Leena Vasenius

XSLT-muunnokset DataPower-integraatioissa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

19.11.2014

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Leena Vasenius XSLT-muunnokset DataPower-integraatioissa 44 sivua + 5 liitettä 19.11.2014
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Jukka Porkka, Projektipäällikkö Erja Nikunen, Yliopettaja

Tämän insinöörityön aiheena oli tarkastella XSLT-muunnoskielen käyttötarkoituksia IBM WebSphere DataPower -laitteella toteutetuissa järjestelmäintegraatioissa. Insinöörityössä kuvataan XSLT:n mahdollisia rooleja DataPower-integraatiolaitteessa ja laitteella toteutetuissa integraatioissa.

Työn alussa esitetään, mitä XSLT ja integraatiokehitys ovat, miten integraatioita toteutetaan sekä mitä DataPower-laite tarjoaa integraatiokehitykselle. Työssä käydään läpi teoriatasolla XSLT:n ja DataPowerin ominaisuuksia sekä keskeisiä XSLT-kielen laajennuksia, joita DataPower-laite tukee.

Käytännön osuus keskittyy kuvaamaan DataPowerin sanomamuunnosprosessia XSLT:llä. Työssä selvitetään, minkälaisia sanomia XSLT voi muuntaa ja miten sanomamuunnosprosessia voi tarkastella. Lopussa kuvataan vielä parhaita toimintatapoja XSLT-ohjelmoinnissa ja DataPower-laitteella toteutettujen integraatioiden konfiguroinnissa.

Työn ohessa on toteutettu DataPower-laitteella kaksi integraatiota, joilla voi havainnollistaa XSLT-muunnoksia yksinkertaisen SOAP-sanoman muuntamisessa. Toinen integraatio havainnollistaa, kuinka SOAP-sanoma muutetaan XML-sanomaksi ja toinen integraatio muuntaa SOAP-sanoman positiopohjaiseksi COBOL-sanomaksi.

Avainsanat	XSLT, DataPower, SOA-arkkitehtuuri, EXSLT, integraatio, SOAP

Author(s) Title Number of Pages Date	Leena Vasenius XSLT Transformations in DataPower Integrations 44 pages + 5 appendices 19 November 2014
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Jukka Porkka, Project Manager Erja Nikunen, Principal Lecturer

The aim of this study was to observe how XSLT language can be used in integrations developed with the IBM WebSphere DataPower SOA Appliance. The study describes the possible roles of XSLT in the DataPower integration appliance and in the system integrations applied with the device.

The study begins with a description of what XSLT and integration development are, how integrations are done and what DataPower appliance can offer to integration development. The thesis also describes the properties of XSLT and DataPower as well as notable extension functions and elements in XSLT which are supported in the DataPower appliance.

The practical part of the study focuses on describing the message transformation process in the DataPower appliance with the transformations done with XSLT. The study explains what kind of messages can be transformed with XSLT and how the message transformations can be examined. The study also describes some of the best practices in XSLT programming and DataPower integration configuration.

Along with this thesis two integrations have been done with the DataPower appliance. These integrations can be used to demonstrate XSLT transformations on a simple SOAP message. The other integration demonstrates how a SOAP message can be transformed into an XML message while the other integration transforms SOAP message to COBOL message.

Keywords	XSLT, DataPower, SOA -architecture, EXSLT, integration, SOAP

Sisällys

Liitteet

Liite 1. DemoService-integraatioiden hakemistorakenne DataPower-laitteella

Liite 2. Pyynnön muuttujien asetus DataPowerilla

Liite 3. Vastauksen muunnos DemoServiceMPG-integraatiopalvelussa

Liite 4. Binäärimuunnokset

Liite 5. Binääripyynnön ja -vastauksen ffd-tiedostot

Lyhenteet

COBOL	Common Business Oriented Language. Positiopohjainen ohjelmointikieli, jota on käytetty kaupallisten sovellusten kehittämiseen.
DOM	Document Object Model. Malli, jolla määritetään muun muassa XML- ja HTML-dokumenttien puumainen rakenne ja jonka prosessoinnin mukaan sanoma käsitellään kokonaisuudessaan.
ESB	Enterprise Service Bus. Sovellusarkkitehtuuri, jolla kuvataan kommunikaatiota eri sovelluksien välillä.
FFD	Flat File Descriptor. XML:n kaltainen metadatan kuvauskieli, joka kuvaa sisään tulevan ja ulosmenevän datan rakennetta.
HTML	Hypertext Markup Language. Verkkosivujen kuvaamiseen tarkoitettu kieli.
HTTP	Hypertext Transfer Protocol. Sovellustasolla toimiva verkkoprotokolla, jota käytetään tiedon siirtoon selaimissa ja verkossa.
JDBC	Java Database Connectivity. Javaan perustuva teknologia, jolla päästään tekemään kyselyjä ja päivitysoperaatioita tietokantaan.
JSON	JavaScript Object Notation. JavaScript-kieleen pohjautuva kieliriippumaton kuvauskieli, jota käytetään tiedonsiirtoon.
MOM	Message-oriented middleware. Infrastruktuuri, jolla tuetaan sanomien lähetystä ja vastaanottamista erilaisten hajautettujen järjestelmien välillä.
MQ	Message Queue. Sanomajono, johon järjestelmät ja sovellukset voivat tallentaa sanomia siksi aikaa, kunnes vastaanottava järjestelmä hakee sanoman. Tällaisia sanomajonosovelluksia ovat muun muassa IBM WebSphere MQ ja Java Messaging Service.
SAX	Simple API for XML. Malli, jonka prosessoinnin mukaan sanoma käsitellään node kerrallaan.

SOA	Service-oriented architecture. Suunnittelumalli, jolla kuvataan sovellustoimintoja palveluina toisille sovelluksille.
SOAP	Simple Object Access Protocol. XML-kieleen perustuva protokolla, jolla voidaan vaihtaa rakenteellista tietoa eri www-sovelluspalvelujen välillä verkossa.
URL	Uniform resource locator. Osoitetieto, jolla haetaan tietoa jostakin resurssista. Käytetään yleensä WWW-sivujen osoitteissa.
XML	eXtensible Markup Language. Kuvauskieli, jolla kuvataan datan rakennetta ja jäsennetään sitä.
XSL	eXtensible Stylesheet Language. XML-kuvauskielelle kehitetty tyylimäärityskieli. Siihen sisältyy kolme osaa: XSLT (muuntaminen), XPath (navigointi) ja XSL-FO (muotoilu).
XSLT	eXtensible Stylesheet Language Transformations. Kieli, jota käytetään muuntamaan XML-dokumentteja toiseen muotoon.
WSDL	Web Services Description Language. XML-kieleen perustuva kuvauskieli, jolla kuvataan www-sovelluspalvelujen toiminnallisuuksia.
WTX	Websphere Transformation Extender. IBM:n tuoteperhe, jolla voidaan tuottaa esimerkiksi binäärimuunnostiedostoja, WTX-kuvaustiedostoja, joita voidaan ajaa muun muassa DataPower-laitteella.

1 Johdanto

Tietoteknisten ratkaisujen yleistyessä myös erilaisia järjestelmiä tulee lisää. Uusia järjestelmiä luodaan, mutta vanhoja järjestelmiä on yhä käytössä. Samaan aikaan on tarve saada eri järjestelmät välittämään tietoa toisillensa. Näihin tarpeisiin vastataan integraatiopalveluilla, jotka tekevät tarpeelliset sanomamuunnokset eri järjestelmien välillä ja jotka reitittävät sanomat oikeisiin järjestelmiin.

XML-sanomia käytetään sovellusten ja järjestelmien välisessä tiedonvälityksessä. XSLT-kieltä käytetään XML-tiedostojen muuntamiseen eri muotoon riippuen kuluttajasovelluksesta. Usein XML-tiedostot ovat isoja ja sen vuoksi raskaita, jolloin XSLT-muunnos vie aikaa. Integraatiokehityksessä ja XML-dokumenttien muuntamisessa IBM WebSphere DataPower -laite on yksi XSLT-muunnosten prosessoija, joka nopeuttaa muunnosten prosessointia. Laitteeseen on kehitetty myös tehokkaat tietoturvaominaisuudet ja samaisesta laitteesta on kehitetty myös integraatioihin keskittyvä malli, joka sisältää sekä tietoturvaominaisuudet että XSL-prosessorin. Laitetta on käytetty muun muassa toteuttamaan integraatioita eri järjestelmien välillä.

