvelukeskeisen arkkitehtuurin asettamia dynaamisuuden vaatimuksia.

Tukeakseen kanavatyyppien dynaamista muuntelua palveluväylän ajoympäristöltä vaaditaan näkyvyyttä kanavatyyppien ominaisuuksiin ja rakenteeseen ja kykyä muokata niitä. Jos palveluväyläratkaisun arkkitehtuuri ei ole tarpeeksi avoin, niin palveluväylän avoimuutta voidaan lisätä toteuttamalla sen yhteyteen tarvittavan toiminnallisuuden tarjoava kehys. Kanavatyyppien dynaamisen muuntelun mahdollistavan kehyksen tulisi täyttää seuraavat vaatimukset:

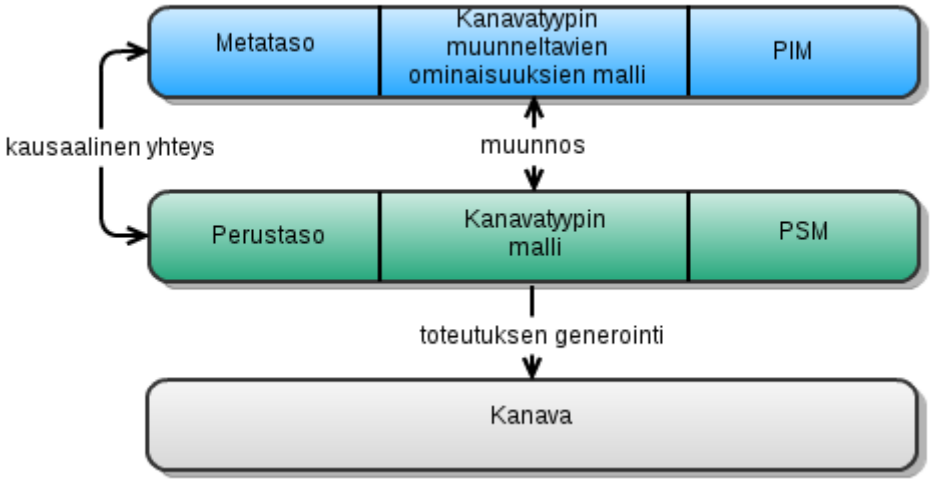
- käyttää abstrakteja kanavatyypin muunneltavista ominaisuuksista johdettuja malleja dynaamisen muuntelun toteutukseen (V1),
- kykenee vastaanottamaan kontekstin tai ympäristön muutoksista tapahtumia ja reagoimaan niihin (V2),
- tarjota avoin ja turvallinen rajapinta, jonka kautta kanavatyypin muunneltavien ominaisuuksien malli on löydettävissä ja muokattavissa (V3),
- toteuttaa luotettava synkronointimekanismi, jonka avulla mallin muutokset heijastuvat kanavatyypin toteutukseen ja toiseen suuntaan (V4),
- joustava ja helppo käyttää (V5),
- palveluväyliä käytetään usein ympäristöissä, joissa on korkeat suorituskykyvaatimukset, joten dynaamisen muuntelun lisääminen ei saa vaikuttaa negatiivisesti järjestelmän suorituskykyyn (V6).

OpenChannel -kehyksen tavoitteena on toteuttaa mekanismit, joilla edellä mainitut toiminnalliset vaatimukset (V1 – V4) ja laadulliset vaatimukset V5 ja V6 täyttyvät. Kehyksen perustana toimii kanavatyypin mallista johdettu korkean abstraktiotason muunneltavien ominaisuuksien malli (variability model). Muunneltavien ominaisuuksien mallia muokkaamalla voidaan vaikuttaa käytössä olevien kanavatyyppien ominaisuuksiin ja rakenteeseen. Kehys pitää sisällään mekanismin, jonka avulla voidaan listata kaikki palveluväylässä olevat kanavatyypit ja niiden muunneltavien ominaisuuksien mallit.

Kehys hyödyntää ajonaikaisesti reflektiota [Mae87] kanavatyyppien sen hetkisen rakenteen ja toiminnallisuuden tutkimiseen. Reflektiolla tarkoitetaan ohjelmistokomponentin kykyä tutkia ominaisuuksiaan ja tehdä päätelmiä sen perusteella. Reflektion edellytyksenä on, että perustason komponentilla on käyttäytymistään ja rakennettaan kuvaava metatason malli, joka on kausaalisesti yhdistetty itse komponenttiin. Kausaalisella yhdistämisellä tarkoitetaan sitä, että metatason mallin muutokset heijastuvat suoraan komponentin toteutukseen ja sama pätee myös toiseen suuntaan, jos komponenttia muutetaan. Kehys on vastuussa siitä, että metatason mallit ovat kausaalisessa yhteydessä perustason komponentteihin.

Kehys käyttää malliperustaisen ohjelmistokehityksen (MDE) menetelmiä kanavatyyppien muunneltavien ominaisuuksien mallintamiseen. Malliperustaista ohjelmistokehitystä käytetään hallitsemaan monimutkaisuutta abstraktioiden ja eri abstraktiotasojen välisten transformaatioiden avulla ja sitä on käytetty myös onnistuneesti dynaamisen muunneltavuuden hallintaan [BBD08]. Mallisuuntautunut arkkitehtuuri (MDA) määrittelee kolme abstraktiotasoa: laskentariippumaton malli (Computation Independent Model, CIM), alustariippumaton malli (Platform Independent Model, PIM), alustariippuvainen malli (Platform Specific Model, PSM). OpenChannel -kehyksessä kanavatyypin malli vastaa MDA-terminologian alustariippuvaista mallia. Kanavatyypin malli toimii lähdemallina, jonka pohjalta muodostetaan mallitransformaation kautta alustariippumaton malli, jossa on kuvattu kanavatyypin muunneltavat ominaisuudet.

Kuvassa 2.4 on nähtävissä miten reflektio- ja MDA-käsitteet yhdistetään OpenChannel -kehyksen yhteydessä. Kanavatyyppien muunneltavien ominaisuuksien malli vastaa reflektion metatason mallia ja MDA:n alustariippumatonta mallia. Kanavatyypin malli vastaa reflektion perustason mallia ja MDA:n alustariippuvaista mallia. Metatason ja perustason välinen kausaalinen yhteys toteutuu mallien välisten mallitransformaatioiden avulla. Varsinaisen kanavan toteutuksen generoiminen kanavatyypin mallista on ajoalustan vastuulla.



Kuva 2.4: OpenChannel -kehyksen abstraktiotasot ja niihin liittyvät käsitteet.

Kehyksen kohdealustaksi valitun Apache ServiceMix palveluväylän arkkitehtuuri on komponenttipohjainen ja sen perustana on OSGi-pohjainen ajoympäristö. OSGi sisältää erityisen luokkien latausmekanismin, jolla komponentteja voidaan dynaamisesti lisätä ja poistaa ajonaikaisesti [OSG11]. Dynaamisen luokkien latausmekanismin ansiosta Apache ServiceMix soveltuu erityisen hyvin OpenChannel -kehyksen kohdealustaksi. Kehyksen suunnittelussa otetaan huomioon mahdollisuus käyttää sitä myös muiden palveluväyläalustojen yhteydessä.

3 Dynaamisen muuntelun mekanismit

Palveluväylässä olevien kanavatyyppien dynaamisen muuntelu edellyttää, että kanavatyypin toiminnallisuuden toteuttavilla kokonaisuuksilla on selkeästi määritelty rakenne ja lisäksi vaaditaan mekanismeja rakenteen tutkimiseen ja muokkaamiseen. Komponenttikehyksiä ja reflektiota käytetään dynaamisesti mukautuvien väliohjelmistoalustojen toteuttamiseen [BBC05]. Samoja mekanismeja voidaan hyödyntää myös komponenttipohjaisten palveluväylien yhteydessä.

3.1 Dynaaminen muuntelu

Dynaamisesti mukautuvaksi järjestelmäksi kutsutaan järjestelmää, joka pystyy mukautumaan ajonaikaisesti (dynaamisesti) kontekstin ja ympäristön muutoksiin [BBF08]. Dynaamisesti mukautuviin järjestelmiin liittyvää muuntelua kutsutaan dynaamiseksi muunteluksi. Dynaamisen muuntelun toteutukseen on ehdotettu monia erilaisia matalan tason mekanismeja. Ne ovat pääasiassa keskittyneet käytössä olevan ohjelmointikielen tarjoamiin menetelmiin. Väliohjelmistotasolla matalan tason mekanismit eivät kuitenkaan ole riittäviä, joten tarvitaan korkeamman abstraktiotason mekanismeja, joiden avulla voidaan hallita joukkoa komponentteja ja niiden välisiä yhteyksiä arkkitehtuurin tasolla [BBF08].

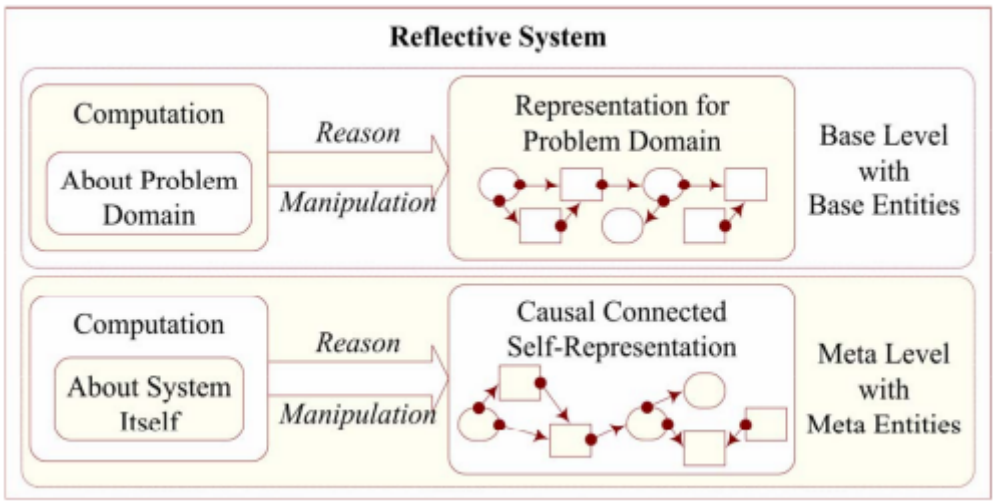
Reflektiiviset väliohjelmistot tarjoavat perustan dynaamisen muuntelun toteuttamiseen mukautuvissa järjestelmissä. Reflektiivisten väliohjelmistot rakentuvat tyypillisesti komponenttikehyksistä, joka mahdollistaa järjestelmän rakenteen ja toiminnallisuuden laajennettavuuden. Reflektiota käytetään ajonaikaiseen komponenttien rakenteen ja toiminnallisuuden tutkimiseen ja muokkaamiseen. Reflektion ja komponenttikehysten käyttäminen dynaamisen muuntelun toteuttamisessa saattaa johtaa komponenttien monimutkaisiin riippuvuussuhteisiin ja odottamattomiin muutoksiin, jonka takia muunneltavuuden hallintaan tarvitaan täsmällisiä menetelmiä [BBF08].

3.2 Reflektiivinen väliohjelmistomalli

Laskennallisella reflektiolla [Mae87] tarkoitetaan ohjelman tai järjestelmän kykyä tehdä päätelmiä itsestään ja muuttaa toimintaansa näiden päätelmien perusteella. Jotta järjestelmä pystyisi näin toimimaan, pitää sillä olla käyttäytymistään ja rakennettaan kuvaava metatason malli, joka on kausaalisesti yhdistetty itse järjestelmään. Kausaalisella yhdistämisellä tarkoitetaan sitä, että metatason mallin muutokset heijastuvat suoraan järjestelmään ja sama pätee myös toiseen suuntaan, jos järjestelmää muutetaan.

Samoja reflektiomekanismeja, joita käytetään ohjelmointikielissä, voidaan käyttää myös väliohjelmistoissa. Väliohjelmistojen tasolla reflektiolla tarkoitetaan sitä, että

järjestelmällä on kuvaus omasta sisäisestä toiminnasta ja rakenteestaan. Järjestelmän kuvausta itsestään kutsutaan metatasoksi ja järjestelmää itseään perustasoksi (kuva 3.1). Jotta järjestelmä olisi reflektiivinen, niin näiden kuvausten pitää olla kausaalisesti yhteydessä toisiinsa eli väliohjelmiston metatasolle tehtävät muutokset heijastuvat väliohjelmiston varsinaiseen toteutukseen perustasolle ja toisin päin jos toteutusta muutetaan, niin myös metatason malli muuttuu [BBC05].



Kuva 3.1: Reflektiivinen järjestelmä [HLM07].

Reflektiivinen järjestelmä tarjoaa metarajapinnan tai metaolioprotokollan (metaobject protocol, MOP), jonka kautta päästään käsiksi perustason rakenteisiin. Metaolioprotokollan ominaisuuksiin kuuluvat operaatiot alla olevan alustan tutkimiseksi ja sitä kautta alustan ominaisuuksien muuttamiseen. Tämä on selkeä ero perinteisiin väliohjelmistoihin verrattuna, jotka piilottavat toiminnallisuuden sisäänsä ja tarjoavat vain rajapinnat toiminnallisuuden käyttämiseen. Reflektiota käytetään väliohjelmistoissa tarjoamaan parempaa ajonaikaista konfiguroituvuutta ja dynaamista sopeutumista, jotka helpottavat väliohjelmiston päällä toimivien palveluiden yhteentoimivuutta.

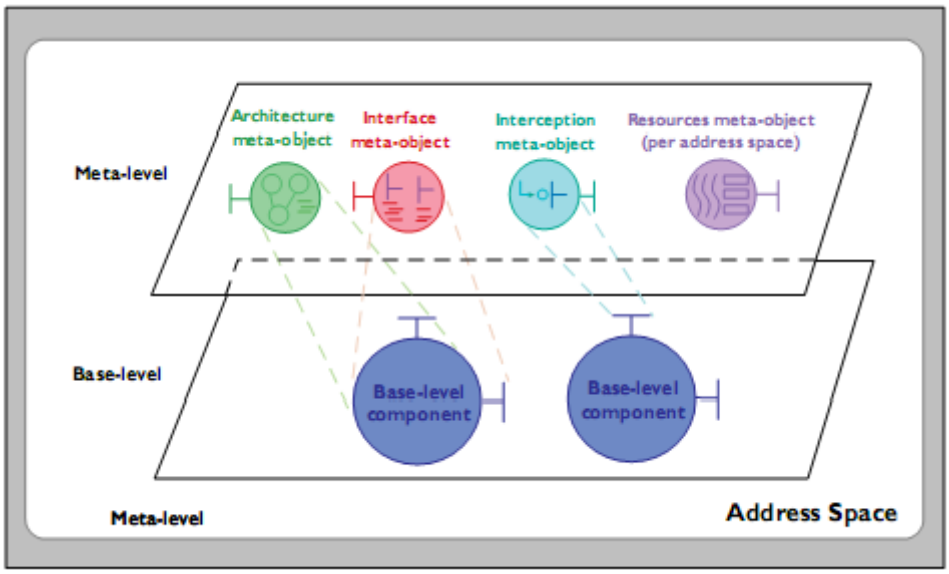
Perustasolla on väliohjelmistojen tarjoamat yleiset palvelut ja metataso tarjoaa mahdollisuuden palveluiden tutkimiseen (inspection) ja niiden toiminnan mukauttamiseen (adaptation) tarpeiden muuttuessa. Osat, joita järjestelmästä halutaan kuvata saattavat olla hyvinkin monimutkaisia, jolloin niiden kuvaaminen yhdellä metatason mallilla saattaa olla vaikeaa. Tästä syystä järjestelmän osat kuvataan usein useilla erillisillä

metatason malleilla, joista kukin tarjoaa näkymän johonkin tiettyyn järjestelmän osaan.

Reflektiiviset väliohjelmistot tarjoavat yleisesti kaksi erilaista reflektiotapaa: rakenteeseen kohdistuvan (structural) reflektion ja käyttäytymiseen kohdistuvan (behavioural) reflektion [GGL02]. Rakenteeseen kohdistuva reflektio mahdollistaa järjestelmän sisäisen rakenteen tutkimisen ja muuttamisen. Käytökseen kohdistuvan reflektion avulla voidaan puolestaan järjestelmän tai sen osien toiminnallisuutta tutkia ja muuttaa. Jos järjestelmän kaikkien osien rakennetta ja toimintaa pysyttäisiin vapaasti muuttamaan, niin järjestelmä menisi helposti tilaan, jossa se ei toimisi enää ollenkaan tai se toimisi väärin. Tästä syystä järjestelmästä kuvataan metatason malleilla vain ne osat, joihin halutaan tehdä muutoksia. Lisäksi metatason malleihin lisätään rajoitteita, jotka varmistavat, että järjestelmä toimii oikein myös dynaamisten muutosten jälkeen.