Opinnäytetyö jakaantuu viiteen osaan. Ensimmäiseksi työssä esitetään yleisellä tasolla, mitä integraatiokehitys on ja mitä siihen liittyy. Toiseksi esitellään DataPower-laite ja XSLT-muunnoskieli. Kolmas osuus keskittyy kuvaamaan, miten käytännössä XSLT-muunnoskieltä on käytetty DataPower-laitteella toteutetussa integraatiossa. Tässä osiossa kuvataan kaksi käytännön toteutusta, joilla voidaan havainnollistaa XSLT:n eri käyttötarkoituksia DataPower-integraatiossa. Neljäs aihealue käsittelee hyviksi todettuja toimintatapoja niin XSLT-ohjelmoinnissa kuin DataPower-laitteen integraatioiden konfiguroinnissa. Lopuksi tehdään yhteenveto työstä.

Työ ei ota kantaa perinteiseen sovelluskehitykseen, XML-kieleen, XSL:n muihin kieliin eikä DataPower-laitteen muihin ominaisuuksiin, kuten tietoturvamäärityksiin tai ylläpitoon.

2 Integraatiokehitys

2.1 Tarpeet integraatioille

Yleensä järjestelmien välinen kommunikointi perustuu perinteiseen ohjelmointimalliin, jossa sovelluksille ohjelmoidaan suoraan kommunikointitapa. Sovellukselle määritetään kaikki tarvittavat yhteydet niihin järjestelmiin, joihin sen pitää olla yhteydessä. Tätä kutsutaan point-to-point-tyyliseksi integraatioksi. Esimerkiksi verkkokauppasovellus voi ottaa JDBC:llä suoran yhteyden yhteen tai useampaan tietokantaan ja tarkat yhteyden määritystiedot ovat sovelluksessa.

Järjestelmien ja sovellusten määrän kasvaessa myös sovelluksiin määritettävien yhteyksien määrä kasvaa. Perinteisellä ohjelmointimallilla tämä tarkoittaisi sitä, että sovellukseen on lisättävä aina uuden yhteyden määritykset ja sitä kautta itse sovellusta muutettava, jotta yhteys toimisi. Mikäli sovelluksen käyttämään taustajärjestelmään tulee muutoksia, myös sovelluksen yhteyttä on mahdollisesti muutettava ja muutokset voivat koskea kaikkia mahdollisia sovelluksia ja järjestelmiä, jotka ovat yhteydessä taustajärjestelmään [Davis 2009: 11-12].

Sovellusmuutoksista seuraa, että sovelluksille kirjoitetaan suuri määrä ylimääräistä koodia, jota on vaikea ylläpitää ja hallita. Lisäksi se voi aiheuttaa inhimillisistä syistä johtuvia tietoturvariskejä. Myös muut tietoturvariskit ovat mahdollisia ilman kunnollisia palomuureja ja tarkistuksia järjestelmien ja sovellusten välillä. Luotettavan ja toimivan datavälityksen ja sanomakulun turvaaminen korostuu nykypäivänä, kun järjestelmiä ja sovelluksia tulee lisää kasvavassa määrin.

Integraatiotarve on näkyvissä myös valtavan isoissa sovelluskokonaisuuksissa, joissa voi olla mukana satoja, tai jopa tuhansia, erilaisia sovelluksia, jotka on voitu toteuttaa eri teknologioilla eri alustoille eri toimijoiden toimesta ja vielä kaiken lisäksi eri vuosina. Kaikkien näiden sovellusten pitäisi muodostaa yksi iso toimiva kokonaisuus. Yhden suuren sovelluksen, joka hoitaisi kaikki nämä toiminnot, toteuttaminen on periaatteessa mahdotonta. Suuret sovellusmäärät on kuitenkin jollakin tavoin yhdistettävä, jotta ne tuottaisivat kokonaisuuden, jota kuluttajasovellukset käyttävät. Integraation on tarkoitus tarjota tällaisten sovellusten välille luotettava ja tehokas tiedonsiirto. [Solving Integration Problems using Patterns.]

Integraatio voidaan siis yleisellä tasolla nähdä keinona vastata tarpeisiin, joissa järjestelmien ja sovellusten välinen kommunikointi on taattava. Integraatio voidaan määritellä tavaksi saada eri järjestelmät kommunikoimaan toistensa kanssa.

2.2 Haasteet

Integraatiokehitykseen sisältyy monenlaisia haasteita, joihin integraatioiden pitäisi kyetä vastaamaan. Tässä luvussa kuvataan muutama haaste, jotka nousevat esille integraatiokehityksessä.

Osa haasteista liittyy sovelluksiin ja järjestelmiin. Useimmiten sovellukset ja järjestelmät tarvitsevat toisten sovellusten ja järjestelmien tietoja voidakseen toteuttaa jonkin toiminnon. Harvoin kaikki nämä toiminnot ovat yhdellä ja samalla tietokoneella, vaan eri sovelluksia ja järjestelmiä käytetään verkon yli. Näistä johtuen sovellusten ja järjestelmien väliseen tiedonvälitykseen sisältyviä haasteita ovat muun muassa verkon epäluotettavuus. Sovellukset eivät myöskään ole samanlaisia toistensa kanssa, ja niihin tulee muutoksia. Nämä tuovat mukanaan haasteet siitä, miten sovellukset saadaan kommunikoimaan toistensa kanssa ja miten sovellusmuutoksia voidaan hallita, sillä sovellukset voivat käyttää erilaisia tietoformaatteja ja muutokset yhdessä sovelluksessa voi pakottaa kaikki sen kanssa kommunikoivat sovellukset muuttumaan.

Sovellusten ja järjestelmien välinen tiedonvälitys voidaan toteuttaa esimerkiksi tiedonsiirroilla, jaetuilla tietokannoilla tai sanomavälityksellä [Enterprise Integration Patterns Introduction]. Integraatioiden tulisi kuitenkin kyetä vastaamaan myös seuraaviin haasteisiin [Ebbers ym. 2008: 3-4; J.Rodriguez ym. 2008: 2-3]:

- Käytettävyys: Koska järjestelmiä ja sovelluksia on valtavasti ja niiden pitäisi kyetä kommunikoimaan toistensa kanssa, järjestelmien ja sovelluksien välille täytyy luoda kommunikointiväylä. Yksittäisiä kommunikointimäärityksiä on vaikea hallinnoida sovelluskohtaisesti eikä käytettävyyttä paranna se, että sovelluskehityksessä suositaan uuden koodin tuottamista kuin olemassa olevien toiminnallisuuksien parantamista ja laajentamista.
- Turvallisuus: Sanomavälityksessä XML-hyökkäykset ovat yleisiä ja niihin olisi osattava varautua. Lisäksi turvallisuustoimenpiteitä, kuten autentikointia ja sanoman validointia vähennetään paremman suorituskyvyn vuoksi, mikäli suorituskyky on heikko.

- Suorituskyky: XML-sanomien prosessointi on raskasta. Kun siihen lisätään vielä tarvittavat muunnokset, joissa raskaita XML-tiedostoja käydään läpi, suorituskyky voi kärsiä.

Yksi keino vastata yllä mainittuihin integraatiohaasteisiin on käyttää esimerkiksi ESB-laitetta tai -sovellusta ja palomuureja. ESB:n käyttö on kuvattu luvussa 2.5.

2.3 Integraatiomallit

Riippuen integraatioiden ja sovelluksien ominaisuuksista ja asiakastoivomuksista voidaan käyttää eri integrointimalleja. Tässä luvussa esitetään muutama integraatiomalli.

Business-to-business (B2B) -menettelyllä voidaan luoda integraatioita yritysorganisaation ja sen ulkopuolella olevien organisaatioiden sovellusten välille. Tällaista integraatiotapaa kannattaa käyttää, kun jonkin toiminnon toteutumiselle tarvitaan usean eri yritysorganisaation järjestelmän toimintoja, jolloin data kulkee yritysorganisaatioiden palomuurien läpi toiselle yritysorganisaatiolle. [Solving Integration Problems using Patterns.] DataPower-laitteesta löytyy B2B-laitesarja, joka perustuu XB-sarjan laitteisiin ja keskittyy nimenomaan B2B-tyyppisiin integraatioihin [Kaplan ym. 2011: 178].

Tiedon kahdennukselle (Data Replication) voidaan nähdä tarvetta silloin, kun jokin sovellus muuttaa dataa ja datan muutos on päivitettävä myös muihin sovelluksiin. Esimerkiksi asiakastietoja voidaan tarvita ja säilyttää monissa eri sovelluksissa, kuten laskutus- ja asiakashallintasovelluksissa. Asiakas saattaa muuttaa tietojaan, jolloin tiedon on päivityttävä kaikkiin sovelluksiin, jotka hyödyntävät samoja tietoja. Käytännön integraatioissa tiedon kahdennustapa on voitu toteuttaa esimerkiksi kopioimalla tietokantaan tulevat muutokset päivitettäviin sovelluksiin tai välittämällä muuttuneet tiedot sanomien avulla toisille sovelluksille hyödyntämällä MOM-middlewarea. [Solving Integration Problems using Patterns.]