OpenORB -projekti [BAB01], on pyrkinyt suunnittelemaan muunneltavan väliohjelmiston, joka tukee useita hajautetuilta järjestelmiltä toivottuja dynaamisia ominaisuuksia [GGL02]. Väliohjelmiston perustoiminnallisuus koostuu komponenteista, joista voidaan koostaa sopivan toiminnallisuuden toteuttavia komponenttikehyksiä (component framework). Komponenttikehykset sisältävät rajoitteet, joita komponenttikehyksen sisäisen rakenteen pitää noudattaa. Komponenttikehyksillä on selkeästi määritellyt riippuvuudet muihin komponentteihin, jonka avulla niistä on yksinkertaista koostaa uusia rakenteita. Määriteltyjen sääntöjen ja riippuvuuksien avulla väliohjelmisto pystyy pitämään järjestelmän sallitussa tilassa dynaamisten muutosten yhteydessä.

Perustaso koostuu perinteisistä väliohjelmiston komponenteista, joiden toteutus paljastetaan metatason komponenteilla. Dynaamista konfigurointia varten järjestelmän jokaista perustason komponenttia kohti on vähintään yksi metatason komponentti. Jokaista perustason komponenttia kohden voi olla joukko metatason komponentteja, jota kutsutaan komponentin meta-avaruudeksi [BBC05]. Väliohjelmiston metatason malli on jaettu selkeiksi osiksi, jotka tarjoavat loogisen näkymän järjestelmän eri osien toteutukseen. Metatason mallit ovat arkkitehtuuri- (architecture), rajapinta- (interface), tarkkailu- (interception) ja resurssimetamalli (resource) (kuva 3.2).



Kuva 3.2: OpenORB abstraktiotasot ja metamallit [GGL02].

Rajapinta- ja arkkitehtuurimetamallit mahdollistavat rakenteellisen reflektion ja tekevät selkeän eron komponentin ulkoisen toiminnallisuuden (rajapinnat) ja sisäisen rakenteen välille. Rajapintametamalli paljastaa komponentin tarjoamat ja tarvitsemat rajapinnat operaatioineen. Metatason malliin liittyvä metaolioprotokolla mahdollistaa elementtien listaamisen ja etsimisen.

Arkkitehtuurimetamalli on keskittynyt nimensä mukaisesti kuvaamansa komponentin sisäiseen toteutukseen. Kuvaus jakaantuu kahteen osaan: komponenttikaavioon (component graph) ja arkkitehtuurisiin sääntöihin (architectural constraints), joista ensimmäinen kuvaa komponentin suhteen muihin komponentteihin ja toinen kuvaa säännöt joilla komponentteja voidaan yhdistellä. Komponenttikaavio on keskeinen osa mallia. Se kuvaa joukon komponentteja, jotka on yhdistetty toisiinsa paikallisilla sidoksilla (local bindings). Paikallinen sidos pitää sisällään kuvauksen komponentin rajapinnoista. Malliin liittyvä metaolioprotokolla mahdollistaa komponenttien lisäämisen, poistamisen ja korvaamisen ja sääntöjen muuttamisen. Tämä mahdollistaa dynaamisen mukautumisen. Perinteisissä väliohjelmistoissa tämän kaltainen toiminnallisuus olisi piilossa komponentin käyttäjältä, mutta arkkitehtuurimetamallin avulla järjestelmän rakennetta voidaan tutkia ja se pystyy mukautumaan ympäristön muutoksiin ajonaikaisesti.

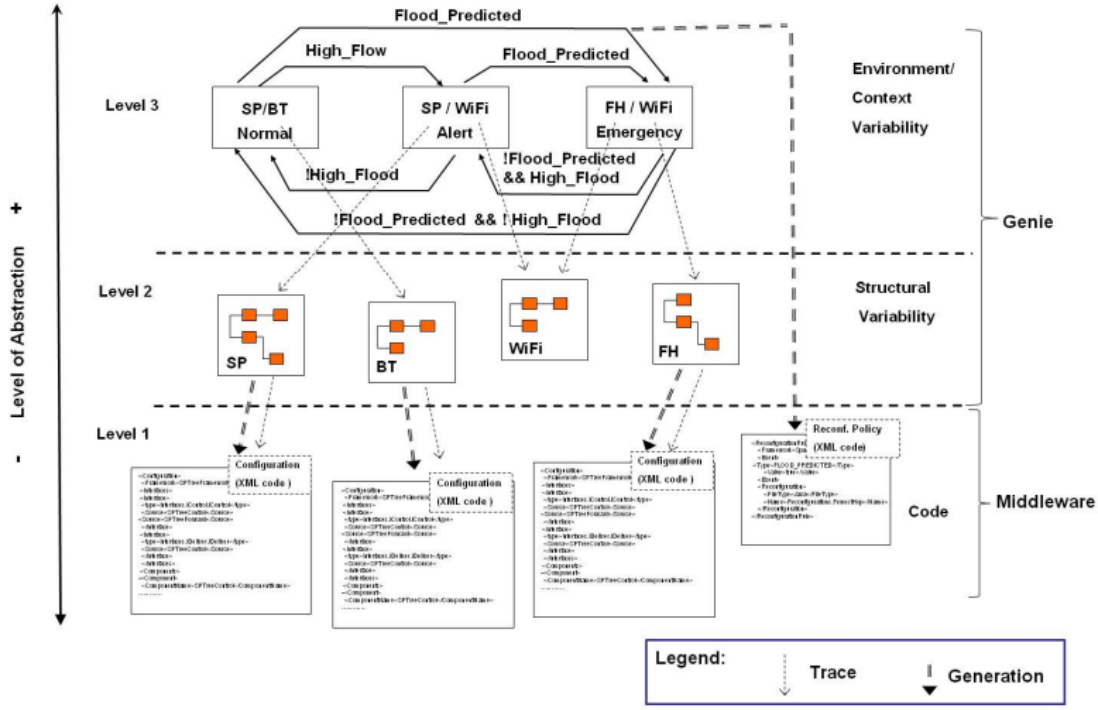
Open ORB -väliohjelmistossa tarkkailumetamalli tukee käyttäytymiseen kohdistuvaa reflektiota. Sen avulla järjestelmän komponenttien rajapintoihin voidaan lisätä dynaamisesti tarkkailijoita (interceptor). Tarkkailijoiden avulla on mahdollista suorittaa uutta toiminnallisuutta ennen komponenttien kutsuja ja kutsun jälkeen. Tämän mekanismin avulla järjestelmään voidaan lisätä ajonaikaisesti esimerkiksi monitorointi-, tietoturva- ja laskutuspalveluja [GGL02].

Resurssimetamallin (resource metamodel) avulla päästään suoraan käsiksi järjestelmän resurssien hallintaan ja sitä kautta resursseihin. Resurssimetamalli perustuu resurssien ja tehtävien abstraktioihin. Resurssit voivat olla primitiivisiä kuten muisti tai käyttöjärjestelmän säikeet tai monimutkaisia kuten puskureita. Resurssimetamallit luodaan resurssitehtaiden (resource factories) toimesta ja niitä hallitaan resurssimanagerien (resource managers) avulla.

3.3 Muunneltavuuden hallinta

Muuntelua kuvaavien variaatiopisteiden ja niiden välisten riippuvuussuhteiden lisääntyessä järjestelmän mahdollisten variaatioiden määrä kasvaa eksponentiaalisesti [BBD08]. Dynaamisen muunneltavuuden hallintaan on ehdotettu monia erilaisia matalan tason mekanismeja. Ne ovat pääasiassa keskittyneet ajonaikaisten ohjelmiston osien vaihtamiseen, parametrisointiin ja perintään [BBF08]. Viime aikoina on myös kehitetty korkeamman tason ratkaisuja, jotka käyttävät hyödykseen mallipohjaista lähestymistapaa [BeB09].

Genie on muunneltavuuden hallintatyökalu, joka tarjoaa mekanismit kehitysaikaiseen muunneltavuuden hallintaan ja mahdollistaa järjestelmällisen tavan generoida väliohjelmistoon liittyviä komponentteja korkean tason kuvauksista [BeB09]. Genie käyttää muunneltavuuden hallintaan mallipohjaista lähestymistapaa, jonka avulla kehittäjällä on käytössään tehokkaammat työkalut verrattuna matalan taso ohjelmointiin. Rakenteen ja kontekstin muutosten kuvaavien mallien rakentamiseen Genie käyttää toimialakohtaisia kieliä (Domain-specific language, DSL). Genie tukee kolmea abstraktiotasoa, jotka näkyvät kuvassa 3.3.



Kuva 3.3: Genien tukemat abstraktiotasot [BeB09]

Ylimmällä abstraktiotasolla 3 ovat siirtymäkaavioiden mallit (transition diagrams), jotka vastaavat ympäristön ja kontekstiin kohdistuvaan muunneltavuuteen. Tällä tasolla kehittäjän tehtävänä on määritellä eri rakenteelliset variantit (structural variants) ja minkälaiset ympäristön ja kontekstin muutokset laukaisevat uudelleenkonfiguroinnin. Jokainen siirtymäkaavion solmu vastaa yhtä rakenteellista varianttia. Uudelleenkonfiguroinnin laukaisevat säännöt on määritelty solmujen välisiin kaariin. Rakenteelliset variantit ovat joukko alustan komponenttikehyksien konfiguraatiota. Kuvan 3.3 esimerkin siirtymäkaaviossa on kolme solmua Normal, Alert ja Emergency, jotka koostuvat komponenttikehyksistä SP, BT, WIFI ja FH. Genie tarjoaa siirtymäkaavioiden mallintamiseen toimialakohtaisen kielen Transition Diagrams DSL.

Taso 2 sisältää mallit, jotka liittyvät rakenteelliseen muunneltavuuteen. Mallit liittyvät komponenttikehyksiin, niiden komponentteihin ja konfiguraatioihin. Mallinnuksessa käytettävä toimialakohtainen kielen OpenCOM DSL avulla rakennetaan tasolla 2 olevia komponenttien konfiguraatioiden malleja, joita rajoittavat komponenttikehysten arkkitehtuurit. Käytössä olevat mallinnuselementit ovat yleisiä

arkkitehtuurin osia kuten komponentit, vaadittavat ja tarjottavat rajapinnat ja sidokset. Kuvan 3.3 esimerkissä taso 2 sisältää neljä mallia: SP, BT, WIFI ja FH. Tason 2 malleista generoidaan alimman abstraktiotason 1 elementtejä, jotka koostuvat käytössä olevan väliohjelmiston komponenttien lähdekoodeista, komponenttikehysten konfiguraatiotiedostoista ja uudelleenkonfigurointisäännöistä.

Mallien käyttö Genie -työkalussa nostaa abstraktiotasoa ja mahdollistaa turvallisen ja toistettavan tavan alemman abstraktiotason komponenttien generoimiseen. Varmistaakseen generoitujen komponenttien oikeellisuuden malleihin määriteltyjä rajoitteita käytetään validoimaan konfiguraatiot ennen kuin varsinainen generointi toteutetaan. Käytössä olevat väliohjelmistoalustat mahdollistavat järjestelmän laajennettavuuden ja mukautumisen tukemalla generoitujen komponenttien lisäämistä ajonaikaisesti järjestelmään.

3.4 Havaintoja OpenChannel -kehyksen suunnittelun perustaksi

Reflektion avulla on mahdollista päätellä mitkä ovat järjestelmän variaatiopisteet, minkälaiset variaatiot ovat mahdollisia ja tutkia missä tilassa järjestelmä on millä tahansa hetkellä. Reflektion käytössä on kuitenkin joitakin haittapuolia, jotka vaikuttavat järjestämän suorituskykyyn ja eheyteen. Reflektiivisiä järjestelmiä kehittäessä täytyy tutkia joustavuuden mahdollisesti aiheuttamat vaikutukset järjestelmän suorituskykyyn. Lisäksi täytyy varmistaa ennen järjestelmään tehtäviä ajonaikaisia muutoksia, että järjestelmä säilyy eheänä ja toimii oikein muutosten jälkeen [BBF08].

OpenORB -väliohjelmiston lähestymistavasta selviää miten komponenttikehyksien käyttö väliohjelmiston rakenteessa mahdollistaa rakenteellisen muuntelun. Järjestelmän mukautumiskäytös on määritelty uudelleenkonfigurointisääntöihin, joita tulkitaan ajonaikaisesti, kun kontekstissa tai ympäristössä huomataan muutoksia.

Genie -työkalu tarjoaa toimialakohtaiset kielet rakenteellisen muuntelun ja ympäristön ja kontekstin muuntelun mallintamiseksi. Yhdistämällä malleista generoidut komponentit ja reflektiivisen komponenttikehyksistä koostuvan väliohjelmiston on mahdollista toteuttaa joustavia ajonaikaisesti mukautuvia järjestelmiä, joiden muutokset ovat turvallisia ja varmistavat järjestelmän eheyden kaikissa tilanteissa.

OpenChannel -kehyksen suunnittelussa otetaan huomioon reflektion käytön tuomat mahdollisuudet ja rajoitteet. Luvussa 5 kuvataan miten reflektiota, komponenttikehyksiä ja mallipohjaista lähestymistapaa käytetään OpenChannel -kehyksen yhteydessä toteuttamaan ja hallitsemaan kanavatyyppien dynaamista muuntelua. Toimialakohtaisia kieliä voitaisiin hyödyntää kanavatyyppien muunneltavuuden kuvaamisessa ja kanavatyyppien suunnittelussa esimerkiksi toimialakohtaisen kielen Guaraná-DSL [CFR11] avulla.

4 OpenChannel – kehyksen arkkitehtuuri

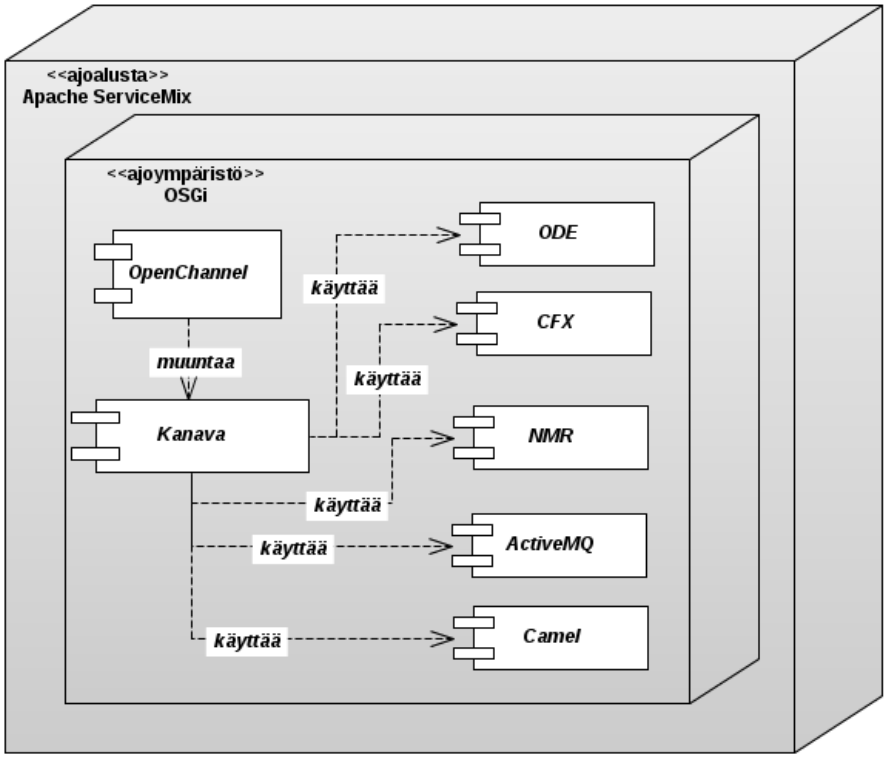
OpenChannel -kehyksen tavoitteena on tukea alustana olevan palveluväylän kanavatyyppien ajonaikaista muuntelua eri sidosryhmien tarpeiden mukaisesti sekä kontekstin ja ympäristön muutoksien aikana. Kehystä käyttäviä sidosryhmiä voivat olla esimerkiksi palveluväylän ylläpitäjät ja erilaiset asiakasohjelmat. Mahdollistaakseen laajan asiakaskunnan kehys toteuttaa avoimen ja helppokäyttöinen rajapinnan kanavatyyppien muunneltavien ominaisuuksien tutkimiseen ja muokkaamiseen, sekä kanavien luomiseen ja poistamiseen. Ympäristön ja kontekstin muutoksiin mukautumista varten kehys kykenee vastaanottamaan kanavatyyppeihin liittyviä tapahtumia, jotka laukaisevat kanavatyypin mukautumisen.

4.1 Apache ServiceMix kehyksen alustana

Apache ServiceMix [Ser13] on aktiivisen kehityksen alla oleva avoimeen lähdekoodin (Apache License v2.0) perustuva joustava ja monipuolinen palveluväylä. Apache ServiceMix palveluväylän kuvaus perustuu vuonna 2012 julkaistuun 4.4.2 versioon. Apache ServiceMix pyrkii tarjoamaan kaikki nykyaikaiselta palveluväylältä toivottavat ominaisuudet, joita on käsitelty aiemmin luvussa 2.1. Kuvassa 4.1 on esitetty palveluväylän yleinen arkkitehtuuri ja sen sisältämät komponentit. Komponentit ja niiden tarjoamat toiminnallisuudet on listattu taulukkoon 4.1.