Sovelluksien käyttäjillä on hyvin usein tarve saada tietoa useasta järjestelmästä voidakseen toteuttaa jonkin toiminnon. Tietoportaalilla (Information Portal) voidaan kerätä yhteen ja samaan näkymään tai operaatioon useasta eri järjestelmästä ja sovelluksesta tietoja. Tällöin käyttäjän ei tarvitse erikseen käydä jokaisessa sovelluksessa erikseen voidakseen saada kaiken tarvitsemansa tiedon. [Solving Integration Problems using Patterns.]

SOA on hyvin yleinen tapa toteuttaa integraatioita. DataPower-laite tarjoaa tehokkaat työkalut SOA-arkkitehtuuriin perustuvien integraatioiden toteuttamiselle. SOA kuvataan tarkemmin luvussa 2.4.

2.4 SOA-arkkitehtuuri ja www-sovelluspalvelut

SOA (Service-oriented architecture) eli palvelukeskeinen arkkitehtuuri on alusta- ja sovellusriippumaton suunnittelumalli, jonka mukaan sovellukset kuvataan palveluina muille sovelluksille. SOA määrittää, miten palvelut, esimerkiksi kaksi eri sovellusta, kommunikoivat toistensa kanssa, jotta toinen voi suorittaa jonkin osan tarvittavasta työstä, mikä pitää tehdä [Rouse, 2008]. Koska arkkitehtuurissa palvelut ovat toistensa toteutuksesta riippumattomia, niiden välillä on löyhä sidonta. Tämä on yksi SOA-arkkitehtuurin eduista, sillä se mahdollistaa sovelluksiin tehtävät muutokset ilman, että toisten sovellusten toimintaa häiritään.

SOA-arkkitehtuuriin liitetään usein termi www-sovelluspalvelut (Web Services), koska SOAP-protokollaan perustuvat www-sovelluspalvelut ovat yksi yleisin SOA-arkkitehtuurin toteutuksista [Rouse 2008]. Ne ovat verkossa toimivia sovelluskomponentteja, jotka käyttävät standardisoitua XML-sanomavälitystä. Tämän vuoksi www-sovelluspalvelujen välinen kommunikointi on mahdollista sovelluskomponenttien teknologiasta, alustasta ja käyttöjärjestelmästä riippumatta. [What are Web Services.] Www-sovelluspalveluilla voidaan julkaista websovelluksen funktioita ja sanomia koko maailman käyttöön. Www-sovelluspalvelut mahdollistavat myös uudelleenkäytettävien sovelluskomponenttien tekemisen, sillä samaa www-sovelluspalvelua voi käyttää usea sovellus, sekä olemassa olevien sovellusten yhdistämisen, jotta sovellukset voivat vaihtaa keskenään dataa. [Introduction to Web Services.] Www-sovelluspalveluita suunnitellaan mahdollisuuksien mukaan SOA-arkkitehtuurin periaatteita käyttäen, koska silloin sovelluspalveluihin saadaan alustariippumatonta yhteistoimivuutta [Kodali 2005], mutta SOA ei vaadi, että kaikki palvelut ovat nimenomaan www-sovelluspalveluita. Käytännössä SOA-arkkitehtuurissa www-sovelluspalvelut toimivat arkkitehtuurin rakennuspalikoina, ja ne linkitetään toisiinsa liiketoiminnallisten prosessien mukaan [Lorenz 2006].

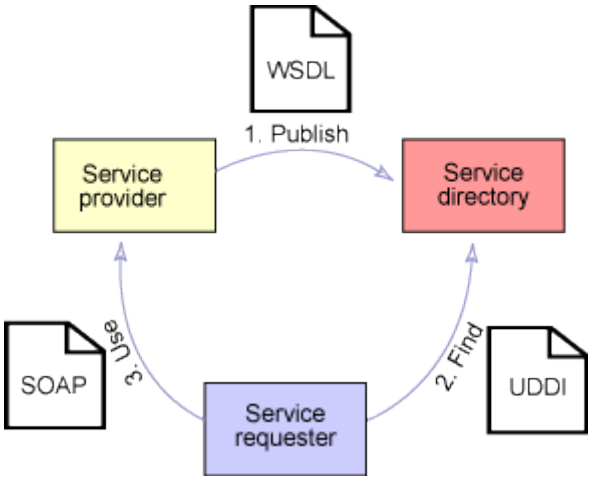
SOA-arkkitehtuurin sovelluskomponenttien välistä kommunikointia kuvataan erilaisilla kuvaustiedostoilla. Yksi tällainen on XML-kuvauskieleen perustuva WSDL-kuvaus, joka

kuvaa www-sovelluspalvelun, sen operaatiot ja palvelun sijainnin [Introduction to WSDL]. Se kertoo, mitä www-sovelluspalvelu tekee, miten ja kuinka sitä käytetään. Sillä voidaan vähentää kuluttajasovelluksiin tehtävää koodimäärää. [Snell ym. 2002: 81-82.]

WSDL-kuvauksia voidaan julkaista UDDI-rekisteriin (Universal Description, Discovery and Integration). UDDI on XML-pohjainen rekisteri, jonka kautta voidaan etsiä olemassa olevia www-sovelluspalveluja käytettäväksi ja julkaista www-sovelluspalveluja. Palveluntarjoajat julkaisevat palveluitaan rekisteriin, jotta kuluttajasovellukset voivat löytää ja käyttää niitä.

Sanomavälitys on mahdollista toteuttaa SOA:ssa monilla eri tavoilla, koska se on riippumaton protokollasta. SOAP-protokolla on kuitenkin yksi yleisimmin käytetty sanomavälitysprotokolla SOA:ssa, ja se toimii sanoman välittäjänä kuluttajasovelluksen ja palvelusovelluksen välillä [Kodali 2005]. SOAP (Simple Object Access Protocol) on XML-kieleen perustuva protokolla, jolla voidaan ottaa yhteyttä www-sovelluspalveluihin. Se mahdollistaa sovellusten välisen kommunikoinnin esimerkiksi HTTP:n tai MQ:n yli teknologiasta ja alustasta riippumatta. [SOAP Introduction.] Varsinkin yritystason sovelluksissa SOAP-sanomat ovat käytännöllisemmät kuin XML-sanomat, koska yritystason sovellusten sanomavälityksessä pitää varmistaa myös turvallisuus [Chase 2006].

Käytännössä SOAP, UDDI ja WSDL liittyvät toisiinsa kuvan 1 mukaisesti. Palveluntarjoaja julkaisee WSDL-kuvauksen palveluhakemistoon ja sitä myöten UDDI-rekisteriin. Palvelunkuluttaja eli kuluttajasovellus etsii UDDI:sta tarvitsemansa palvelun ja siten tietoonsa palvelun sijainnin. Kun palvelunkuluttaja tietää, mistä palvelua kutsutaan, voidaan käyttää SOAP:ia kutsumaan palveluntarjoajan palvelua. Sisäisen verkon sovelluksissa UDDI ei ole aina välttämätön, koska usein näissä tapauksissa palvelunkuluttajat tietävät jo ennestään, missä palveluntarjoaja on ja miten sitä kutsutaan. [Lorenz 2006.]



Kuva 1. SOAP:in, UDDI:n ja WSDL:n liittyminen toisiinsa [Lorenz 2006].

SOA-arkkitehtuuri on suosiossa, koska se on joustava ratkaisu isoihin järjestelmäarkkitehtuureihin. Sen avulla useat sovellustoiminnot voidaan tarjota kaikille muille palveluna, jolloin toisten sovellusten ei tarvitse toteuttaa kyseistä toiminnallisuutta uudestaan. Koska palvelut ovat löyhästi sidottuja toisiinsa, kuluttajasovellusten ei tarvitse tietää palvelun fyysistä sijaintia, jolloin palvelu voidaan tarvittaessa siirtää toiseen paikkaan häiritsemättä ja muuttamatta kuluttajasovelluksen toimintaa. Sovelluksiin ja järjestelmiin on helpompi tehdä muutoksia.