ServiceMix arkkitehtuuri on komponenttipohjainen ja sen perustana on Java-virtuaalikoneessa ajettava OSGi-pohjainen Apache Karaf-ajoympäristö. OSGi-ajoympäristö mahdollistaa komponenttien lisäyksen, päivityksen ja poistamisen dynaamisesti ilman

ajoympäristön uudelleenkäynnistystä.



Kuva 4.1: Apache ServiceMix komponentit ja OpenChannel.

Komponentti	**Toiminnallisuus**
NMR	Apache NMR (Normalized Message Router) on palveluväylän sisäisen reitityksen ja sanomanvälityksen hoitava komponentti.
Camel	Integraatiokehys, joka sisältää sanoman välitystä, reititystä ja eri tietoliikenneprotokollien ja viestitysmallien välisiä muunnoksia tarjoavia komponentteja. Tukee suurinta osa yleisesti tunnetuista [HoW03] integraatiomalleista (Enterprise Integration Patterns).
CXF	Monipuoliset Web Services -toiminnallisuudet tarjoava kehys.
ActiveMQ	JMS-standardin toteuttava sanomanvälityspalvelin (message broker).
ODE	WS-BPEL standardin toteuttava prosessimoottori.

Taulukko 4.1: ServiceMix palveluväylän komponentit.

Palveluväylän käyttämä Camel -kehys sisältää useita valmiita komponentteja kanavien rakentamiseen ja valmiiden komponenttien lisäksi uusia komponentteja voidaan toteuttaa itse. Kanavat voidaan toteuttaa käyttämällä Java-pohjaista DSL-kieltä tai vaihtoehtoisesti käyttämällä XML-kuvausta. Näistä vaihtoehdoista OpenChannel -kehyksen käyttöön valittiin XML-kuvaus, vaikka Java-pohjainen DSL-kieli olisi ollut luettavampi ja ilmaisuvoimaisempi. Valintaan vaikutti se, että XML-kuvaus tarjoaa paremmat mahdollisuudet ja työkalut kanavatyyppien muunneltavien ominaisuuksien mallintamiselle ja mallin validoimiselle.

Alla on esimerkki yksinkertaisesta kanavasta, joka siirtää tiedoston from-elementin uri-attribuutin määrittelemästä lähdehakemistosta vastaavaan to-elementissä määriteltyyn kohdehakemistoon ja kirjoittaa lokille siirretyn tiedoston nimen.

```
<camelContext xmlns="http://camel.apache.org/schema/blueprint">
   <route>
      <from uri="file:/tmp/from"/>
      <log message="Moving ${file:name} to destination"/>
      <to uri="file:/tmp/to"/>
  </route>
</camelContext>
```

Esimerkki 4.1: Kanavan kuvaus

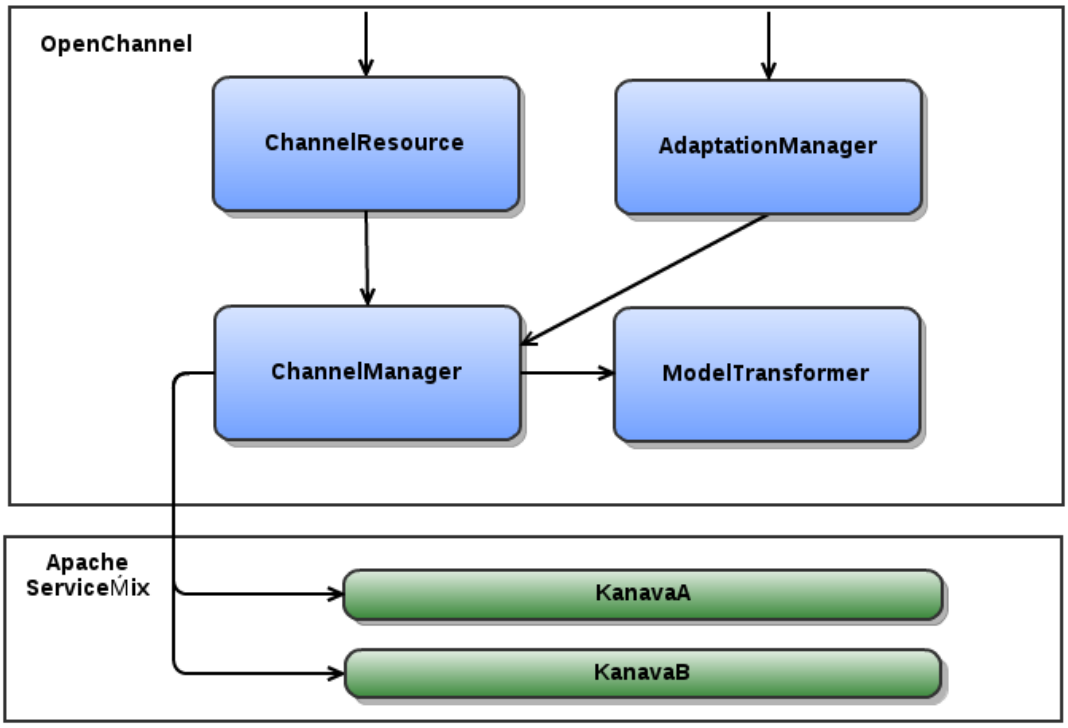
Kanavan asentamiseksi ajoympäristöön riittää kanavan XML-kuvauksen tallentaminen palveluväylään määriteltyyn deploy-hakemistoon, jonka jälkeen alusta generoi kuvauksen perustella kanavan toteutuksen ja käynnistää kanavan automaattisesti. Jotta ajossa olevien kanavien suoritukset eivät häiriintyisi kanavien päivityksen aikana, niin alusta sulkee vanhan kanavan ja käynnistää uuden kanavan vasta, kun vanhan kanavan kaikki suoritukset ovat päättyneet.

4.2 OpenChannel arkkitehtuurin yleiskuvaus

OpenChannel -kehys tukee rakenteellista muuntelua tarjoamalla toiminnallisuudet kaikkien palveluväylässä olevien kanavatyyppien listaamiseen, kanavan luomiseen ja poistamiseen, yksittäisen kanavatyypin muunneltavien ominaisuuksiin mallin hakemiseen ja mallin muokkaamiseen. Toiminnallisuudet tarjotaan avoimen HTTP-

pohjaisen rajapinnan kautta. Toteuttaakseen toiminnallisuudet kehykseen kuuluu kanavien hallinnan, mallien validoinnin ja mallimuunnoksien toteuttavia komponentteja. Ympäristön ja kontekstin muutoksiin mukautumista varten kehys sisältää komponentin, joka vastaanottaa kontekstin ja ympäristön muutoksiin liittyviä tapahtumia ja laukaisee tarvittaessa kanavatyyppien mukautumisen. OpenChannel -kehykseen ei ole toteutettu tyyppitietovarastoa, josta kanavatyyppien kuvaukset ja niihin liittyvät muunneltavien ominaisuuksien mallit olisivat haettavissa.

Kehystä voidaan ajaa samassa ajoympäristössä, jossa palveluväylän kanavia ajetaan. Tämän ratkaisun avulla kehys voi hyödyntää toteutuksessaan samoja palveluväylän palveluja, jotka ovat kanavien käytössä. Kuvassa 4.2 on esitetty Open-Channel -kehyksen arkkitehtuurin korkean tason kuvaus. Arkkitehtuurin komponenttien vastuut ja toteutusteknologiat on listattu taulukossa 4.2. Komponenttien vastaavuus konkreettiseen toteutukseen ja viite lähdekoodien säilytyspaikkaan löytyvät liitteestä 1.



Kuva 4.2: OpenChannel arkkitehtuurin korkean tason kuvaus.

Komponentti	Vastuut	Teknologia
ChannelResource	Toteuttaa avoimen rajapinnan kanavatyyppien kanavien muunneltavien ominaisuuksien mallien listaamiseen ja operaatiot niiden muokkaamiseen.	Java, JAX-RS
ChannelManager	Sisältää operaatiot kanavatyyppien toteutusten ja kanavatyyppien muunneltavien ominaisuuksien mallien hallintaan.	Java
AdaptationManager	Vastaanottaa ympäristön ja kontekstin muutoksiin liittyviä tapahtumia ja laukaisee tarvittaessa kanavatyypin uudelleenkonfiguroinnin.	Java
ModelTransformer	Suorittaa muunnoksia molempiin suuntiin muunneltavien ominaisuuksien mallin ja kanavatyypin kuvauksen välillä.	Java, XSLT, JAXB

Taulukko 4.2: OpenChannel -kehyksen komponentit ja niiden vastuut.

Kanavatyypit ja niiden muunneltavat ominaisuudet tarjotaan REST-arkkitehtuurityylin [FiT02] mukaisina resursseina, joiden esitykset ovat käsiteltävissä HTTP-metodeilla. REST-arkkitehtuurityylin käyttämä HTTP-protokolla sopii hyvin avoimen rajapinnan toteuttamiseen sen helppokäyttöisyyden, alustariippumattomuuden ja palomuuriystävällisyyden takia. Lisäksi erilaisten asiakassovellusten ja käyttöliittymien toteuttaminen sen päälle on helppoa [GMT11].

OpenChannel -kehyksen suunnittelussa on otettu huomioon mahdollisuus käyttää alustana jotakin toista palveluväylää kuin Apache ServiceMix. Vain reflektiomekanismi ja transformaatiot riippuvat käytössä olevasta palveluväylästä. Tästä syystä ChannelManager ja ModelTransformer komponentit ovat Javan rajapinta-luokkia ja niille voidaan tehdä palveluväyläkohtaiset toteutukset.

Kanavatyyppien muunneltavien ominaisuuksien malli esitetään XML-dokumenttina, jonka rakenne on määritelty XML-skeemalla. XML-skeeman käyttö mahdollistaa mallin validoinnin ja lisäksi skeemasta voidaan generoida toteutuksen luokkia. Rajapinnan operaatiot ja niiden kuvaukset on listattu taulukossa 4.3.

Operaatio	REST URI	Kuvaus
listChannels()	GET /channels/	Palauttaa listan kaikista palveluväylän kanavatyypeistä.
getChannel(id)	GET /channels/{id}	Palauttaa parametrina saadun kanavatyypin muunneltavien ominaisuuksien mallin.
updateChannel(id, model)	PUT /channels/{id}	Päivittää olemassa olevan kanavan rakenteen ja ominaisuudet uuden mallin mukaisiksi.
createChannel(model)	POST /channels/	Luo uuden kanavan mallin perusteella.
deleteChannel(id)	DELETE /channels/{id}	Poistaa kanavan.

Taulukko 4.3: OpenChannel -kehyksen rajapinnan operaatiot.

Seuraavassa esimerkki rajapinnan tyypillisestä käyttötapauksesta, jossa käyttäjä haluaa muokata kanavatyypin ominaisuuksia. Esimerkissä käytetty kanavatyyppi tunnuksella FileCopy on sama tiedoston lähdehakemistosta kohdehakemistoon siirtävä kanavatyyppi, joka on kuvattu edellisen luvun esimerkissä 4.1.

1. Käyttäjä tekee GET-pyynnön kehykselle ja saa vastauksena listan palveluväylän kanavatyypeistä. Jokainen kanavatyypin kuvaus pitää sisällään kanavatyypin tunnuksen, URI:n josta kanavatyypin muunneltavien ominaisuuksien malli voidaan hakea ja vapaamuotoisen kanavatyypin kuvauksen.

```
GET http://example.org/channels
```

```
<channels>
    <channel>
        <id>FileCopy</id>
        <link href="http://example.org/channels/FileCopy"/>
        <description>Kanavatyypin file-copy kuvaus</description>
    </channel>
    <channel>
        <id>SecureService</id>
        <link href="http://example.org/channels/SecureService"/>
        <description>Kanavatyypin secure-service
      kuvaus</description>
    </channel>
</channels>
```

2. Käyttäjä pyytää kehykseltä kanavatyypin FileCopy muunneltavien ominaisuuksien mallin tekemällä GET-pyynnön edellisen pyynnön vastaukselta saatuun URI:iin. Vastauksella saadaan kanavatyypin muunneltavien ominaisuuksien malli. Vastaus pitää sisällään myös mallin rakennetta kuvaavan XML-skeeman sijainnin, jonka perusteella käyttäjä voi päätellä minkälaiset muutokset malliin ovat sallittuja.

```
GET http://example.org/channels/FileCopy
```

```
<channel xsi:schemaLocation="http://example.org/FileCopy.xsd">
    <source>/tmp/from</source>
    <destination>/tmp/to</destination>
    <logging>false</logging>
</channel>
```

3. Käyttäjä tekee muutoksia mallin lähde- ja kohdehakemistot sisältävien elementtien arvoihin ja lisäksi asettaa lokikirjoituksen päälle muuttamalla logging-elementin arvoksi true. Muutosten jälkeen käyttäjä tekee PUT-pyynnön kehykselle, joka päivittää kanavatyypin vastaamaan muutettua mallia.

```
PUT http://example.org/channels/FileCopy
```

```
<channel xsi:schemaLocation="http://example.org/FileCopy.xsd">
    <source>/tmp/new/from</source>
    <destination>/tmp/new/to</destination>
    <logging>true</logging>
</channel>
```

OpenChannel -kehys välittää virhetilanteista tiedon käyttäjälle palauttamalla tilanteeseen sopivan HTTP-paluukoodin ja virheen tarkemman kuvauksen. Esimerkiksi, jos yritetään hakea kanavatyyppiä jota ei ole, niin palautetaan HTTP-paluukoodi 404 - Not Found.

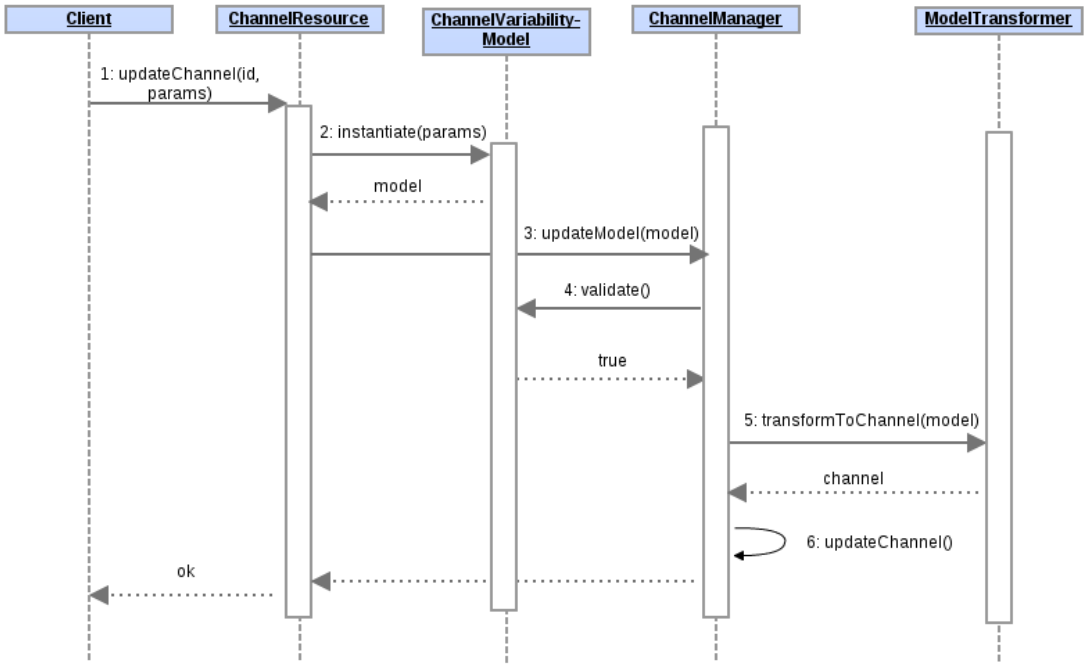
4.3 OpenChannel -kehyksen toteutus

Kehyksen toteutus voidaan erottaa kahdeksi loogiseksi kokonaisuudeksi sen mukaan minkälaiseen muuntelun se mahdollistaa. Rakenteellista muuntelua varten kehys toteuttaa avoimen rajapinnan (kuvaus 4.1), jonka kautta kanavien rakenteeseen ja toiminnallisuuteen päästään käsiksi. Kontekstin muutoksien aiheuttamaa muuntelua varten kehys sisältää komponentteja, jotka vastaanottavat kontekstin muutoksiin liittyviä tapahtumia ja laukaisevat tarpeen vaatiessa kanavan uudelleenkonfiguroinnin. Komponenttien vastaavuus konkreettiseen toteutukseen ja viite lähdekoodien säilytyspaikkaan löytyvät liitteestä 1.

```
/**
 * Returns state of channel's variability model.
 *
 * @param channelId
 *            channel identifier
 * @return XML representation of channel's variability model
 */
@GET
@Path("/{channelId}")
@Produces(MediaType.APPLICATION_XML)
public String getChannel(@PathParam("channelId") String channelId);

/**
 * Returns all currently executing channels.
 *
 * @return list of channels
 */
@GET
@Path("/")
@Produces(MediaType.APPLICATION_XML)
public Response listChannels();

/**
 * Updates channel based on values in variability model.
 *
 * @param channelId
 *            channel identifier
 * @param model
 *            channel variability model
 * @return
 */
@PUT
@Produces(MediaType.APPLICATION_XML)
@Consumes(MediaType.APPLICATION_XML)
@Path("/{channelId}")
public Response updateChannel(@PathParam("channelId") String channelId,
               ChannelVariabilityModel model);

/**
 * Removes executing channel from platform.
 *
 * @param channelId
 *            channel identifier
 * @return
 */
@DELETE
@Path("/{channelId}")
public Response deleteChannel(@PathParam("channelId") String channelId);

/**
 * Creates a new channel based on values in variability model.
 *
 * @param model
 *            channel variability model
 * @return
 */
@POST
@Consumes(MediaType.APPLICATION_XML)
@Path("/")
public Response createChannel(ChannelVariabilityModel model);
```

Kuvaus 4.1: ChannelResource -luokka.

Kanavan päivittämistä varten kanavatyypin muunneltavien ominaisuuksien malli on pitänyt ensin hakea OpenChannel -kehyksen avulla. Tämän jälkeen päivitetty malli voidaan lähettää takaisin kehykselle, joka muokkaa kanavan vastaamaan uutta mallia. Kuvassa 4.3 on sekvenssikaavio kanavan päivityksen vaiheista.



Kuva 4.3: Kanavan päivityksen sekvenssikaavio.

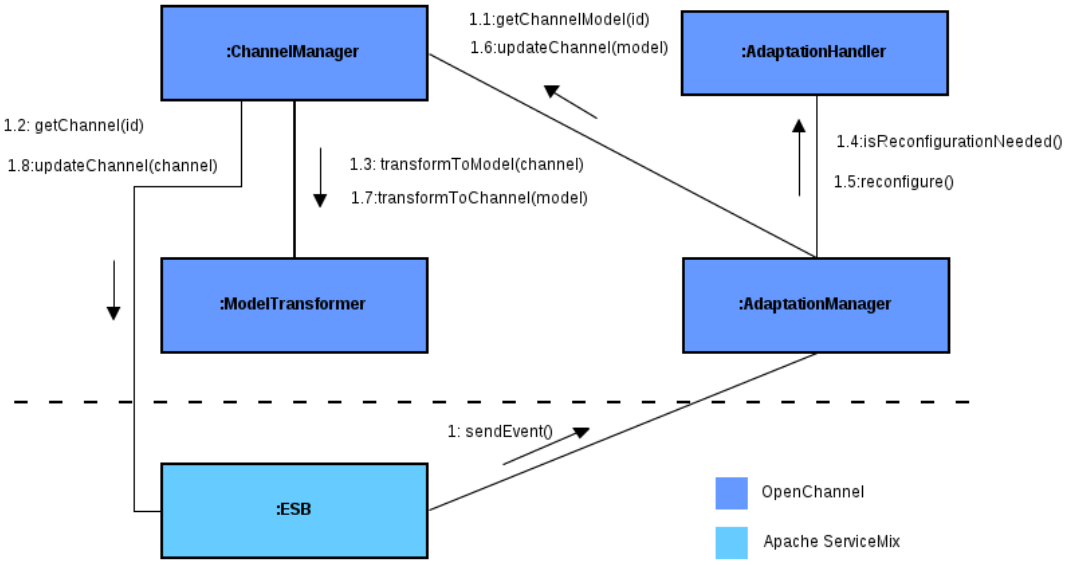
Kanavatyypin muunneltavien ominaisuuksien mallin päivittämisen vaiheet ovat:

1. Asiakas lähettää HTTP-pyynnön kehykselle, jonka ChannelResource komponentin operaatio updateChannel() vastaanottaa. Pyynnön parametreina saadaan kanavatyypin tunnus ja muunneltavien ominaisuuksien malli.
2. ChannelResource komponentti luo pyyntöä vastaavan kanavatyypin muunneltavien ominaisuuksien mallin.
3. ChannelResource kutsuu ChannelManagerin updateModel()-operaatiota parametrina edellisessä vaiheessa luotu malli.
4. ChannelManager todentaa mallin oikeellisuuden mallin validate()-operaatiolla.
5. Muuntaaksen muunneltavien ominaisuuksien mallin kanavatyypin mallin

XML-kuvaukseksi ChannelManager kutsuu ChannelTransform-komponentin transform()-operaatiota.

6. ChannelManager tallentaa kanavatyypin mallin XML-kuvauksen palveluväylän ennalta määriteltyyn hakemistoon, josta palveluväylä lukee kuvauksen ja generoi sen perusteella kanavan toteutuksen ja käynnistää kanavan.

Kontekstin ja ympäristön muutoksiin reagointia varten OpenChannel -kehys sisältää AdaptationManager-komponentin, joka kykenee vastaanottamaan tapahtumia erilaisista lähteistä. Kuvan 4.4 kommunikaatiokaavion esimerkissä AdaptationManager vastaanottaa tapahtumia palveluväylältä, mutta tapahtumia voidaan vastaanottaa myös järjestelmän ulkopuolelta esimerkiksi jonon kautta. Tapahtumat ovat käytännössä avain-arvo-pareja, jotka osataan ohjata oikealle kanavatyyppikohtaiselle käsittelijälle kanavatyypin tunnisteen perusteella.



Kuva 4.4: Kontekstin muutoksiin reagoinnin kommunikaatiokaavio.

Kehyksen kontekstin muutoksiin reagoinnin vaiheet ovat:

1. Palveluväylä lähettää kontekstin muutoksesta tapahtuman, jonka AdaptationManager vastaanottaa.

1.1 AdaptationManager pyytää ChannelManager komponentilta tapahtumaa koskevan kanavatyypin muunneltavien omaisuuksien mallin.

1.2 ChannelManager hakee kanavatyypin toteutuksen kuvauksen palveluväylästä.

1.3 ChannelManager muuntaa kanavatyypin kuvauksen muunneltavien ominaisuuksien malliksi ModelTransformer komponentin avulla.

1.4 Kanavatyyppikohtainen AdaptationHandler tutkii onko kanavan ominaisuuksia tai rakennetta tarpeen muokata kontekstin muutoksen takia.

1.5 Jos muokkaaminen on tarpeen, niin AdaptationHandler suorittaa tarvittavat muutokset malliin.

1.6 AdaptationManager välittää päivitetyn mallin ChannelManagerille.

1.7 ModelTransformer muuntaa mallin takaisin kanavatyypin kuvaukseksi.

1.8 Kanavatyypin kuvaus tallennetaan palveluväylän alustan hakemistoon, josta palveluväylä käynnistää kanavan.

Yhdistämällä Apache ServiceMix palveluväylän tarjoamat toiminnallisuudet OpenChannel -kehyksen lisäominaisuuksilla voidaan toteuttaa kanavatyyppien dynaamisen muuntelun vaatimukset. Kehyksen tarjoama helppokäyttöinen rajapinta mahdollistaa toiminnallisuuksien laajan hyödyntämisen erilaisten sidosryhmien ja asiakasohjelmien käyttöön. OpenChannel -kehys varmistaa, että kanavatyyppien muutokset ovat turvallisia, eikä ajossa olevan järjestelmän toiminta häiriinny.

5 Kanavatyyppien dynaaminen muuntelu OpenChannel -kehyksen avulla

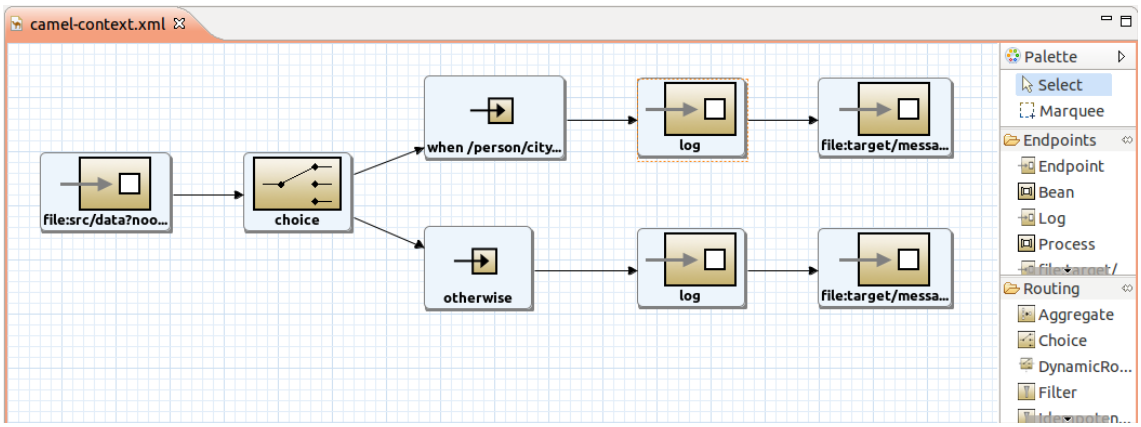
Kuten aiemmin on todettu, kanavatyyppien dynaamisen muuntelun avulla voidaan ratkaista useita palvelukeskeisen arkkitehtuurin vaatimuksista. OpenChannel -kehys hyödyntää komponenttikehyksiä yhdessä reflektion kanssa muuntelun toteutukseen. Abstraktiotason nostamiseksi alustariippuvaisen kanavatyypin mallin muuntelua tukeva toiminnallisuus esitetään alustariippumattomana kanavatyypin muunneltavien ominaisuuksien mallina. Alustariippumattomasta mallista voidaan mallitransformaatioiden avulla tuottaa kanavatyypin malli erilaisille palveluväyläalustoille. Kehys pitää huolen siitä, että muunneltavien ominaisuuksien malliin tehdyt muutokset heijastuvat kanavatyypin malliin ja sitä kautta palveluväylän toimesta ajossa olevaan

kanavaan. Dynaamisesti mukautuvan kanavatyypin kehittämisessä muunneltavuuden tuomat haasteet pitää ottaa huomioon kaikissa kehityksen vaiheissa vaatimusmäärittelystä toteutukseen.

5.1 Reflektion ja mallien käyttäminen dynaamisen muuntelun perustana

Palveluväylässä olevien kanavatyyppien dynaamisen muuntelu edellyttää, että kanavatyypin toiminnallisuuden toteuttavilla kokonaisuuksilla on selkeästi määritelty rakenne ja lisäksi vaaditaan mekanismeja rakenteen tutkimiseen ja muokkaamiseen. Komponenttikehyksiä ja reflektiota käytetään dynaamisesti mukautuvien väliohjelmistoalustojen toteuttamiseen [BBC05]. Jos kanavatyypit rakentuvat komponenttikehyksistä ja mahdollistavat reflektion, niin kanavatyyppien dynaaminen muuntelu on mahdollista toteuttaa.

Apache ServiceMix -palveluväylän kanavat rakennetaan yhdistelemällä palveluväylän tarjoamia komponentteja rakenteeseen, jonka pitää noudattaa alustan määrittelemiä sääntöjä. Apache ServiceMix -palveluväylän kanavatyypit voidaan siten rinnastaa komponenttikehyksiin. Esimerkissä 5.1 on kuvankaappaus Fuse IDE -työkalusta, jolla kanavien kuvaukset voidaan määritellä graafisesti. Työkalun oikeassa laidassa on lista valittavista palveluväylän komponenteista ja keskellä on piirtoalue, jolle komponentit on sijoitettu. Komponentit muodostavat yhdessä komponenttikehyksen, jonka rakenteen eheyden työkalu varmistaa.



Esimerkki 5.1: Kanavien kuvauksen graafinen näkymä Fuse IDE työkalussa.

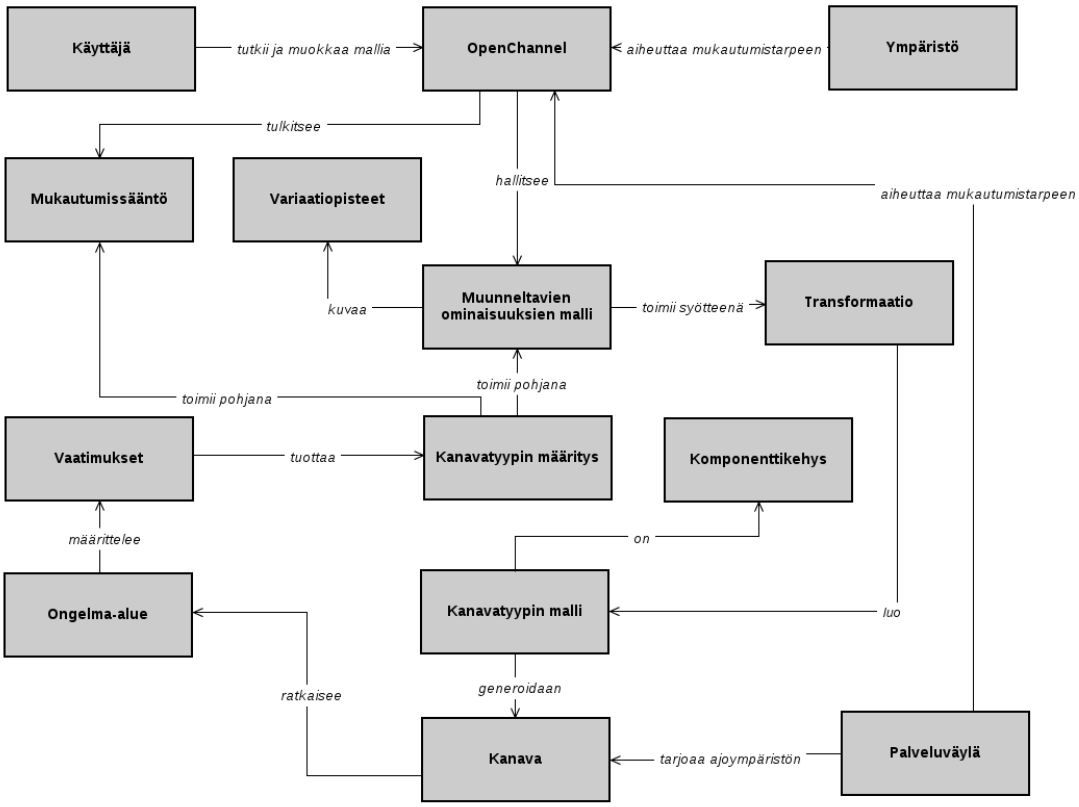
Apache ServiceMix mahdollistaa reflektion lukemalla ja muokkaamalla palveluväylän deploy-hakemistossa olevia kanavien XML-pohjaisia kuvauksia. Ajoympäristö pitää huolen siitä, että kanavien kuvausten muutokset heijastuvat kanavan toteutukseen.

Apache ServiceMix -palveluväylän kanavatyypit kuvataan XML-kuvauksella, joka pitää sisällään kaiken tiedon, jota alusta tarvitsee kanavan toteutuksen generoimiseen. Apache ServiceMix -palveluväylän reflektiomekanismia käyttämällä olisi mahdollista toteuttaa rajapinta, jonka kautta koko kanavatyypin malli olisi tutkittavissa ja muokattavissa. Ratkaisun etuna on monipuoliset mahdollisuudet muuntelulle, koska koko kanavan rakenne ja siihen kuuluvien komponenttien ominaisuudet olisivat muokattavissa. Monimutkaisesta matalan tason kuvauksesta on kuitenkin vaikea tunnistaa kohtia, joissa muuntelu olisi hyödyllistä eikä muutoksien vaikutuksia kanavan toimintaan voida ennakoida luotettavasti.

OpenChannel -kehys ratkaisee ongelman kuvaamalla kanavatyypin mallista erillisellä korkeamman abstraktiotason muunneltavien ominaisuuksien mallilla vain ne osat, joiden muuntelua halutaan tukea. Malli sisältää myös rajoitteet, jotka määrittelevät sallitut muutokset. Mallissa voi olla esimerkiksi rajoitettu, että kanavan käyttämä todennustapa voi olla joko BASIC tai CLIENT-CERT. Muunneltavien ominaisuuksien mallia käytetään sekä määrittelemään dynaamista muuntelua suunnitteluvaiheessa että hallitsemaan ajonaikaisia muutoksia.

5.2 OpenChannel ratkaisun yleiskuvaus

OpenChannel -kehyksen lähestymistavan perustana ovat kanavatyyppien määritysten perusteella luodut alustariippumattomat muunneltavien ominaisuuksien mallit ja transformaatiot muunneltavien ominaisuuksien mallien ja alustariippuvaisen kanavatyyppien mallien välillä. Kuvassa 5.1 on nähtävissä kehykseen liittyvät käsitteet ja niiden väliset suhteet.



Kuva 5.1: OpenChannel käsitemalli.