2.5 Sanomavälitys

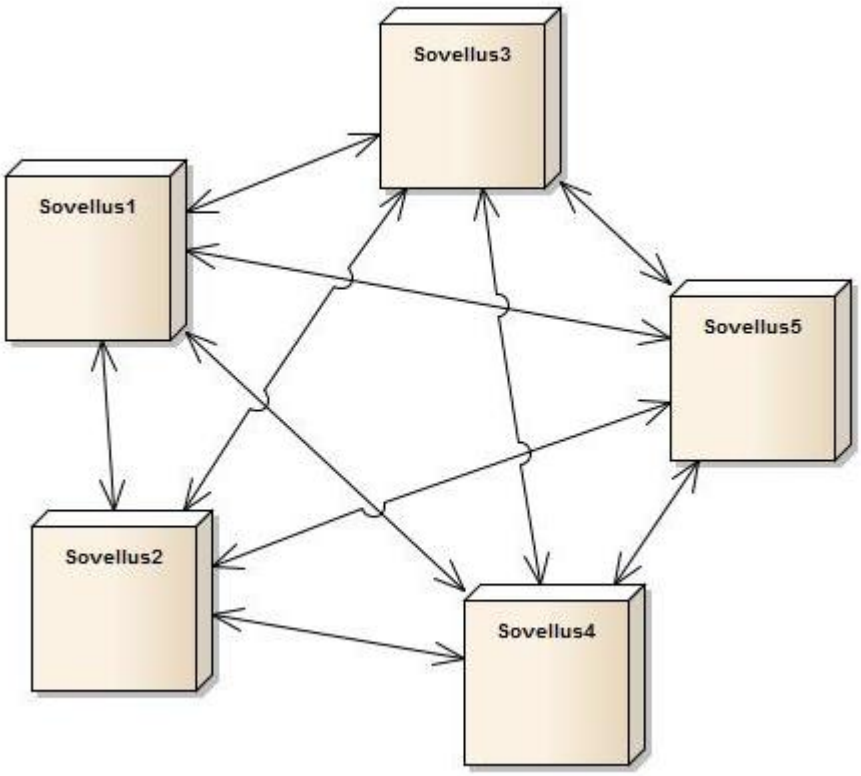
Jotta sovellukset ja järjestelmät voivat kommunikoida toistensa kanssa, niiden on välitettävä toisilleen dataa jollakin tavalla. Sanomien perusteella sovellukset voivat välittää toisilleen tarvitsemaansa dataa ja suorittaa tehtäviä, jotta haluttu toiminnallisuus saadaan aikaiseksi. [Enterprise Integration Patterns Introduction.] Sanomien avulla toteutuvaa datan välitystä on tuettu esimerkiksi MOM-middlewarella. Middleware eli väliohjelmisto on kaikki se, mitä on sovellusten välillä, ja MOM (Message-oriented middleware) on tarkoitettu tukemaan nimenomaan sanomilla toteutuvaa datan vaihtoa hajautettujen järjestelmien välillä. [Curry.]

ESB (Enterprise Service Bus) on väliohjelmistoille suunnattu sovellusarkkitehtuuri, jolla kuvataan kommunikaatiota eri sovelluksien välillä. ESB voi esimerkiksi muuntaa binäärimuotoisen sanoman XML-muotoon ja välittää sen eteenpäin. ESB kykenee hoitamaan monia asioita sovellusten välisessä kommunikoinnissa, ja siksi ne ovat hyvin

yleisiä komponentteja, jotka "sitovat yhteen" useat eri SOA-komponentit tai toistensa kanssa kommunikoivat sovellukset. [Davis 2009: 19-20.] ESB:n ydintoiminnallisuuksiin kuuluvat muun muassa:

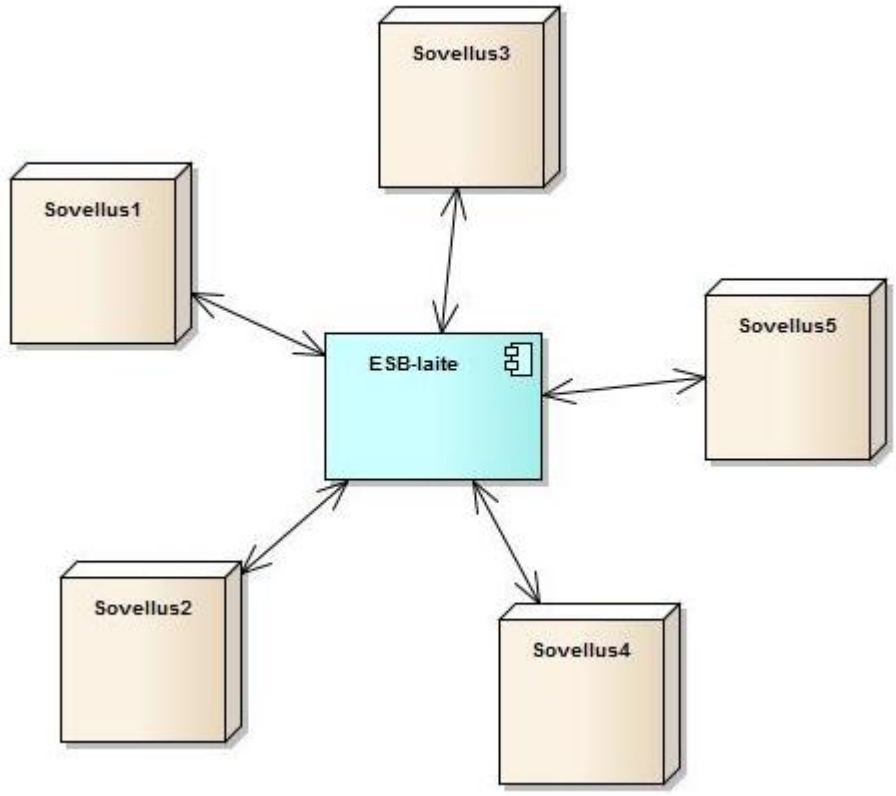
- adapterit, kuten HTTP-, FTP- ja JMS-yhteydet
- datamuunnokset (esimerkiksi XSLT)
- reititys
- ajastus
- monitorointi ja lokitus
- palvelun hallinta.

Yleinen ongelma useita järjestelmiä sisältävässä sovellusarkkitehtuurissa on se, miten hallinnoidaan järjestelmien väliset yhteydet. Mikäli järjestelmille määritetään suoraan yhteydet eri järjestelmiin, arkkitehtuuri voi helposti muistuttaa kuvan 2 tilannetta, varsinkin jos mukaan tulee uusi järjestelmä. Muutostilanteisiin tällainen point-to-point-tyyppinen integraatio järjestelmien välissä reagoi kehnosti, sillä yksi muutos yhdessä järjestelmässä voi pakottaa siihen, että kaikissa järjestelmissä joudutaan yhteyksiä muokkaamaan.



Kuva 2. Esimerkki point-to-point-tyyppisestä järjestelmäarkkitehtuurista. Jokaiselle sovellukselle on määriteltävä yksitellen yhteys toiseen sovellukseen, jotta dataa voidaan jakaa.

ESB:llä voidaan välttää point-to-point-tyyppiset integraatiot. Tällä tavoin kaikki olennaiset asetukset määritetään ESB:lle ja muutokset tehdään ainoastaan sinne. ESB hoitaa sanomamuunnokset, reititykset ja muut olennaiset asiat järjestelmien välisissä yhteyksissä. Tällöin sovellusarkkitehtuuri muistuttaa kuvaa 3, jossa sanomavälitys kulkee keskitetyn yhteysväylän läpi.



Kuva 3. Esimerkki järjestelmäarkkitehtuurista, jossa ESB-laite tai -sovellus hoitaa sovelluksien välisen yhteydenpidon.

Yhtenä suurena etuna ESB:ssä on se, että se voi toimia adapterina erilaisten kommunikointikeinojen välillä. Esimerkiksi kuluttajasovellus voi vaatia SOAP-muotoista dataa HTTP:n kautta, mutta taustajärjestelmä vaatii JSON-muotoista dataa JMS:n yli. ESB kykenee hoitamaan näiden datamuotojen ja välityskeinojen väliset erot ja siten varmistamaan, että kuluttaja ja taustajärjestelmä voivat kommunikoida. [Davis 2009: 39-40, 256-261.]

Sanomat itsessään voivat olla jokin datarakenne, kuten merkkijono tai taulukko. Sanomat kuitenkin koostuvat yleensä kahdesta osasta: Header- ja Body-osioista. Header-osio sisältää sanomaan liittyvää metadataa, kuten mistä sanoma on lähtenyt ja minne se menee. Useimmiten sovellukset eivät välitä Header-osion sisällöstä. Sen sijaan Body-osioon sisältyy sanoman varsinainen data, jota sovellukset hyödyntävät. [Enterprise Integration Patterns Introduction.]

Eri sovellukset käyttävät erilaisia sanomamuotoja. Haaste on siinä, miten nämä eri tietoformaatit voidaan sovittaa yhteen, jotta sovelluksien välinen sanomavälitys onnistuisi.