Ongelma-alueen vaatimusten perusteella tuotetaan kanavatyypin määritys, joka toimii pohjana muunneltavien ominaisuuksien mallille sekä kanavatyyppiin mahdollisesti liittyville mukautumissäännöille. Kanavatyypin määritys sisältää kanavatyypin perustoiminnallisuuden eikä se pidä sisällään kanavatyypin muunteluun liittyvää tietoa. Kanavatyypin toiminnallisuus voidaan kuvata esimerkiksi jollakin teknologiariippumattomalla visuaalisella notaatiolla [HoW03] tai jollakin sovellusintegraatioiden suunnittelua varten kehitetyllä toimialakohtaisella kielellä kuten Guaraná-DSL [CFR11]. Kanavatyyppiin liittyvää muuntelua kuvataan erillisellä muunneltavien ominaisuuksien mallilla ja mahdollisilla mukautumissäännöillä. Muunneltavien ominaisuuksien malli pitää sisällään tiedon kanavatyyppiin liittyvistä variaatiopisteistä ja niiden rajoitteista. Mukautumissäännöillä kuvataan tilasiirtymäkaaviona, joista ilmenee miten kanavatyyppi reagoi kontekstin ja ympäristön muutoksiin.

Kehyksen tärkeänä tehtävänä on varmistaa, että alustariippumaton muunneltavien ominaisuuksien malli vastaa alustariippuvaista kanavatyypin mallia (kausaalinen yhteys). Kausaalinen yhteys toteutetaan muunneltavien ominaisuuksien mallin ja kanavatyypin mallin välisillä molempiin suuntiin tehtävillä transformaatioilla. Kanavatyypin muunneltavien ominaisuuksien mallia tutkittaessa kehys suorittaa muunnoksen kanavatyypin mallista muunneltavien ominaisuuksien malliksi. Toiseen suuntaan tehtävää muunnosta tarvitaan, kun päivitetään kanavatyypin malli vastaamaan muunneltavien ominaisuuksien malliin tehtyjä muutoksia.

Kehyksen avulla kanavatyypit pystyvät reagoimaan eri tahoilta saapuviin mukautumistarpeisiin. Kehys kykenee vastaanottamaan ympäristön ja kontekstin muutoksista tapahtumia, joiden perusteella kanavatyyppiin liittyvä mukautumissääntö osaa päätellä miten kanavatyypin pitää mukautua tapahtumaan. Esimerkkeinä ympäristön muutoksista ovat kanavatyypin käyttämän ulkoisen palvelun todennustavan muuttuminen ja kanavatyypin palvelutasosopimuksen muutokset. Palveluväylän resurssien väheneminen on esimerkki kontekstin muutoksesta, joka voi aiheuttaa tarpeen kanavatyypin mukautumiselle. Kehyksen tarjoaman rajapinnan kautta käyttäjä tutkii kanavatyypin muunneltavien ominaisuuksien mallin tilaa ja tekee siihen muutoksia. Käyttäjä voi olla joko ihminen kuten palveluväylän ylläpitäjä tai jokin kanavatyyppien hallintaa varten toteutettu sovellus.

OpenChannel -kehys ei sisällä kanavatyypin valintaan eikä kanavatyypin muunneltavien ominaisuuksien mallin arvojen valintaan liittyvää logiikkaa. Valinnat tekee manuaalisesti käyttäjä tai automaattisesti jokin kanavatyyppien hallintasovellus. Kehys ei sisällä myöskään vastaava toiminnallisuutta, jota ODP viitemallin [ODP98a] tyyppitietovarasto tarjoaa. Tyyppitietovarastoa voitaisiin käyttää muunneltavien ominaisuuksien mallien ja kanavatyyppeihin liittyvän metadatan varastona.

OpenChannel -kehyksen käyttämä lähestymistapa yhdistää reflektion ja komponenttikehyksien tuomat mahdollisuudet kanavatyyppien muunteluun. Kehys hyödyntää mallipohjaista lähestymistapaa määrittelemään dynaamista muuntelua suunnitteluvaiheessa ja hallitsemaan ajonaikaisia muutoksia. Esimerkkejä muutoksien aiheuttajista ovat palvelun laatuvaatimusten muutokset, ympäristön muutokset ja uudet käyttäjäryhmät.

5.3 OpenChannel -kehyksen hyödyntäminen palvelukeskeisessä arkkitehtuurissa

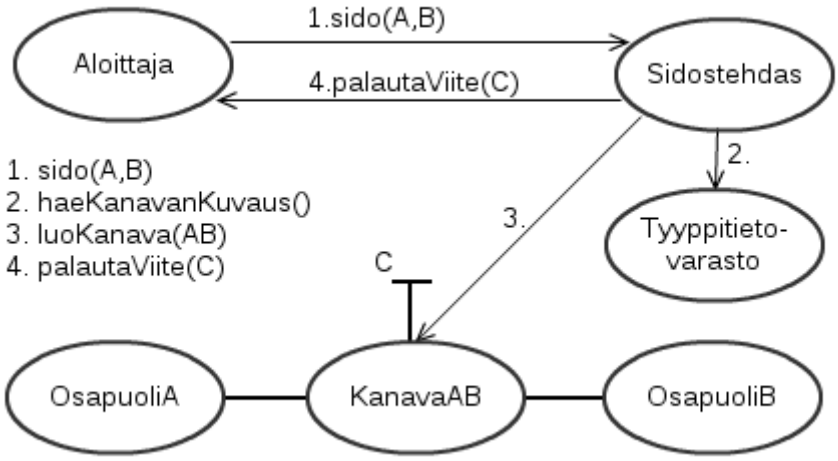
Kuten palvelukeskeisen arkkitehtuurin yleisiä piirteitä esittelevässä luvussa 2.1 todettiin, palvelun valinta ja palveluiden välisen sidoksen muodostaminen ovat keskeisessä roolissa palveluiden yhteentoimivuuden varmistamisessa. Palvelun valintaan ja sidoksen muodostamiseen liittyy Web Services standardi UDDI [CHR04]. UDDI määrittelee palvelurekisterin, johon palveluntarjoajat rekisteröivät palvelun kuvauksen ja sen käyttämiseen tarvittavat tiedot. UDDI ei kuitenkaan tue palvelun valintaa ajonaikaisesti evaluoitavien parametrien perusteella eikä siten kykene hyödyntämään dynaamisesti muunneltavien kanavatyyppien ominaisuuksia.

ODP viitemalli [ODP98a] sisältää ODP meklaripalvelun (trading service) määritelmän [ODP97]. ODP meklaripalvelu toteuttaa vastaavan toiminnallisuuden kuin UDDI, mutta lisäksi se mahdollistaa sopivan palveluntarjoajan valinnan ajonaikaisesti tukemalla palveluiden välisiä dynaamisia sidoksia. RM-ODP standardi Interface References and Binding [ODP98b] määrittelee kehyksen dynaamisen sidonnan kuvaamiselle ja toteutukselle.

ODP viitemallissa dynaamisen sidonnan perustana on sidontaprosessi (binding process) (kuva 5.2), jonka lopputuloksena syntyy sidossuhde (binding liasion). Sidontaprosessin aikana prosessiin osallistuvat järjestelmät vaihtavat metatietoa palveluista ja prosessia tukevista komponenteista. Sidossuhde varmistaa, että sidoksen osapuolten välille voidaan luoda kommunikointikanava [Kut98]. Sidossuhteen perusteella luodaan osapuolien välinen sidossopimus (binding contract). Sidossopimus sisältää kaiken tiedon, jota sidoksen luomiseen tarvitaan. Sidossopimukseen kuuluu oleellisena osana tieto kaikkien osapuolten tunnistamasta kanavatyypistä, joka kuvaa kanavalta odotetun toiminnallisuuden ja siltä vaaditun käytöksen kanavan virhetilanteissa. Kanavan toteutus voi olla hajautettuna usealle hallintoalueelle (domain), joten sidossopimuksen tiedot kopioidaan kaikille sidossuhteen osapuolille, jotta ne kykenevät luomaan oman osuutensa kanavasta [Kut98].

Jokaisella hallintoalueella on oma sidostehdas (binding factory), joka suorittaa varsinaisen kanavan tai kanavan osan luomisen. Sidostehdas käyttää kanavan luomisessa sidossopimuksella olevia tyyppitietoja ja täydentää niitä tyyppitietovarastosta (type re-

pository) haetulla kanavatyypin kuvauksella. Kun kanava on luotu, niin sidostehdas palauttaa sidonnan aloittajalle viitteen kanavakontrollerin (channel controller) rajapintaan. Kanavakontrolleri sisältää hallintarajapinnan, joka mahdollistaa sidossuhteen muutokset sidossuhteen elinkaaren aikana. Kanavakontrollerilla on jokaisella hallintoalueella erillinen komponentti, jotka yhdessä pystyvät tarjoamaan sidossuhteen hallintapalvelun [Kut98].



Kuva 5.2: RM-ODP sidontaprosessi.

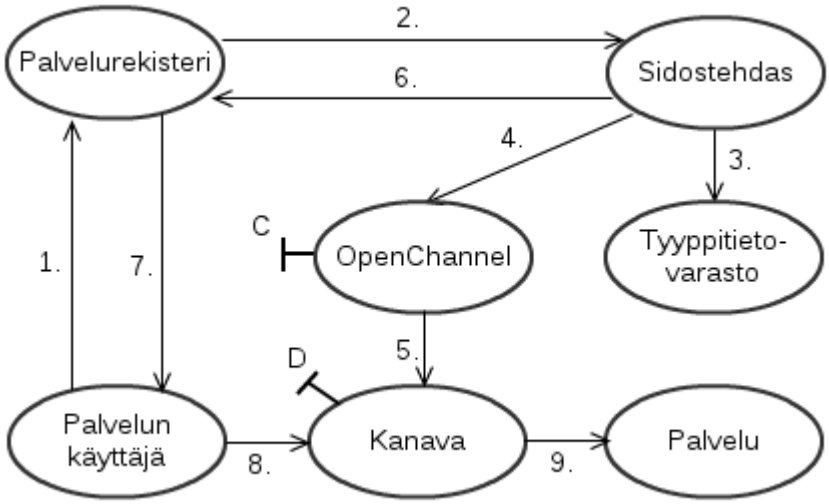
OpenChannel -kehystä voidaan hyödyntää palvelukeskeisessä arkkitehtuurissa RM-ODP sidontaprosessia muistuttavassa palveluiden dynaamisen sidonnan mallissa (kuva 5.3). Dynaamisen sidonnan edellytyksenä on, että arkkitehtuuriin kuuluu palvelurekisteri, joka mahdollistaa palvelun valinnan abstraktin palvelutyypin ja palvelun laatuominaisuuksien perusteella. Sellainen on esimerkiksi dynaamisilla ominaisuuksilla laajennettu UDDI [APS06]. Toinen edellytys on tyyppitietovarasto, johon on tallennettuna tieto kanavatyypeistä ja niiden muunneltavien ominaisuuksien malleista. Sidostehdas käyttää tyyppitietovarastoa hyväkseen valitessaan kanavatyypin muunneltavien ominaisuuksien mallille sopivat arvot, jotta muodostettava kanavatyypin malli täyttää palvelusopimuksen palvelun laatuvaatimukset.

Kanavan toteutus voi olla hajautettuna usealle hallintoalueelle, jolloin myös jokaisella hallintoalueella on oma paikallinen sidostehdas. Paikalliset sidostehtaat toimivat yhteystyössä ja vaihtavat tietoja keskenään esimerkiksi päätepisteiden osoitteista.

Paikallisilla sidostehtailla saattaa olla palvelusopimuksen laatuvaatimuksien parametreille erilaiset tiedon esitysmuodot, jolloin ne käyttävät tyyppitietovarastoa hyväkseen muuntaakseen palvelusopimuksella olevat tiedot sidostehtaan tukemaan esitysmuotoon. Hajautetun kanavan tapauksessa jokaisella hallintoalueella pitää myös olla oma OpenChannel instanssi, joka kykenee luomaan alustariippumattomasta muunneltavien ominaisuuksien mallista kyseisellä hallintoalueella käytössä olevan alustan mukaisen alustariippuvaisen kanavatyypin mallin.

Palveluiden dynaaminen sidonta sisältää seuraavat vaiheet:

1. Palvelun käyttäjä pyytää palvelurekisteriltä palvelutyypin ja palvelun laatuparametrien mukaisen palvelun kutsumiseen tarvittavia tietoja.
2. Palvelurekisteri valitsee parametrien perusteella kriteerit täyttävän palvelun ja kanavatyypin, jonka jälkeen palvelurekisteri pyytää sidostehdasta luomaan kanavan palvelun käyttäjän ja palvelun välille.
3. Sidostehdas hakee tyyppitietovarastosta alustariippumattoman kanavatyypin muunneltavien ominaisuuksien mallin ja asettaa sille arvot, jotka toteuttavat palvelun laatuparametrien vaatimukset. Sidostehdas voi käyttää tyyppitietovarastoa hyväkseen palvelun laatuparametrien tulkitsemiseen.
4. Sidostehdas välittää muunneltavien ominaisuuksien mallin OpenChannel -kehyksen rajapinnalle C.
5. OpenChannel -kehys muodostaa muunneltavien ominaisuuksien mallin perusteella alustariippuvaisen kanavatyypin mallin, jonka pohjalta alusta generoi kanavan toteutuksen.
6. Sidostehdas palauttaa palvelurekisterille kanavan rajapinnan D osoitteen ja rajapinnan C osoitteen, jonka kautta kanavaa voidaan muunnella.
7. Palvelurekisteri palauttaa molemmat osoitteet palvelun käyttäjälle.
8. Palvelun käyttäjä kutsuu kanavaa.
9. Kanava välittää kutsun palvelulle.



Kuva 5.3: Palveluiden dynaamisen sidonnan malli.

Kanavan luomisen jälkeen palvelun käyttäjä voi tehdä muutoksia kanavaan OpenChannel -kehyksen rajapinnan kautta. Tämä mahdollistaa kanavan mukautumisen palveluiden välisen sidoksen elinkaaren aikana tapahtuviin muutoksiin. Edellä kuvattu palveluiden dynaamisen sidonnan malli toteuttaa palvelukeskeisen arkkitehtuurin vaatimukset palveluiden löyhälle kytkennälle ja yhteentoimivuudelle.

5.4 Suorituskykyvaatimusten muutoksiin mukautuva kanavatyyppi

Yksi tärkeimmistä palvelun laatuun liittyvistä ominaisuuksista saatavuuden ja virheettömyyden ohella on suorituskyky [BGR05]. Kanavatyypille, joka kykenee mukautumaan suorituskykyvaatimusten muutoksiin on helppo löytää useita käyttötarkoituksia. Palvelun käyttäjä voi olla valmis maksamaan palvelun käytöstä enemmän ruuhka-aikoina taatakseen riittävän nopean vastausajan tai käyttäjä voi tyytyä alhaisempaan turvatasoon saadakseen paremman suorituskyvyn. Seuraavassa esitetään miten OpenChannel -kehyksen avulla kehitetään Apache ServiceMix -palveluväylään kanavatyyppi, joka pystyy mukautumaan käyttäjän tarpeisiin ja ympäristön muutoksiin.

Osakekaupankäynti on esimerkki sovellusalueesta, jonka ratkaisujen kehittämisessä suorituskykyvaatimukset ovat oleellisia. Tässä skenaariossa ajantasaisten pörssikurssien hakupalvelulla on kaksi palveluntarjoajaa . Molemmat palveluntarjoajat tarjoavat samanlaisen rajapinnan. Ensisijaisen palveluntarjoajan vastausaika vaihtelee kutsukertojen välillä ja voi ajoittain olla korkea, mutta sen käyttö on edullista. Toissijainen palveluntarjoaja takaa alhaisen vastausajan, mutta sen käyttö on kertaluokkaa kalliimpaa. Taulukossa 5.1 on kuvattu eri palveluntarjoajien palvelun laatuominaisuudet.

Ominaisuus	**Ensisijainen palveluntarjoaja**	**Toissijainen palveluntarjoaja**
vastausaika	90% =< 1 s, 10 % > 1 – 2 s	99% < 1 s
hinta	0,01€	0,10€

Taulukko 5.1: Palveluntarjoajien palvelun laatuominaisuudet.