Integraatioissa sanomavälityksen aikana tehdään sanomille tarvittavat muunnokset, jotta sanoma olisi kohdesovelluksensa ymmärtämässä muodossa. Tällainen sanomamuunnos voisi olla esimerkiksi XML-muotoisen sanoman muuntaminen COBOL-muotoiseksi sanomaksi.

Sanomavälitysmenetelmiä on erilaisia. Yksi yleisimmin käytetyistä sanomavälitysmenetelmistä on request-response-menettely. Kuluttajasovellus lähettää pyynnön (request) toiselle sovellukselle tai järjestelmälle. Pyynnön aiheuttama toiminto taustajärjestelmässä voi olla esimerkiksi tietojen tai dokumenttien palauttaminen kuluttajasovellukselle tai jonkin toiminnallisuuden toteuttaminen. Pyyntöön odotetaan usein vastausta (response). Taustajärjestelmä tai -sovellus käsittelee pyyntöön liittyvät toiminnallisuudet ja palauttaa vastauksen kuluttajasovellukselle. [Request/Response.] Hyvin tyypillinen esimerkki tällaisesta request-response-menettelystä on siirtyminen verkkosivun linkkiä klikaten toiselle sivulle nähdäkseen sen tiedon, minkä haluaa nähdä.

Tässä insinöörityössä keskitytään sanomavälityksen osalta eri datamuotojen yhteensovittamiseen. Tämä tarkoittaa sitä, että data muutetaan sovelluksien ja järjestelmien hyväksymään muotoon jollakin muunnoksella, kuten XSLT-muunnostiedostolla. Sanomavälitysmenetelmistä tässä työssä keskitytään request-response-menettelyllä toteutuviin integraatioihin.

3 IBM WebSphere DataPower

Integraatiotarpeisiin on kyetty vastaamaan erilaisilla laitteilla ja sovelluksilla, jotka ovat hoitaneet järjestelmien ja sovelluksien välillä tapahtuvan kommunikoinnin. Yksi tällainen laite on IBM WebSphere DataPower. Se on IBM:n jatkokehittämä laitesarja, joka toimii muun muassa verkossa sanomavälityksessä. Tässä työssä keskitytään pääsääntöisesti XI-sarjan laitteeseen DataPower XI52, joka keskittyy ensisijaisesti järjestelmäintegraatioihin.

DataPower on palomuurilaite, joka on erikoistunut XML-pohjaisten sovellusten suojaamiseen. Laitteeseen on lisäksi rakennettu tehokas XSL-prosessori, ja se hyödyntää myös monia tietoturvaominaisuuksia. Laite sisältää myös valmiiksi XSLT-muunnoksia,

joita voidaan käyttää esimerkiksi vahingollisten sanomien karsimiseen. Tästä voi mainita esimerkkinä muunnostiedoston, jota voidaan käyttää tarkistamaan sanoma SQL-injektioiden varalta.

Laite vastaa myös luvussa 2.2 esitettyihin integraatiohaasteisiin [J. Rodriguez, ym. 2008: 3-7):

- Käytettävyys: Laite hoitaa sanomavälityksen eri järjestelmien välillä, jolloin itse järjestelmiin ei tarvitse ohjelmoida sanomavälitystä ja siihen liittyviä muunnoksia.
- Turvallisuus: Laite tarjoaa monenlaisia validointikeinoja ja palomuureja, joilla voi rajoittaa sanomavälityksen dataa hallitusti ja tarkistaa sanomia mahdollisten hyökkäysten varalta. Se tarjoaa myös laajat autentikointihallinnat, SSL-suojaukset sekä kryptaustoimenpiteet.
- Suorituskyky: Laite on rakennettu nimenomaan sanomavälitystä varten ja se keskittyy yksinomaan siihen. Näin kuluttajapuolten sovellusten työkuormaa on kevennetty huomattavasti eikä palvelimien tarvitse kuluttaa resursseja sanomakäsittelyyn.

Koko laitteen toiminta painottuu voimakkaasti XSLT:n käsittelyyn. DataPower-laite tarjoaa myös erilaisia muuttujia ja funktioita XSLT-kielen käyttöön. Nämä on kuvattu tarkemmin luvussa 4.2.2.

4 XSLT

4.1 XSLT-muunnoskieli

XSLT-muunnoskieli on osa XSL-kieltä. XSL on tarkoitettu kuvaamaan XML-dokumenttien muoto ja esitystapa. XSL koostuu kolmesta eri osasta, jotka ovat

- XSLT, jota käytetään muuntamaan XML-dokumenttien rakennetta. Tällä voidaan muuntaa XML-dokumentti rakenteeltaan toiseksi XML-dokumentiksi tai mahdollisesti jopa toiseen muotoon, kuten SOAP- tai tekstimuotoon.

- XSL-FO, jota käytetään määrittämään, miten XML-dokumentteja näytetään. Tällä voidaan määrittää, näytetäänkö XML-dokumentti esimerkiksi HTML tai PDF-muodossa.

- XPath, jota käytetään XML-dokumentin puurakenteen läpi navigoimiseen ja eri rakenteellisten osien, kuten elementtien tai attribuuttien, valitsemiseen. Tätä käytetään hyvin usein XSLT:n kanssa, jotta voidaan valita halutut XML-dokumentin osat käsiteltäväksi.

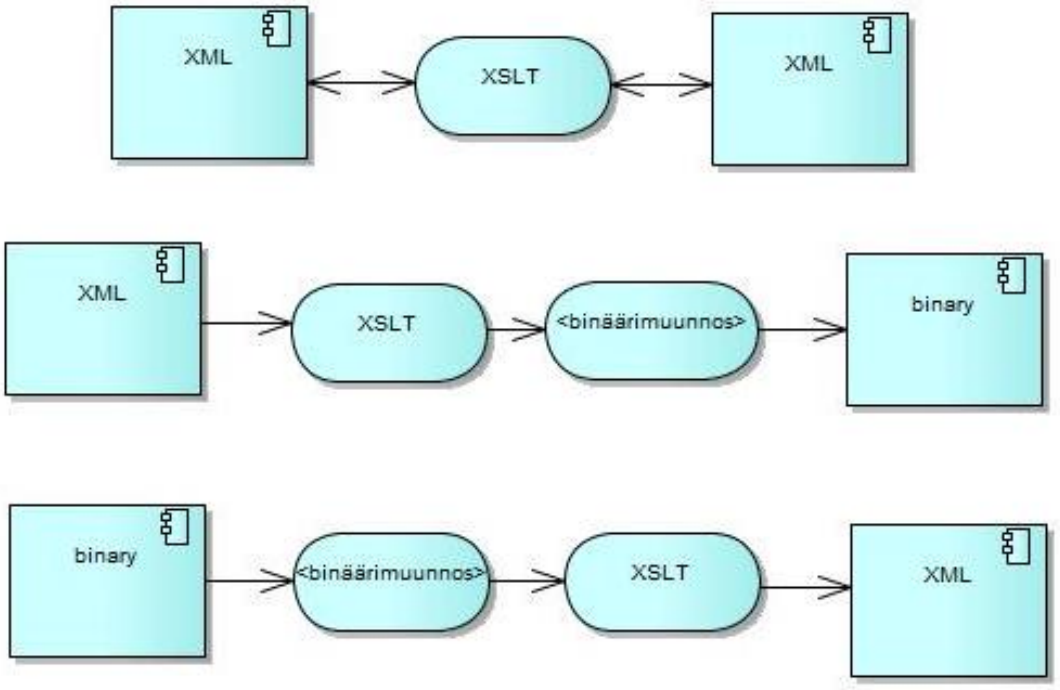
XSLT on World Wide Web Consortiumin (W3C) kehittämä muunnoskieli, joka käyttää XML-kuvauskielen syntaksia. W3-spesifikaationsa mukaan XSLT on kieli, jolla muunnetaan XML-dokumentteja toiseen muotoon. Sillä voidaan kuitenkin muuntaa XML-dokumentteja moniin eri muotoihin, kuten HTML- tai tekstimuotoon. XSLT on konkreettisella tasolla kieli, jolla muunnetaan XML-dokumenttien rakennetta. Se kuvaa tarvittavan muunnoksen ja antaa sitten XSL-prosessorin päättää, miten itse muunnos tehdään. [Kay 2000: 9-10, 13, 33.]

XSL-prosessori on se, joka toteuttaa XSLT:n kuvaaman muunnoksen XML-dokumenttiin ja tuottaa tulosdokumentin [Kay 2000: 15]. XSL-prosessoreja on monenlaisia, kuten Saxon tai Microsoft MSXML3. DataPower-laitteesta sarja XA on tietynlainen XSL-prosessori, koska sen päätarkoitus on tehostaa XSLT-muunnoksia. Laitteen XI-sarja sisältää myös XA-sarjan toiminteet, joten silläkin on sisäänrakennettu XSL-prosessori ja siten sen XSLT-muunnosten käsittely on tehty tehokkaammaksi [Ebbers ym. 2008: 8].

XSLT-kielestä on julkaistu versio 2.0, joka on laajentanut muunnoskielen ominaisuuksia. Vaikka vuonna 2007 versiosta 2.0 on tullut suositus [XSL Transformations (XSLT) Version 2.0], versio 1.0 on vielä laajalti käytössä. Tämä johtuu siitä, että monet selaimet ja järjestelmät, ja niiden XSL-prosessorit, eivät tue 2.0 versiota ja siksi käytetään versiota 1.0. Käytettävä XSLT-muunnoskielen versio riippuu pitkälti sovellusten ja järjestelmien teknisistä tiedoista. DataPower-laite sisältää koko tuen versioon 1.0, mutta vain joiltakin osin versioon 2.0 [Ebbers ym. 2008: 5-6]. Koska DataPower-laitteen tuki versiolle 2.0 on rajallinen tällä hetkellä, tämän insinöörityön käsittelemä XSLT-kieli on käsitelty versiossa 1.0.

Vaikka XSLT:tä käytetään muuntamaan XML-dokumentteja muodosta toiseen, sitä käytetään myös tapauksissa, joissa tieto muokataan ihmisille ymmärrettävään ja mielekkäämpään muotoon. XSLT pystyy myös muuntamaan muitakin kuin XML-dokumentteja [Kay 2000: 39-41]. XSLT-muunnoskieltä voi käyttää esimerkiksi muun-

tamaan XML-dokumentin tekstidokumentiksi ja toisin päin, kuten kuvassa 4 on näytetty. XSLT:llä pystyy käsittelemään monenlaista dataa, ja koska järjestelmien määrä kasvaa ja sitä myötä niiden käyttämät erilaiset tietomuodot, XSLT on käytetty järjestelmien välisissä integraatioissa, jotta järjestelmät saadaan kommunikoimaan keskenään.



Kuva 4. Tapauksia, miten XSLT-muunnoskieltä voidaan hyödyntää sanomamuunnoksissa. Muut kuin XML-muotoiset sanomat vaativat yleensä avukseen muita muunnoksia, mutta XSLT voi kuitenkin olla mukana muunnosprosessissa.

4.2 Laajennukset

XSLT-muunnoskielen käytössä voi tulla vastaan tilanteita, joissa XSLT:n olemassa olevat peruselementit ja -funktiot eivät riitä. Silloin XSLT-muunnoskieltä voi laajentaa omilla elementeillä ja funktioilla. [Tidwell 2008: 277.] Sen lisäksi XSLT:n tueksi on luotu erilaisia laajennoksia ja lisäosia, jotka tukevat XSLT-muunnoskieltä. Tässä luvussa esitetään DataPower-laitteen osalta olennaiset laajennukset, jotka ovat EXSLT ja DataPower-laitteen laajennuselementit ja -funktiot.

4.2.1 EXSLT

EXSLT laajennuselementit ja -funktiot ovat EXSLT-yhteisön kokoama joukko erilaisia moduuleja, joilla voidaan laajentaa XSLT-muunnoskielen käyttöä [EXSLT]. Eri moduulit keskittyvät eri käsittelyihin, kuten päivämääriin (dates and times -moduuli) ja merkkijonoihin (strings-moduuli). Nämä laajennukset eivät kuitenkaan ole käytettävissä niissä XSL-prosessoreissa, jotka eivät tue EXSLT:tä. Vaikka XSL-prosessori tukisikin EXSLT:tä, se ei välttämättä tue kaikkia EXSLT:n elementtejä ja funktioita [Tidwell 2008: 331]. DataPower-laitteen XSL-prosessori tukee joiltakin osin EXSLT-laajennuksia, mutta kaikkia niitä ei ole tuettu [Extension functions and elements]. Siksi on parasta tarkistaa DataPower-laitteen IBM:n infokeskuksesta, mitkä EXSLT laajennuselementit ja -funktiot on tuettu laitteessa.

Jotta EXSLT laajennuselementtejä ja -funktioita voidaan käyttää, pitää määrittää muunnostiedostolle niiden nimiavaruusmääritykset. Koodiesimerkki 1 näyttää, miten EXSLT laajennuselementit otetaan käyttöön XSLT-muunnostiedostossa.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
           xmlns:str="http://exslt.org/strings"
           xmlns:date="http://exslt.org/dates-and-times"
           extension-element-prefixes="date"
           exclude-result-prefixes="date">

    <xsl:template match="/">
        <pvm><xsl:value-of select = ”date:date(’2011-12-13’)” /></pvm>
    </xsl:template>
</xsl:stylesheet>
```

Koodiesimerkki 1. EXSLT laajennuselementtien ja -funktioiden käyttöönotto. Esimerkissä käytetään EXSLT moduulia ”Date and Times”. Käytetty funktio date() on tuettu DataPower-laitteessa.

4.2.2 DataPower-laajennuselementit ja -funktiot

Tässä luvussa esitetään yleisellä tasolla DataPower-laitteen tarjoamat laajennukset muutaman esimerkin kautta. Laajennuksista käydään läpi vain muutama olennainen ja hyödyllinen kappale, koska niitä on lukuisia ja niiden käyttö riippuu integraation ja tarpeen luonteesta.

DataPower-laite tarjoaa myös erilaisia laajennuselementtejä ja -funktioita XSLT-muunnoskielen käyttöön. Nämä laajennokset ovat käytettävissä ainoastaan DataPower-laitteissa, joten niitä ei voi käyttää muunnoksissa, jotka eivät toteudu DataPower-laitteella. Pääsääntöisesti nämä laajennukset mahdollistavat esimerkiksi erilaisten DataPower-muuttujien luomisen ja hyödyntämisen sanomakäsittelyn aikana. Nämä laajennukset on luotu ottamaan kaikki hyöty irti DataPowerin XSL-prosessorista [Extension functions and elements].

Jotta DataPower-laajennuselementtejä ja -funktioita voidaan käyttää, pitää määrittää muunnostiedostolle niiden nimiavaruusmääritykset. Koodiesimerkki 2 näyttää, miten DataPower-laajennuselementit otetaan käyttöön XSLT-muunnostiedostossa.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
           xmlns:dp="http://www.datapower.com/extensions"
           extension-element-prefixes="dp"
           exclude-result-prefixes="dp">

    <xsl:template match="/">
    <!--
    Muuttujan asetus siten, etta muuttuja haetaan XML-sanoman
    rakenteesta, joka on muotoa:
    <pyynto>
        <arvo>Hello</arvo>
    </pyynto>
    Muuttuja saisi arvokseen ”Hello”.
     -->
    <dp:set-variable
      name="'var://context/myVariables/select_var'"
      select="//pyynto/arvo"/>
    <!--
    Muuttujan asetus siten, etta muuttujalle annetaan merkkijonoarvo.
    -->
    <dp:set-variable
      name="'var://context/myVariables/value_var'"
      value=" world!"/>
    </xsl:template>
</xsl:stylesheet>
```

Koodiesimerkki 2. DataPower-laajennuselementtien käyttöönotto ja set-variable-laajennuselementin käyttö yksinkertaisessa tapauksessa. Attribuutit extension-element-prefixes ja excluide-result-prefixes ovat mukana, koska esimer-

kissä käytetään Datapower-laajennuselementtejä eikä niiden nimiavaruutta haluta näyttää sanomamuunnoksen lopputuloksessa.

DataPower-laajennusten käyttöönotossa voidaan käyttää kahta attribuuttia, jotka ovat vapaaehtoisia. Attribuutilla ”extension-element-prefixes” ilmaistaan ne nimiavaruudet, jotka liittyvät laajennuselementteihin. Attribuutilla ”exclude-result-prefixes” voidaan kertoa, mitä nimiavaruuksia ei pidä ottaa mukaan muunnoksen tuottamaan lopputulokseen. [J. Rodriguez ym. 2008: 124.] Näistä attribuuteista extension-element-prefixes on kuitenkin pakko olla mukana, jos muunnostiedostossa käytetään nimenomaan DataPower laajennuselementtejä [Extension functions and elements]. Laajennusfunktioiden suhteen molemmat attribuutit ovat vapaaehtoisia, koska silloin ei tarvitse implementoida laajennuselementtejä [J. Rodriguez ym. 2008: 124]. Jos muunnostiedosto käyttää pelkästään laajennusfunktioita, määritetään pelkästään DataPower-laajennusten nimiavaruus. Tämä attribuuttimenettely pätee myös EXSLT-laajennuksilla.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:dp="http://www.datapower.com/extensions"
    extension-element-prefixes="dp"
    exclude-result-prefixes="dp"
>
```

laajennuselementtien käyttöönotto

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:dp="http://www.datapower.com/extensions"
>
```

laajennusfunktioiden käyttöönotto

Kuva 5. Laajennuselementtien ja -funktioiden käyttöönotto koodissa. Vasemmalla ovat laajennuselementtien käyttöönottoon vaaditut nimiavaruudet, oikealla laajennusfunktioille.