Ajantasaisia pörssikursseja hakevan kanavatyypin vaatimuksena on, että kanavatyyppi pystyy valitsemaan palveluntarjoajan itsenäisesti ennalta määriteltyjen sääntöjen mukaisesti käyttämällä päätöksenteossa käyttäjän antamia raja-arvoja. Ensisijaista edullisempaa ja ajoittain hitaampaa palveluntarjoajaa käytetään siihen asti kunnes käyttäjän määrittelemän vastausajan raja-arvo ylittyy. Jotta ruuhka-aikoina voidaan taata alhaiset vastausajat suurimmalle osalle käyttäjistä, voidaan hyväksyä, että pörssikurssin arvo ei välttämättä ole ajantasainen. Vaatimusmäärittelyn pohjalta ongelma-alueen ratkaisevan kanavatyypin yhteiset vaatimukset ovat:

1. pystyy välittämään käyttäjän pyynnön palveluntarjoajalle,
2. mahdollistaa kahden eri palveluntarjoajan käytön,
3. sisältää välimuistikomponentin, josta aiemmin haetut pörssikurssit voidaan hakea,
4. kirjoittaa lokille kaikki sanomat myöhempää analyysia varten,

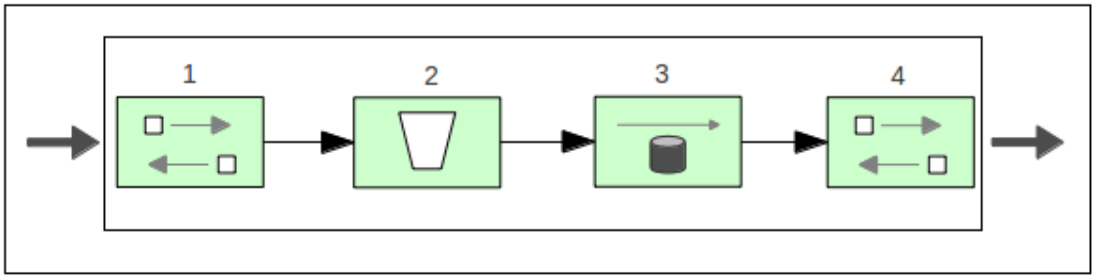
ja muunneltavuusvaatimukset:

5. kanava pystyy vaihtamaan ajonaikana palveluntarjoajaa itsenäisesti palvelun vastausajan ja käyttäjän antaman raja-arvon perusteella,
6. välimuistikomponentti on valinnainen ja käyttäjä voi valita välimuistin ele-

menttien TTL (time to live) arvon,

7. palvelulta toivottavaa vastausajan ylärajaa voidaan muokata.

Yhteisten vaatimusten 1 – 4 perusteella suunnitteluvaiheessa syntyy kanavatyypin määritys. Kuvan 5.4 määrityksessä komponentit 1 ja 4 liittyvät pyynnön vastaanottamiseen ja sen välittämiseen palveluntarjoajalle. Komponentti 2 kirjoittaa sanomat lokille ja komponentti 3 on välimuisti.



Kuva 5.4: Kanavatyypin määritys (notaatio [HoW03]).

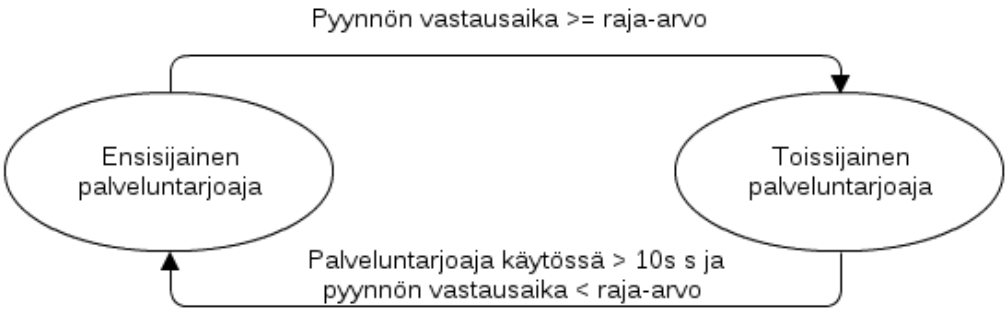
Kanavatyypin määrityksen ja muunneltavuusvaatimusten perusteella voidaan tunnistaa kanavatyyppiin liittyvät variaatiopisteet ja niihin liittyvät rajoitteet, jotka on esitetty taulukossa 5.2.

Variaatio-piste	**Tyyppi**	**Rajoite**	**Selitys**
useCache	Valinnainen	Kyllä/Ei	Totuusarvo, joka kertoo onko välimuisti käytössä.
cacheTTL	Parametri	Positiivinen kokonaisluku	Aika millisekunneissa, jonka välimuistissa oleva arvo on käyttökelpoinen.
responseTime-Limit	Parametri	Positiivinen kokonaisluku	Vastausajan raja-arvo millisekunneissa.
serviceProvider	Vaihtoeh-toinen	Ensisijainen/ Toissijainen	Kumpi vaihtoehtoisista palveluntarjoajista on käytössä.

Taulukko 5.2: Suorituskykyvaatimusten muutoksiin mukautuvan kanavatyyppiin liittyvät variaatiopisteet.

Kanavatyyppiin liittyvät mukautumissäännöt liittyvät käytössä olevan palveluntarjoajan valintaan. Kun kanava asennetaan ajoympäristöön, niin aluksi käytetään

ensisijainen palveluntarjoajaa. Palveluntarjoaja vaihdetaan toissijaiseen, jos pyynnön vastausajalle määritelty raja-arvo ylittyy. Ensisijainen palveluntarjoaja vaihdetaan takaisin, jos toissijaista on käytetty vähintään 10 sekuntia ja vastausaika on pienempi tai yhtä suuri kuin raja-arvo. Kuvassa 5.5 on kuvattu tilasiirtymäkaaviolla kanavatyyppiin liittyvät mukautumissäännöt.



Kuva 5.5: Mukautumisäännöt kuvaava tilasiirtymäkaavio.

Variaatiopisteiden ja kanavatyypin määrityksen pohjalta toteutusvaiheessa kuvataan kanavatyypin muunneltavien ominaisuuksien mallin XML-skeema (kuvaus 5.1). Muunneltavien ominaisuuksien malli voidaan ensin kuvata UML-luokkakaaviolla, jonka pohjalta generoidaan XML-skeema. Kuvauksessa 5.2 on esimerkki muunneltavien ominaisuuksien mallin ilmentymästä, jolle on asetettu variaatiopisteiden arvot cacheTTL = 800, responseTimeLimit = 1000 ja serviceProvider = PRIMARY.

```
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="stockQuoteChannelModel" type="stockQuoteChannelModel"/>
  <xs:complexType name="channelVariabilityModel" abstract="true">
    <xs:sequence>
      <xs:element name="id" type="xs:string" minOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="stockQuoteChannelModel">
    <xs:complexContent>
      <xs:extension base="channelVariabilityModel">
        <xs:sequence>
          <xs:element name="useCache" type="xs:boolean" minOccurs="1"/>
          <xs:element name="cacheTTL" type="xs:positiveInteger"/>
          <xs:element name="responseTimeLimit" type="xs:positiveInteger"
               minOccurs="1"/>
          <xs:element name="serviceProvider" type="stockQuoteServiceProvider"
               minOccurs="1"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:simpleType name="stockQuoteServiceProvider">
    <xs:restriction base="xs:string">
      <xs:enumeration value="PRIMARY"/>
      <xs:enumeration value="SECONDARY"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Kuvaus 5.1: Muunneltavien ominaisuuksien mallin XML-skeema.

```
<stockQuoteChannelModel xsi:noNamespaceSchemaLocation=
  "http://localhost:8080/schemas/StockQuoteChannel.xsd">
    <useCache>true</useCache>
    <cacheTTL>800</cacheTTL>
    <responseTimeLimit>1000</responseTimeLimit>
    <serviceProvider>PRIMARY</serviceProvider>
</stockQuoteChannelModel>
```

Kuvaus 5.2: Muunneltavien ominaisuuksien mallin ilmentymä.

Seuraavaksi toteutetaan kanavatyypin mallin ja muunneltavien ominaisuuksien mallin väliset molemmansuuntaiset transformaatiot. Transformaatiot toteutetaan XSL-kielellä tai Java-ohjelmointikielellä. Kuvauksessa 5.3 on kanavatyypin malli, joka syntyy kun transformaatiolle annetaan syötteenä kuvauksen 5.2 mukainen muunneltavien ominaisuuksien malli. Kuvauksen rivit 12 ja 18 liittyvät pyynnön vastaanottamiseen ja sen välittämiseen palveluntarjoajalle. Riveillä 4 – 9 on määritelty komponentit, jotka tekevät kutsun eri palveluntarjoajille. Välimuistikomponentti ja sen toiminnallisuus on määritelty riveillä 1 – 3, 14 – 17 ja 19.