DataPower laajennuselementeillä ja -funktioilla voidaan muokata DataPower-muuttujia. Laitteella on käytössä neljää eri tyypistä muuttajaa, jotka ovat:

- kontekstimuuttujat (context variables)
- palvelumuuttujat (service variables)
- lokaalit muuttujat (local variables)
- järjestelmämuuttujat (global variables).

Järjestelmämuuttujat ovat näkyvissä ja käytettävissä DataPower-laitteen kaikissa palveluissa. Palvelumuuttujat ovat olemassa transaktion aikana, ja ne ovat valmiiksi määriteltyjä. Ne sisältävät tietoa palveluun liittyvistä asetuksista ja ominaisuuksista, kuten taustajärjestelmän osoitteen. Kontekstimuuttujat ovat myös olemassa transaktion ajan, mutta niitä ei oletusarvoisesti ole olemassa. Kontekstimuuttujat luodaan muunnostie-

dostoissa DataPower-laajennuselementillä set-variable. Lokaalit muuttujat toimivat kuten kontekstimuuttujat, mutta niiden elinkaari on vain yhden kontekstin pituinen. Heti kun siirrytään toiseen kontekstiin sanomaprosessoinnin aikana, nämä muuttujat poistuvat käytöstä. Kontekstien toiminta on kuvattu tarkemmin luvussa 5.3.

DataPower-laajennuselementeillä voidaan esimerkiksi asettaa omia muuttujia ja niille arvoja. Koodiesimerkissä 1 määritettiin DataPower-laitteelle kaksi omaa muuttujaa: context/myVariables/select_var ja context/myVariables/value_var. Näitä kutsutaan kontekstimuuttujiksi ja niihin tallennetaan sovelluskohtaisia tietoja. Merkkijono ”var://” ilmaisee, että kyseessä on DataPower-laitteella oleva muuttuja.

DataPower-laajennuselementeillä voidaan käsitellä myös DataPower-laitteen olemassa olevia muuttujia. Tällaisia ovat esimerkiksi palvelumuuttujat, joiden muuttujaosoite alkaa merkkijonolla ”var://service”. Näissä tiedoissa on DataPower-palveluiden sanomaprosesseihin liittyviä tietoja, kuten kuluttajasovelluksen ja taustajärjestelmän osoite. DataPower-palvelut on kuvattu tarkemmin luvussa 5.2 DataPower-palveluobjektit. Kuva 6 näyttää esimerkin mukaisesti, miten palvelumuuttujia voidaan hyödyntää esimerkiksi asettamaan taustajärjestelmän osoite ajonaikaisesti pyynnön käsittelyssä. Tässä on käytetty samaa laajennuselementtiä, dp:set-variable, kuten Koodiesimerkissä 1, mutta muuttujaa ei luoda vaan olemassa olevan muuttujan arvoa muokataan.

Kuva 6. Laajennuselementin dp:set-variable käyttö DataPower-palvelumuuttujaan. Vasemmalla on määritetty, että DataPower-palvelun taustajärjestelmä asetetaan ajonaikana muunnostiedostossa. Oikealla nähdään muunnostiedoston koodi, joka asettaa taustajärjestelmän osoitteen, kun muunnostiedosto ajetaan.

DataPower-laajennusfunktioiden käyttöönotto ja käyttäminen toimivat samalla tavalla kuin laajennuselementit. Koodiesimerkissä 3 on esitetty, miten laajennusfunktiolla va-

riable() voidaan hakea DataPower-muuttujista tietoa. Oletuksena on, että koodiesimerkin 2 mukaiset muuttujat on asetettu.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl=http://www.w3.org/1999/XSL/Transform
     xmlns:dp="http://www.datapower.com/extensions">

     <xsl:template match="/">
         <vastaus>
            <select_arvo>
                <xsl:value-of select=
                "dp:variable('var://context/myVariables/select_var')"/>
            </select_arvo>
            <value_arvo>
                <xsl:value-of select=
                "dp:variable('var://context/myVariables/value_var')"/>
            </value_arvo>
         </vastaus>
     </xsl:template>
</xsl:stylesheet>
```

Koodiesimerkki 3. DataPower-laajennusfunktioiden käyttöönotto ja variable()-laajennusfunktion käyttö yksinkertaisessa tapauksessa perustuen koodiesimerkkiin 1.

DataPower-laajennuselementtejä ja -funktioita on monen tyyppisiä ja eri käyttötarkoituksiin tarkoitettu. Tarkka lista näistä laajennuselementeistä ja -funktioista toimintoineen on lähteessä [Extension functions and elements].

5 Sanomamuunnokset DataPower-integraatiossa

Tässä luvussa on tarkoitus kuvata, miten käytännössä DataPower-integraatioita toteutetaan ja miten niihin liitetään sanomamuunnokset. Luvussa käydään läpi DataPower-laitteen käsittelemät sanomatyypit ja palveluobjektit, joilla integraatioita voidaan toteuttaa. Pääpainotus on siinä, miten XSLT-muunnoskieltä on voitu hyödyntää integraatioissa. Luvun lopussa käydään läpi esimerkki-integraatioiden toteutusta ja sitä, miten niiden tekemiä sanomamuunnoksia voi tarkastella.

Pääsääntöisesti kehitystyö DataPower-laitteella on erilaisten DataPower-palvelujen konfigurointia. Ohjelmointia tarvitaan lähinnä XSLT-muunnostiedostojen toteuttamisessa, jotta sanomamuunnoksia voidaan tehdä. DataPower-integraatioissa toteutuu usein request-response-menettely, sillä integraatioissa konfiguroidaan käsittely sekä pyyntö- että vastaussanomille. Riippuen tilanteesta sanomavälityksen käsittely voidaan konfiguroida myös kaksisuuntaiseksi, jolloin sekä pyyntöön että vastaukseen pätevät samat toimintatavat. Myös yksisuuntainen prosessointi eli pelkän pyynnön käsittely on mahdollista.

5.1 Sanomatyypit

DataPower-laitteella pystyy toteuttamaan monenlaisia sanomamuunnoksia. Se tukee seuraavia sanomatyyppejä:

- SOAP
- XML
- JSON
- Pass-thru
- Non-XML.

Pass-thru-tyyppiset sanomat voivat olla mitä tahansa mikä tarkoittaa, että sanoma välitetään sellaisenaan DataPower-laitteen läpi sanomien prosessointitoiminnoista huolimatta. Tätä sanomatyyppiä voidaan käyttää esimerkiksi testaamaan, että taustajärjestelmä vastaa ja sanomavälitys DataPower-laitteen läpi toimii. Non-XML-sanomatyyppi valitaan silloin, kun sanoma on muuta muotoa kuin XML:ää, SOAP:ia tai JSON:ia. Tällöin sanomat voivat olla mitä tahansa muotoa, kuten esimerkiksi COBOL-koodia tai tavallista tekstiä.

XML- ja SOAP-sanomat ovat usein käytössä sanomavälityksessä, ja DataPower kykenee käsittelemään niitä. XML on pitkään käytetty sanomamuoto, ja monet järjestelmät tukeutuvat sen käyttöön. SOAP-sanomat ovat XML-muotoisia sanomia, joiden sisältö on käärityt SOAP-kuorien sisälle. DataPower kykenee käsittelemään myös JSON-sanomia, jotka ovat yleisempiä REST-pohjaisissa palveluissa.

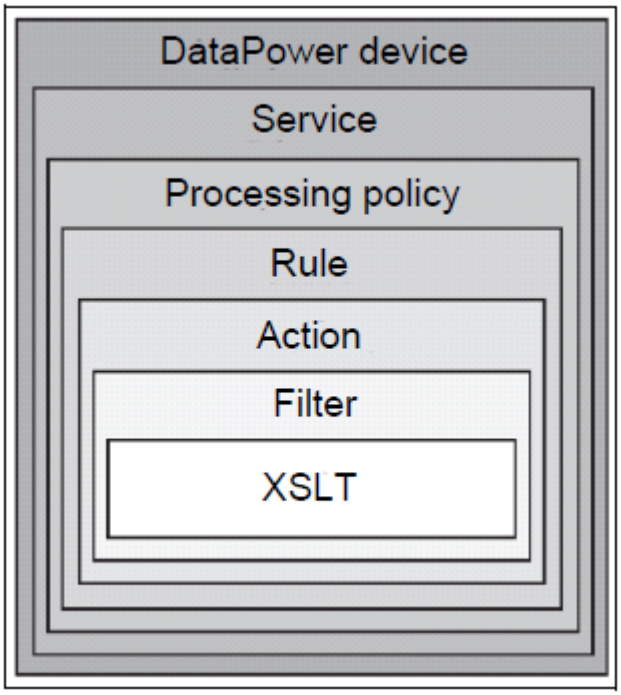
```
<!-- SOAP MESSAGE -->
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:prod="http://www.ibm.com/datapower/ProductService/">
    <soap:Header/>
    <soap:Body>
        <prod:ProductServiceOutput>
            <datapower-type>XS40</datapower-type>
            <software-brand>Security</software-brand>
            <encoded-description>Encoded description</encoded-description>
            <benefits level="A">Great</benefits>
        </prod:ProductServiceOutput>
    </soap:Body>
</soap:Envelope>