```
<bean id="stockQuoteCache"
    <property name="cacheTTL" value="800" />
</bean>
<bean id="stockQuotePrimaryWSClient"
    <property name="responseTimeLimit" value="1000" />
</bean>
<bean id="stockQuoteSecondaryWSClient"
    <property name="responseTimeLimit" value="1000" />
</bean>
<camelContext xmlns="http://camel.apache.org/schema/blueprint">
    <route id="stockQuoteChannel">
            <from uri="vm:get.stockquote"/>
            <to uri="log:input" />
            <to uri="bean:stockQuoteCache?method=getQuote"/>
            <choice>
           <when>
               <el>${in.headers.cacheHit != 'true'}</el>
               <to uri="bean:stockQuotePrimaryWSClient" />
               <to uri="bean:stockQuoteCache?method=updateCache" />
           </when>
            </choice>
    </route>
</camelContext>
```

Kuvaus 5.3 Kanavatyypin mallin Camel-määrityksen mukainen kuvaus.

Viimeisenä toteutetaan kanavatyyppikohtainen mukautumiskäsittelijä mukautumissäännöt sisältävän tilasiirtymäkaavion pohjalta. Mukautumiskäsittelijä toteutetaan Java -ohjelmointikielellä. OpenChannel -kehykseen kuuluu komponentti, joka vastaanottaa tapahtumia jonon kautta ja välittää ne edelleen kanavatyyppikohtaiselle mukautumiskäsittelijälle. Jos mukautumiskäsittelijä käyttää jotain muuta mekanismia tapahtumien vastaanottamiseen, pitää se toteuttaa mukautumiskäsittelijän yhteydessä.

Kanavatyypin muuntelun testausta varten kanava asennettiin ajoympäristöön ja sitä käytettiin kolmella eri kanavatyypin variantilla: ensimmäisessä ajossa vastausajan raja-arvona käytettiin viittä sekuntia, toisessa ajossa raja-arvo oli yksi sekunti ja viimeisessä ajossa raja-arvona oli yksi sekunti ja lisäksi välimuisti otettiin käyttöön TTL-arvolla 900 millisekuntia. Jokaisessa testiajossa kanavalle lähetettiin 100 pyyntöä. Taulukossa 5.3 esitettyjen tulosten perusteella voidaan todeta, että kanavan käytös erosi merkittävästi eri testiajojen välillä.

Tunnusluku	Raja-arvo 5 s	Raja-arvo 1 s	Raja-arvo 1 s, välimuisti käytössä
vastausajan keskiarvo	1052 ms	962 ms	692 ms
vastausajan mediaani	774 ms	733 ms	560 ms
vastausajan keskihajonta	393 ms	609 ms	746 ms
pyyntöjen käsittelyaika	52,5 s	49,9 s	44,7 s
kutsuja ensisijaiselle palveluntarjoajalle	100 kpl	63 kpl	39 kpl
kutsuja toissijaiselle palveluntarjoajalle	0 kpl	37 kpl	24 kpl

Taulukko 5.3: Testiajojen tulokset.

Ensimmäisessä testiajossa kanava käytti kaikissa suorituksissa ensisijaista palveluntarjoajaa eikä muuntelua näin ollen tapahtunut. Toisessa testiajossa vastausajan keskiarvo 962 millisekuntia alitti raja-arvona olleen yhden sekunnin. Kolmannessa testiajossa välimuistin käytön ansioista vastausajat olivat keskimäärin selvästi alhaisempia kuin kahdessa ensimmäisessä ajossa ja 100 pyyntöä käsiteltiin lähes 10 sekuntia nopeammin kuin ensimmäisessä testiajossa. Toisessa ja kolmannessa testiajossa vastausajan keskihajonta oli selvästi suurempi kuin ensimmäisessä testiajossa, jossa kanavatyypin muuntelua ei tapahtunut. Luvussa 6.2 tutkitaan OpenChannel -kehyksen arvioinnin yhteydessä tarkemmin, että mikä on syynä muuntelun yhteydessä esiintyvään vastausaikojen suureen hajontaan.

6 OpenChannel -kehyksen arviointi

Luvussa 2.3 määriteltiin kuusi vaatimusta V1 – V6, jotka kanavatyyppien dynaamisen muuntelun mahdollistavan kehyksen tulisi täyttää. Vaatimukset voidaan jakaa toiminnallisiin vaatimuksiin, jotka liittyvät siihen mitä osia OpenChannel -kehyksen pitää sisältää ja miten sen pitää toimia ja ei-toiminnallisiin kehyksen laatuun liittyviin

vaatimuksiin. Jotta pystyttiin selvittämään miten OpenChannel -kehys täyttää vaatimukset suorituskykymuutoksiin, mukautuvaa kanavatyyppiä ajettiin oikeaa käyttöympäristöä simuloivassa testiympäristössä. Testiympäristöön asennettiin OpenChannel -kehyksen ja Apache ServiceMix -palveluväylän lisäksi kaksi palvelua, jotka käyttäytyivät samoin kuin taulukossa 5.1 määritellyt palveluntarjoajat. Lisäksi toteutettiin kuormageneraattori, jolla pyrittiin kuormittamaan palveluväylää tavalla, joka paljastaisi Apache ServiceMix -palveluväylän ja OpenChannel -kehyksen mahdolliset puutteet ja vahvuudet. OpenChannel -kehyksen käytettävyyden arviointia varten toteutettiin yksinkertaisen hallintasovelluksen, jolla käyttäjä pystyy mobiililaitteella tutkimaan ja muokkaamaan kanavatyyppien muunneltavia ominaisuuksia.

6.1 Toiminnalliset vaatimukset

Vaatimuksen V1 mukaan kehyksen tulee käyttää abstrakteja kanavatyypin muunneltavista ominaisuuksista johdettuja malleja dynaamisen muuntelun toteutukseen. Vaatimus toteutuu kehyksen käyttämien kanavatyyppien muunneltavien ominaisuuksien mallien avulla. Mallia käytetään suunnittelun aikana kuvaamaan kanavatyypin variaatiopisteet ja niiden rajoitteet sekä ajonaikaisesti hallitsemaan kanavatyypin muutoksia.

Vaatimus V2 liittyy kanavatyypin kykyyn tunnistaa ja reagoida kontekstin muutoksia. Kontekstin muutoksiin reagointia varten kehykseen on mahdollista toteuttaa kanavatyyppikohtaisia mukautumiskäsittelijöitä, jotka kykenevät vastaanottamaan tapahtumia kontekstin ja ympäristön muutoksista. Mukautumiskäsittelijät muokkaavat kanavatyyppien muunneltavien ominaisuuksien mallia ennalta määriteltyjen sääntöjen mukaisesti.

Jotta kanavatyyppien muunneltavien ominaisuuksien mallit olisivat löydettävissä ja muokattavissa, pitää mallien olla käsiteltävissä avoimen ja turvallisen rajapinnan kautta (vaatimus V3). Vaatimusta varten kehykseen toteutettiin REST-arkkitehtuurityylin mukainen rajapinta, joka sisältää operaatiot kaikkien palveluväylässä ajossa olevien kanavatyyppien listaukseen, yksittäisen kanavatyypin muunneltavien mallin hakemiseen ja sen muokkaamiseen. Rajapinnan väärinkäytön estämiseksi rajapinnan operaatiot on suojattu käyttämällä HTTP-perustodennusta. HTTP-perustodennus vaa-

tii, että jokainen pyyntö sisältää käyttäjätunnuksen ja salasanan.

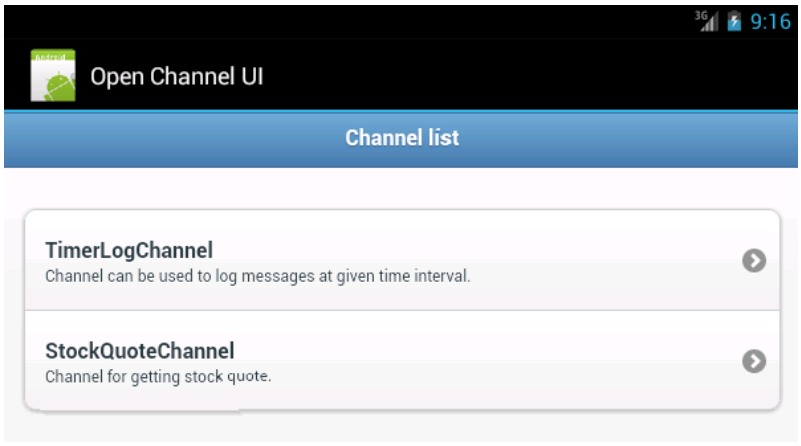
Reflektiivisen järjestelmän perusvaatimuksena on kausaalinen yhteys järjestelmän mallin ja sen toteutuksen välillä. OpenChannel -kehyksen yhteydessä kausaalinen yhteys tulisi toteutua kanavatyypin alustariippumattoman muunneltavien ominaisuuksien mallin ja alustariippuvaisen kanavatyypinn mallin välillä (vaatimus V4). Kausaalisen yhteyden takaamiseksi kehykseen toteutettiin luotettava synkronointimekanismi, jonka avulla kanavatyypin muunneltavien ominaisuuksien mallin muutokset heijastuvat kanavatyypin malliin ja myös toiseen suuntaan. Käytännössä kausaalisen yhteys toteutetaan jokaiseen kanavatyyppiin liittyvien transformaatioiden avulla. Transformaatiot suorittavat muunnokset muunneltavien ominaisuuksien mallin ja kanavatyypin mallin välillä. Palveluväylä huolehtii siitä, että kanavatyypin malli vastaa ajossa olevaa kanavaa.

Arvioinnin perusteella voidaan todeta, että OpenChannel -kehys toteuttaa kaikki sille asetetut toiminnalliset vaatimukset. Kehyksen avulla voidaan tukea turvallisesti ja monipuolisesti Apache ServiceMix -palveluväylään toteutettujen kanavatyyppien muuntelua. Muuntelua rajoittaa vain transformaatioiden kyky pitää kanavan toteutus ja kanavatyypin muunneltavien ominaisuuksien malli eheänä.

6.2 Laadulliset vaatimukset

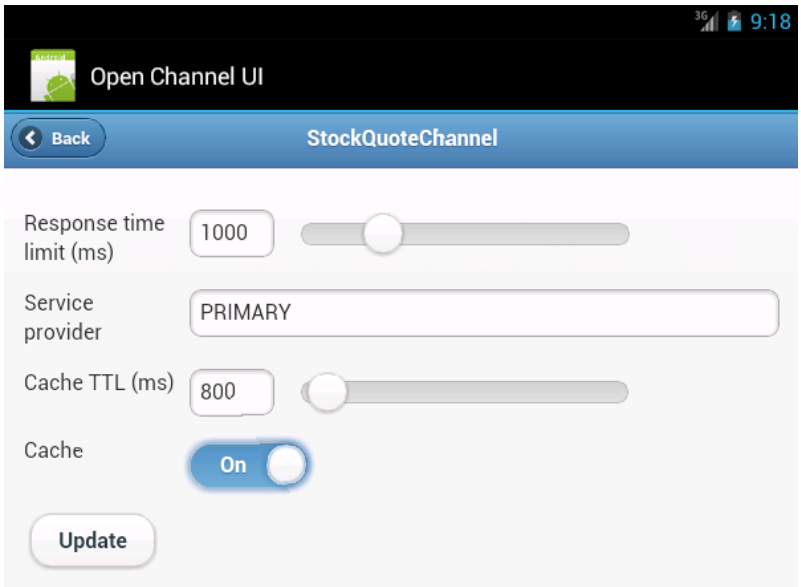
Vaatimuksen V5 mukaan kehyksen tulisi olla joustava ja helppo käyttää. OpenChannel kehyksen REST-arkkitehtuurityylin mukainen rajapinta toteuttaa monipuoliset mahdollisuudet erilaisten ihmis- ja konekäyttäjille tarkoitettujen asiakasohjelmien toteutukseen [GMT11]. Väitteen tueksi toteutettiin yksinkertaisen ja helppokäyttöisen mobiililaitteille suunnatun kanavatyyppien hallintasovelluksen.

Kanavatyyppien hallintasovelluksen aloitusnäytöllä listataan kaikki palveluväylässä olevat muunneltavat kanavatyypit kuvauksineen (kuva 6.1). Aloitusnäytön tiedot ja linkit kanavatyyppien kuvauksiin saadaan OpenChannel -kehyksen rajapintaan tehdyn GET-pyynnön vastauksella.



Kuva 6.1: Kuvankaappaus hallintasovelluksen aloitusnäytöstä.

Valittuaan kanavatyypin listauksesta käyttäjälle avautuu näyttö, joka sisältää kanavatyypin muunneltavat ominaisuudet ja niiden tämän hetkiset arvot (kuva 6.2). Vastauksella saadaan myös tieto siitä mistä arvojen rajoitteet sisältävän XML-skeeman löytää. XML-skeeman perusteella on helppo toteuttaa käyttötarkoitukseen sopivat käyttöliittymän komponentit. Esimerkissä käyttäjä voi säädellä positiivisten kokonaislukujen arvoja liukusäätimin ja totuusarvoa vivulla. Update-painikkeen painaminen lähettää kanavatyypin päivityspyynnön OpenChannel -kehykselle. Onnistunut päivitys näkyy näytöllä kuittauksena.



Kuva 6.2: Kuvankaappaus hallintasovelluksen kanavatyypin muokkausnäytöstä.

Vaatimuksen V6 mukaan kanavatyyppien dynaamisen muuntelun tukeminen ei saa vaikuttaa negatiivisesti palveluväylän suorituskykyyn. Lähtökohtaisena oletuksena oli, että ainoa suorituskykyvaikutus tulee olemaan sillä miten palveluväylä korvaa kanavatyypin muuntelun aikana vanhan kanavan uudella päivitetyllä versiolla. Erityisesti kiinnostavaa oli tietää mitä tapahtuu tilanteessa, jossa vanhassa kanavassa on vielä suorituksia kesken silloin, kun päivitys halutaan suorittaa. Apache ServiceMix -palveluväylän dokumentaatiosta ei selvinnyt millä mekanismilla kanavan päivitys tapahtuu. OpenChannel -kehyksen toteutuksen lähtökohtana oli, että kanavan päivitys tapahtuisi jollakin seuraavista tavoista:

1. Uusi ja vanha kanava toimivat rinnakkain sen aikaa, kun vanhassa kanavassa on vielä keskeneräisiä suorituksia. Uudet suoritukset ohjataan siirtymäaikana uudelle kanavalle ja vanha kanava suljetaan vasta, kun sen kaikki suoritukset ovat valmistuneet.
2. Uudet suoritukset jäävät odottamaan siksi aikaa, kun vanhan kanavan suoritukset ovat päättynet ja uusi kanava on käynnistetty.
3. Kun uusi kanava käynnistetään, niin vanha kanava suljetaan ja ajossa olevat suoritukset hylätään.

Suorituskykymuutoksiin mukautuvaa kanavatyyppiä toteutettaessa huomattiin jo varhaisessa vaiheessa, että Apache ServiceMix -palveluväylän kanavan käynnistys muutosten jälkeen ei toiminut millään edellä mainituista tavoista. Palveluväylä salli uusien pyyntöjen saapumisen kanavaan, jota oltiin jo päivittämässä uuteen versioon. Uusi kanava otettiin käyttöön vasta, kun vanhassa kanavassa ei ollut enää aktiivisia suorituksia. Jos kanavaan saapui tasaisesti pyyntöjä, niin pahimmassa tapauksessa kanava päivitettiin uuteen versioon vasta viiden minuutin aikarajan tultua täyteen, jolloin palveluväylä sulki vanhan kanavan ja siinä suorituksessa olleet pyynnöt hylättiin. OpenChannel -kehyksen kannalta ratkaisun ongelmaksi muodostui se, että kanavatyypin malli ei enää välttämättä vastannut kanava oikeaa tilaa eikä kausaalinen yhteys näin enää ollut voimassa. Yksinkertaisena ratkaisuna ongelmaan toteutettiin palveluväylään jonotuskomponentin, joka tutki jokaisen saapuvan pyynnön kohdalla onko uusi kanava käynnistymässä ja asetti siinä tapauksessa pyynnön

odottamaan siihen asti kunnes vanha kanava oli suljettu ja uusi kanava oli käynnistynyt. Tässä ratkaisussa suorituskyvyn kannalta tärkeintä tulee olemaan se kuinka nopeasti palveluväylä pystyy käynnistämään uuden kanavan ja pyynnöt pääsevät jatkamaan käsittelyään kanavassa.

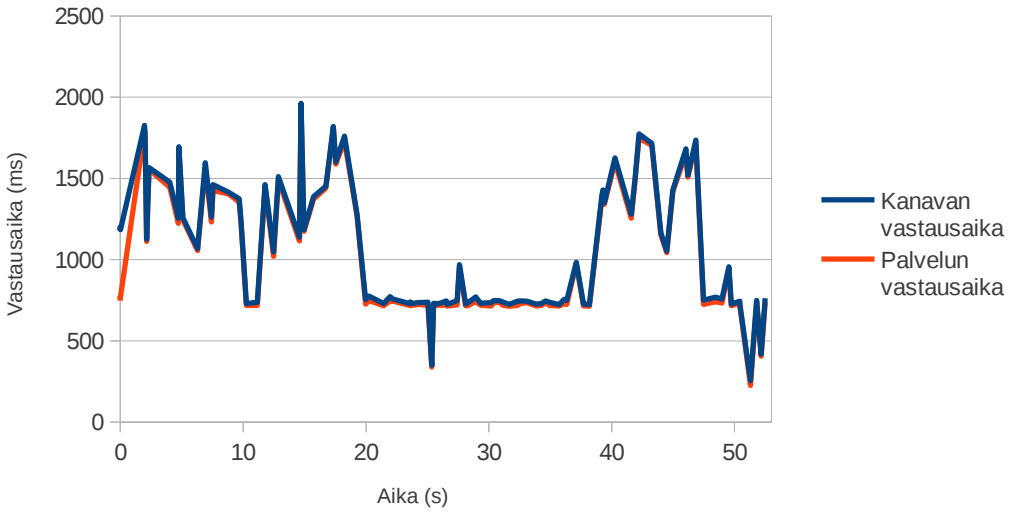
Tieto kanavan päivityksen alkamisesta ja päivityksen valmistumisesta olisi mahdollista välittää kanavan käyttäjälle, jolloin käyttäjä voisi päättää miten tilanteessa toimitaan. Käyttäjä voisi esimerkiksi itse odottaa uuden kanavan käynnistymistä tai käyttää toista kanavaa siirtymäajan aikana.

Kanavatyypin muuntelun aiheuttamien vaikutuksien tutkimista varten luvussa 5.4 esitellyllä suorituskykymuutoksiin mukautuvalla kanavatyypillä ajettiin aluksi kaksi testiajoa. Ensimmäisen testiajon tarkoituksena oli nähdä miten kanavatyyppi käyttäytyy silloin kun muuntelua ei tapahdu. Se toimisi vertailukohtana toiselle testiajolle, jossa muuntelua pitäisi tapahtua. Testikoneen suorittimena oli Phenom II X4 955, keskusmuistia 6 GB ja tallennuslaitteena oli SSD-levy. Käyttöjärjestelmänä testikoneessa oli Ubuntu 12.04 32-bit Linux 3.2.0-32-generic-pae ja sovelluksia ajettiin Oracle:n JDK versiolla 1.6.0_32. Testissä käytetty Apache ServiceMix palveluväylän versio oli 4.4.2.

Testissä käytetyllä kuormageneraattorilla pyrittiin kehittämään kuorma, jolla nähtäisiin miten Apache ServiceMix -palveluväylä ja OpenChannel -kehys käyttäytyvät kanavatyypin muuntelun aikana, kun useita pyyntöjä on suorituksessa samanaikaisesti. Testiajon aluksi kuormageneraattori käynnisti neljä säiettä, joista jokainen lähetti pyynnön palveluväylälle ja vastauksen saatuaan odotti 800 – 1500 millisekuntia ennen uuden pyynnön lähettämistä. Jokainen säie lähetti yhteensä 25 pyyntöä, joten pyyntöjä lähettiin yhteensä 100. Jokaiselle pyynnölle mitattiin kanavan vastausaika, joka sisälsi kanavassa käytetyn ajan sekä kanavan kutsuman ulkoisen palvelun vastausajan. Lisäksi mitattiin erikseen ulkoisen palvelun vastausaika, jotta pystyttiin erottamaan kanavan osuus vastausajasta. Suorituksesta otettiin talteen myös tieto siitä kutsuiko kanava ensi- vai toissijaista palveluntarjoajaa.

Ensimmäisen testiajon aikana palvelun vastausajan raja-arvo oli viisi sekuntia ja ajo kesti näillä asetuksilla 52,5 sekuntia. Kuten kuvan 6.3 kaaviosta voidaan havaita palvelun vastausajat olivat 250 – 2000 millisekunnin välillä. Koska vastausajan raja-arvo ei ylittynyt, niin ensisijaista palveluntarjoajaa käytettiin koko testiajon ajan eikä

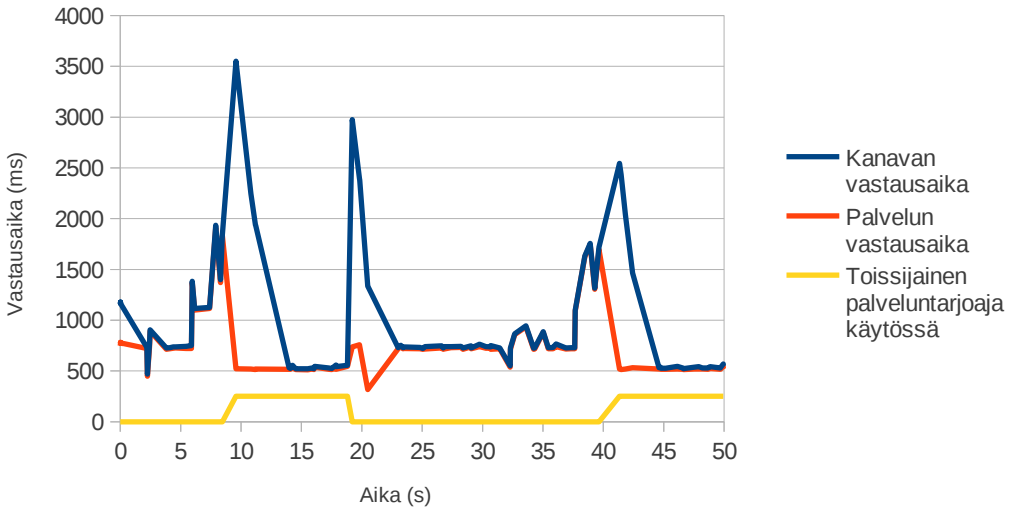
muuntelua siten tapahtunut. Kanavan vastausaika seurasi muutamaa ensimmäistä pyyntöä lukuun ottamatta palvelun vastausaikaa 10-20 millisekunnin erolla. Palveluväylän lokitietojen perusteella selvisi, että muutaman ensimmäisen pyynnön viive aiheutui kanavan käyttämien SOAP -pohjaisten Web Service -komponenttien toteutuksien generoinnista.



Kuva 6.3: Vastausajat vastausajan raja-arvolla 5000 ms.

Toisessa testiajossa palvelun vastausajan raja-arvoksi asetettiin 1000 millisekuntia, jolloin odotin edellisen testiajon tuloksen perusteella kanavatyypin muuntelua jo tapahtuvan. Kuvan 6.4 kaaviosta havaitaan, että palvelun vastausajan vaihtelivat tässäkin ajossa 250 – 2000 millisekunnin välillä. Kanavatyypin muuntelua tapahtui kolme kertaa 49,9 sekuntia kestäneen testiajon aikana. Ensimmäisen kerran vastausajan raja-arvo ylittyy kuuden sekunnin kohdalla. Kanavatyyppiin liittyvä mukautumiskäsittelijä reagoi raja-arvon ylitykseen muokkaamalla kanavatyypin muunneltavien ominaisuuksien malliin palveluntarjoajaksi toissijaisen palveluntarjoajan. Tämä näkyy kaaviossa piikkinä kanavan vastausajoissa, koska uudet pyynnöt joutuvat odottamaan suoritukseen pääsyä vanhan kanavan suoritusten valmistumiseen ja uuden kanavan käynnistymiseen asti. Seuraava muuntelu tapahtui 16 sekunnin kohdalla, kun mukautumiskäsittelijä vaihtoi ensisijaisen palveluntarjoajan takaisin, koska toissijaista palveluntarjoajaa oli käytetty mukautumissäännössä mää-

ritellyn 10 sekunnin ajan. Testiajon loppuvaiheessa vastausajan raja-arvo ylittyi vielä kerran, jolloin mukautumiskäsittelijä vaihtoi käyttöön toissijaisen palveluntarjoajan.



Kuva 6.4: Vastausajat vastausajan raja-arvolla 1000 ms.

Testiajon tulosten mukaan kanavatyypin muuntelu aiheutti sen, että 12 prosenttia pyynnöistä joutui odottamaan uuden kanavan käynnistymistä. Odotusajan keskiarvo oli 1700 millisekuntia. Tämän perusteella ratkaisua, jossa uudet suoritukset joutuvat odottamaan vanhassa kanavassa olevien suorituksien valmistumista ei voi suositella ympäristöihin, joissa on tiukat vastausaikavaatimukset. Tiukkoja vastausaikarajoja vaativiin ympäristöihin kanavatyypin päivittäminen olisi syytä toteuttaa sillä tavalla, että uusi ja vanha kanava olisivat molemmat käytössä uuteen kanavaan siirtymisen aikana. Tämän tutkielman yhteydessä ei kuitenkaan lähdetty tutkimaan miten se olisi Apache ServiceMix -palveluväylään toteutettavissa. Apache ServiceMix on aktiivisen kehityksen alla, joten on hyvin mahdollista, että palveluväylän tulevissa versioissa kanavien käynnistyminen on toteutettu tavalla, joka soveltuu paremmin OpenChannel -kehyksen käyttöön.

Ympäristöissä, joissa muuntelua tapahtuu harvemmin tai pyyntöjen saapumistiheys on alhaisempi kuin testiajossa, muuntelun ei pitäisi aiheuttaa merkittävää suorituskyvyn heikentymistä. Väitteen tueksi ajettiin vielä kolmas testiajo. Viimeisessä testiajossa kaikki muut parametrit pysyivät samoina kuin toisessa ajossa, mutta pyyn-

töjen lähettämisen aikaväliä kasvatettiin niin, että 100 pyyntöä lähetettiin viisi kertaa pidemmällä aikavälillä eli noin viiden minuutin aikana. Tässä testiajossa muuntelua tapahtui 10 kertaa ja vain kolme prosenttia pyynnöistä joutui odottamaan uuden kanavan käynnistymistä.

Laadullisten vaatimusten täyttymisen arvioiminen on haastavaa, koska OpenChannel -kehyksen käytös riippuu toteutettavan kanavatyypin monimutkaisuudesta ja siitä millaisessa ympäristössä kanavaa käytetään. Helppokäyttöisyysvaatimus toteutuu kehyksen tarjoaman rajapinnan kautta, joka mahdollistaa monipuoliset mahdollisuudet kanavatyyppien muunneltavien ominaisuuksien mallien tutkimiseen ja muokkaamiseen. Testiajojen perusteella suorituskykyvaatimukset eivät täysin toteudu. Kehys ei nykyisellään sovellu järjestelmiin, joissa vastausaikavaatimukset ovat tiukat, kuten reaaliaikajärjestelmissä.

7 Yhteenveto

Palvelukeskeinen arkkitehtuuri on hajautettujen järjestelmien suunnittelu- ja toteutusmalli, joka on kehittynyt nykyaikaisten ketterien liiketoimintaprosessien tarpeisiin. Palveluväylä tarjoaa työkalut ja ajonaikaisen infrastruktuurin palvelukeskeisen arkkitehtuurin toteuttamiselle. Ympäristön, palvelutasosopimuksen ja käyttäjän vaatimusten muutokset aiheuttavat muutostarpeita, joihin palveluväylän pitäisi pystyä reagoimaan. Nykyiset palveluväyläratkaisut eivät kykene reagoimaan muutostarpeisiin, koska suuri osa palveluväylän toimintaa ja ominaisuuksia koskevista valinnoista ja asetuksista joudutaan suljetun rakenteen takia tekemään järjestelmän rakentamisvaiheessa tai ne vaativat palveluväylän uudelleen käynnistyksen tullakseen käyttöön. Ratkaisuna ongelmaan tutkielmassa esiteltiin dynaamisesti muunneltavat kanavatyypit, jotka mahdollistavat kanavien rakenteen ja ominaisuuksien ajonaikaisen valinnan.

Tukeakseen kanavatyyppien dynaamista muuntelua palveluväylän ajoympäristöltä vaaditaan näkyvyyttä kanavatyyppien ominaisuuksiin ja rakenteeseen ja kykyä muokata niitä. Usein palveluväyläalustan arkkitehtuuri ei tarjoa tarvittavaa toiminnallisuutta, mutta palveluväylän avoimuutta voidaan lisätä toteuttamalla sen yhteyteen puuttuvan toiminnallisuuden tarjoava kehys. Tutkielman yhteydessä suun-

niteltiin ja toteutettiin OpenChannel -kehys, joka toteuttaa avoimen ja turvallisen rajapinnan Apache ServiceMix -palveluväylässä ajossa olevien kanavatyyppien rakenteeseen ja sitä kautta tuen dynaamiselle muuntelulle. Apache ServiceMix -palveluväylän käyttämä OSGi-pohjainen ajoympäristö mahdollistaa komponenttien lisäämisen, muokkaamisen ja poistamisen ajonaikaisesti ja teki siitä siten kehykselle sopivan kohdealustan.

OpenChannel -kehyksen suunnittelussa hyödynnetään reflektiivisten väliohjelmistojen ja muunneltavuuden hallinnan yhteydessä esiteltyjä menetelmiä. Kehys hyödyntää ajonaikaisesti reflektiota kanavatyyppien sen hetkisen rakenteen ja toiminnallisuuden tutkimiseen ja muokkaamiseen. Mallipohjaista lähestymistapaa hyödynnetään kehyksessä mallintamalla kanavatyyppien variaatiopisteet ja niihin liittyvät rajoitteet. Mallien käyttö tekee muuntelun suunnittelun ja ymmärtämisen helpommaksi ja mahdollistaa muutosten todentamisen ennen niiden käyttöönottoa.

OpenChannel -kehyksen arviointia varten toteutettiin suorituskykymuutoksiin mukautuva kanavatyyppi, jota ajettiin oikeaa ajoympäristöä simuloivassa ajoympäristössä. Testitulosten perusteella todettiin, että kehys toteutti kaikki sille asetetut toiminnalliset ja laadulliset vaatimukset lukuun ottamatta suorituskykyvaatimusta, joka ei täysin toteutunut. Testiajoissa havaittiin, että kanavatyypin muuntelu aiheutti vastausaikoihin viivettä, kun uusi kanava oli käynnistymässä. Suorituskyvyn todettiin kuitenkin olevan riittävä järjestelmiin, joissa suorituskykyvaatimukset eivät ole erityisen tiukat.

OpenChannel -kehyksen avulla toteutettuja dynaamisesti muunneltavia kanavatyyppejä voidaan hyödyntää palvelukeskeisessä arkkitehtuurissa palveluiden dynaamisen sidoksen muodostamisessa ja sidoksen elinkaaren hallinnassa. Kehyksen toiminnallisuuden ulkopuolelle rajattiin osa tutkielmassa kuvatun palveluiden dynaamisen sidonnan muodostamisen mallin vaatimista toiminnallisuuksista. Kehyksen toiminnallisuuden ulkopuolelle rajattiin sidoksen kuvaavien kanavatyyppien ominaisuuksien valintaan liittyvät mekanismit, sekä kanavatyyppien ja niiden muunneltavien ominaisuuksien mallien tallentamisessa käytettävän tyyppitietovaraston toiminnallisuus.

Dynaamisesti muunneltavien kanavatyyppien mahdollisuudet yksinkertaisissa käyttötapauksissa voitiin todeta jo tutkielman yhteydessä toteutetun suorituskykymuutoksiin mukautuvan kanavatyypin tapauksessa. Dynaamisesti muunneltavien kanavatyyppien todellinen hyödyllisyys nähdään kuitenkin vasta, kun OpenChannel -kehyksen avulla toteutetaan monimutkaisempia kanavatyyppejä, jotka kykenevät toimimaan osana dynaamista palvelukeskeistä arkkitehtuuria.

Lähteet

APS06 Andrade Almeida, J. P., L. Ferreira Pires, and M. J. Van Sinderen. "Abstract Platform and Transformations for Model-Driven Service-Oriented Development., Proceedings of the 2nd International Workshop on Model-Driven Enterprise Information Systems MDEIS 2006.

BAB01 G. Blair, G. Coulson, A. Andersen, L. Blair, M. Clarke, F. Costa, H. Duran-Limon, T. Fitzpatrick, L. Johnston, R. Moreira, N. Parlavantzas, K. Saikoski, The Design and Implementation of Open ORB 2. IEEE Distributed Systems Online, 2(6), 2001.

BBC05 N. Bencomo, G. Blair, G. Coulson, P. Grace, A. Rashid, Reflection and Aspects Meet Again: Runtime Reflective Mechanisms for Dynamic Aspects, OMD '05: Proceedings of the 1st workshop on Aspect oriented middleware development. New York, NY, USA, 2005. ACM Press.

BBD08 N. Bencomo, G. Blair, Dehlen, F. Fleurey, J.-M. Jézéquel, B. Morin, A. Solberg, An Aspect-Oriented and Model-Driven Approach for Managing Dynamic Variability. MoDELS'08: 11th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, Toulouse, France, 2008.

BBF08 Nelly Bencomo, Gordon Blair, Carlos Flores, Pete Sawyer, Reflective Component-based Technologies to Support Dynamic Variability. 30th International Conference on Software Engineering, Formal demos at ICSE, 2008.

BBG06 N. Bencomo, G. Blair, P. Grace, Models, Runtime Reflective Mechanisms and Family-based Systems to Support Adaptation. Workshop on Model Driven Development for Middleware (MODDM), 2006.

BBG08 N. Bencomo, G. Blair, P. Grace, P. Sawyer. Dynamically adaptive systems are product lines too: Using model-driven techniques to capture dynamic variability of adaptive systems. In 2nd International Workshop on Dynamic Software Product Lines (DSPL 2008), Limerick, Ireland, 2008.

BCG04 Blair, G., Coulson, G., Grace, P.: Research Directions in Reflective Middleware: the Lancaster Experience, Proc. 3rd Workshop on Reflective and Adaptive Middleware (RM2004), 2004, sivut 262-267.

BCR98 G.S. Blair, G. Coulson, P. Robin, M. Papathomas , An Architecture for Next Generation Middleware. Proc. IFIP Int'l Conf. Distributed Systems Platforms and Open Distributed Processing (Middleware'98), Springer-Verlag, New York, 1998, sivut 191-206.

BeB09 N. Bencomo, G. S. Blair, Using architecture models to support the generation and operation of component-based adaptive systems. In Proceedings of Software Engineering for Self-Adaptive Systems, 2009.

BGR05 Rainer Berbner, Tobias Grollius, Nicolas Repp, et al. , An approach for the Management of Service-oriented Architecture (SoA) based Application Systems . Enterprise Modelling and Information Systems Architectures – Proceedings of the Workshop in Klagenfurt, 2005.

Bos04 J. Bosch. Et al., COVAMOF: A Framework for Modeling Variability in Software Product Families. In Proceedings of the Third Software Product Line Conference (SPLC04), San Diego, CA, 2004.

CBM04 Grace, P., Coulson, G., Blair, G., Mathy, L., Yeung, W.K., Cai, W, Duce, D. Cooper, C., GRIDKIT: Pluggable Overlay Networks for Grid Computing. Proc. Distributed Objects and Applications (DOA'04), 2004.

CCG09 Cardellini, V., Casalicchio, E., Grassi, V., Presti, F.L., Mirandola, R., Towards self-adaptation for dependable service-oriented systems. In: Proc. of the Workshop on Architecting Dependable Systems, 2008.

CCM01 Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S., Web Services Description Language (WSDL) 1.1., 2001. http://www.w3c.org/TR/wsdl. [31.1.2012]

CGI07 M. Caporuscio, and N. Georgantas, V. Issarny, A Perspective on the Future of Middleware-Based Software Engineering, Future of Software Engineering 2007, L. Briand and A. Wolf (eds.), IEEE-CS Press, 2007.

CHR04 Luc Clement, Andrew Hately, Claus von Riegen, Tony Rogers. Universal Description, Discovery and Integration of Web Services (UDDI) 3.0.2, 2004. http://uddi.org/pubs/uddi_v3.htm. [30.1.2013]

Clo08 Clotet R. et al., Dealing with changes in service-oriented computing through integrated goal and variability modeling. In Second International Workshop on Variability Modelling of Software-intensive Systems (VAMOS 2008), Essen, Germany, 2008, sivut 43–52.

CFR11 R. Corchuelo, R. Z. Frantz, A. M. Reina-Quintero, A domain-specific language to design enterprise application integration solutions. International Journal of Cooperative Information Systems, 20(2), 2011, sivut 143–176.

FiT02 Roy T. Fielding, Richard N. Taylor. Principled Design of the Modern Web Architecture. University of California, Irvine, ACM Transactions on Internet Technology, Vol. 2, No. 2, 2002, sivut 115–150.

GGL02 Blair Gordon, S. Coulson Geoff, Blair Lynne, Duran-Limon Hector, Grace Paul, Moreira Rui, Parlavantzas Nikos. Reflection, Self-Awareness and Self-Healing in OpenORB. WOSS '02: Proceedings of the first workshop on Self-healing systems, 2002.

GJJ97 M. Griss, I. Jacobson, P. Jonsson. Software Reuse. Architecture, Process and Organization for Business Success. Addison-Wesley, ISBN: 0-201-92476-5, 1997.

GMT11 D. Guinard, F. Mattern, V. Trifa, E. Wilde. From the Internet of Things to the Web of Things: Resource Oriented Architecture and Best Practices. Architecting the Internet of Things, luku 5, 2011.

HLM07 Gang Huang, Xuanzhe Liu, Hong Mei. Towards Dependable Service-Oriented Architecture via Reflective Middleware, International Journal of Simulation and Process Modeling, Vol. 3, No. 1/2, 2007, sivut 55–65.

HoW03 Gregor Hohpe and Bobby Woolf, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions (Addison-Wesley Signature Series (Fowler) Addison-Wesley Sign), 2003.

Kut98 L. Kutvonen. Trading services in open distributed environments. PhD thesis, Department of Computer Science, University of Helsinki, 1998.

Liq12 Chen Liqiang, Integrating Cloud Computing Services Using Enterprise Service Bus (ESB). Business and Management Research, Vol. 1, No. 1, 2012, sivut 26-31.

LMT11 Peter F. Linington, Zoran Milosevic, Akira Tanaka, Antonio Vallecillo, Building Enterprise Systems with ODP: An Introduction to Open Distributed Processing, Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series, 2011, sivut 8-22.

Mae87 Pattie Maes, Concepts and Experiments in Computational Reflection. OOPSLA '87: Conference proceedings on Object-oriented programming systems, languages and applications, ACM Press, New York, NY, USA, 1987, sivut 147-155.

Men07 Falko Menge, Enterprise Service Bus. Free and Open Source Software Conference, 2007.

MSK04 P. K. McKinley, S. M. Sadjadi, E. P. Kasten, B. H. C. Cheng, Composing Adaptive Software. IEEE Computer, 37(7), 2004.

ODP97 ITU-T Rec. X.950, Open Distributed Processing – Trading function: Specification, 1997. http://www.itu.int/rec/T-REC-X.950/en. [25.1.2013]

ODP98a ITU-T Rec. X.901, Open Distributed Processing – Reference Model: Overview, 1998. http://www.itu.int/rec/T-REC-X.901/en. [25.1.2013]

ODP98b ITU-T Rec. X.930 | ISO/IEC 14753:1999, Open Distributed Processing - Interface References and Binding, 1998. http://www.itu.int/rec/T-REC-X.930/en. [25.1.2013]

OSG11 Open Services Gateway Initiative, OSGi Service Platform Core Specification, Version 4.3, April 2011. http://www.osgi.org/Specifications/HomePage. [27.4.2012]

Pap03 Mike P. Papazoglou , Service -Oriented Computing: Concepts, Characteristics and Directions. Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE'03).

Ser13 Apache ServiceMix, http://servicemix.apache.org. [23.3.2013]

SHL05 Marc-Thomas Schmidt, Beth Hutchison, Peter Lambros, Rob Phippen. The Enterprise Service Bus: Making Service-oriented Architecture Real. IBM SYSTEMS JOURNAL, VOL 44, NO 4, 2005.

Ven02 N. Venkatasubramanian, Safe 'Composability' of Middleware Services, Comm. ACM, June 2002, sivut 49-52.

Wee06 Weerawarana S., et al., Web Services Platform Architecture. Prentice Hall, 2006, sivut 3-22.

Liite 1. OpenChannel -kehyksen toteutus

Tutkielman yhteydessä toteutetun OpenChannel -kehyksen ja suorituskykymuutoksiin mukautuvan kanavatyypin lähdekoodit ovat ladattavissa osoitteesta https://github.com/tbertell/open-channel/archive/master.zip.

Projektin rakenne on seuraava:

```
/open-channel           päätaso
  /camel-channel-types  kanavatyyppien toteutukset
  /channel-models       mallit ja transformaatiot
  /open-channel-core    kehyksen ytimen toteutus
  /open-channel-ui      Android käyttöliittymän prototyyppi
```

OpenChannel -kehyksen keskeiset luokat vastuineen on listattu seuraavassa taulukossa.

Moduuli	**Java-luokka**	**Vastuu**
open-channel-core	AdaptationManager.java	Vastaanottaa tapahtumia ja välittää ne oikealle mukautumiskäsittelijälle.
open-channel-core	AuthenticationHandler.java	Toteuttaa rajapinnan kutsujen todennuksen.
open-channel-core	QueueListener.java	Kuuntelee jonoa, jonka kautta kehys vastaanottaa tapahtumia.
open-channel-core	ChannelResource.java	Kanavatyyppien REST-rajapinta.
open-channel-core	ChannelManager.java	Sisältää kanavatyyppien käsittelyn toiminnallisuuden.
open-channel-core	AdaptationPolicy.java	Rajapintaluokka, joka kaikkien mukautumiskäsittelijöiden pitää toteuttaa.
channel-models	ChannelVariabilityModel.java	Kaikkien mallien yhteinen yläluokka.
channel-models	ModelCamelXslTransformer.java	XSL-transformaatioden toteutusluokka.
channel-models	ModelTransformer.java	Transformaatioiden rajapintaluokka.