<!-- XML MESSAGE -->
<prod:ProductServiceOutput>
    <datapower-type>XS40</datapower-type>
    <software-brand>Security</software-brand>
    <encoded-description>Encoded description</encoded-description>
    <benefits level="A">Great</benefits>
</prod:ProductServiceOutput>
```

Kuva 7. Esimerkki SOAP-sanomasta (yllä) ja XML-sanomasta (alla) samalla tietosisällöllä. SOAP on yhtä lailla XML:ää, mutta sen ympärillä on SOAP-kuoret. Käytännössä SOAP Envelopen sisällä on SOAP Header ja SOAP Body ja Bodyn sisällä itse sanoma.

Jotta sanomamuunnoksia voidaan tehdä, pitää luoda XSLT-muunnostiedostot ja määrittää muunnostoiminnolle, mistä muunnostiedosto haetaan. Se voidaan hakea esimerkiksi DataPower-laitteen hakemistosta. Varsinaiset sanomamuunnokset toteutuvat DataPower-palveluobjekteissa, joihin on määritetty, miten pyyntö- ja vastaussanomia tullaan käsittelemään. XSLT-kieltä voidaan hyödyntää missä tahansa sanomatyypissä.

5.2 DataPower-palveluobjektit

DataPower-laitteelle tehtävä integraatiokonfiguraatio voidaan esittää yleisellä tasolla kuvan 8 mukaisesti. DataPower-laitteen sisälle määritetään ensin palvelu, jolle luodaan prosessointisäännöstö (processing policy). Sen jälkeen luodaan tarvittavat käsittelysäännöt (Rule), kuten pyyntö-, vastaus- ja virhesäännöt. Käsittelysääntöihin lisätään erilaisia toiminteita (Action), joilla käsitellään sanomaa. Muunnostoiminteille voidaan valita XSLT-muunnostiedosto, jolla muunnetaan sanomaa.



Kuva 8. Peruskonfiguraatioarkkitehtuuri DataPower-laitteessa [Ebbers ym.2008: 14].

DataPower-laitteella on viisi erilaista palveluobjektia, joilla voidaan toteuttaa integraatioita. DataPowerilla palvelu tarkoittaa tietyntyyppistä palveluobjektia, joka toteuttaa palveluobjektille määritetyt asetukset ja käsittelee asetuksien mukaisesti kuluttajasovelluksien kutsujen myötä tulevia sanomia. DataPower-palvelut tarjoavat käsittelyt sanomaprosesseille kahden tai useamman järjestelmän välillä. Palveluobjektit ovat:

- Web Service Proxy
- Multi-Protocol Gateway
- XML Firewall
- Web Application Firewall (WAF)
- XSL Processor.

XSL Processor -palvelu tehostaa XSLT-muunnosten prosessointia. Sitä käytetään vain harvoin XI-sarjan DataPower-laitteissa. Web Application Firewall -palvelua käytetään lähinnä käsittelemään HTTP-liikenteen pyyntöihin kuin XML-sanomien käsittelyyn. Sitä voidaan käyttää tilanteissa, joissa käytetään ainoastaan HTTP-liikennettä ja pelkästään muuta kuin XML-muotoista dataa [Ebbers ym. 2008: 23].

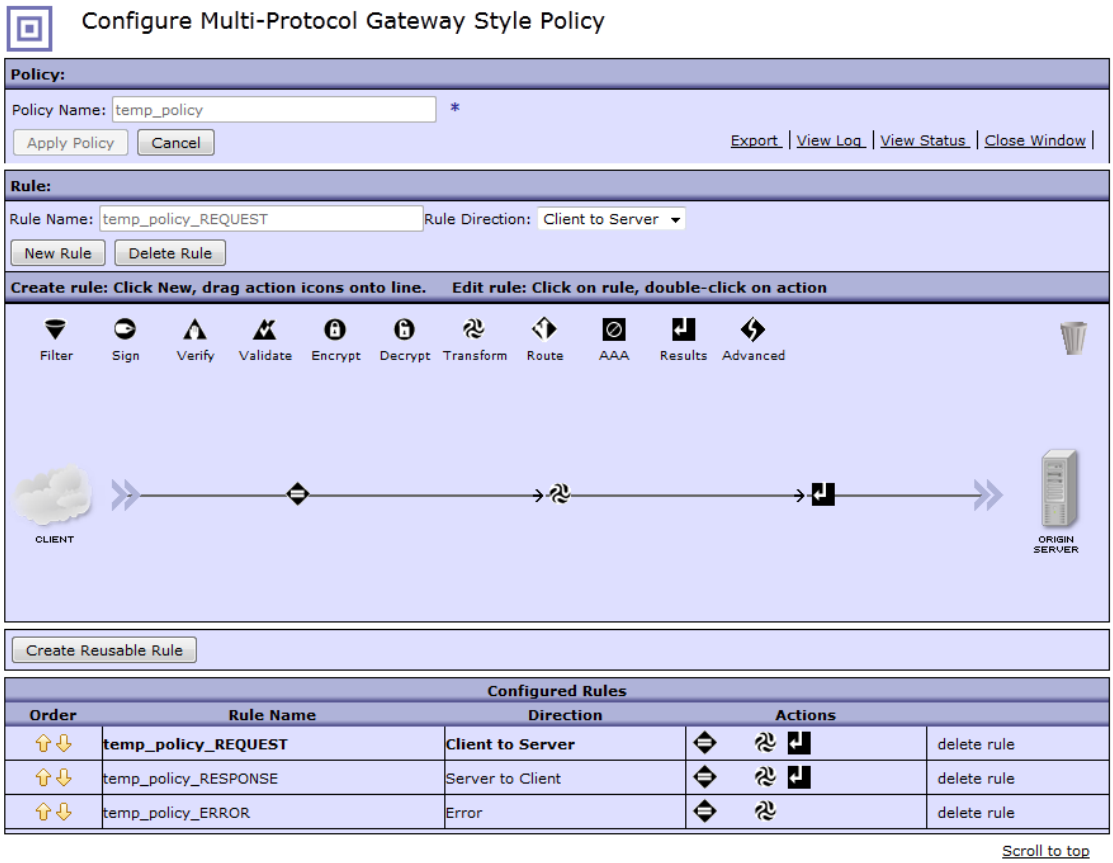
XML Firewall -palvelu tarjoaa XML-turvallisuuteen liittyviä operaatioita. Se on kätevä tapauksissa, joissa sanomavälitys toteutuu kokonaan XML-muotoisena. Se kykenee käsittelemään myös muitakin kuin XML-muotoisia sanomia, mutta näitä varten suositellaan käytettäväksi muuta kuin XML Firewall -palvelua [Ebbers ym. 2008: 19-20]. Tätä palvelua suositellaan käytettäväksi lähinnä testaamisessa.

Multi-Protocol Gateway -palvelu pystyy toteuttamaan sen, minkä XML Firewall pystyy. Tämän palvelun vahvuus on se, että se kykenee vastaanottamaan pyyntöjä ja vastauksia monesta eri protokollasta, kuten HTTP-, JMS-, MQ- tai FTP-protokollista. Tätä voidaan käyttää moneen eri tarkoitukseen, ja se on kätevimmillään tapauksissa, joiden käsittelysäännöt sisältävät monimutkaista logiikkaa.

Web Service Proxy -palvelu pystyy toteuttamaan palvelun, joka on kuvattu WSDL-tiedostossa. Koska palvelu muodostuu WSDL-tiedoston perusteella, Web Service Proxylla voidaan määrittää WSDL-tiedoston kuvaamiin operaatioihin yksitellen omat prosessointisäännöstöt käsittelysääntöineen. Lisäksi Web Service Proxy tekee automaattisen SOAP-sanomien skeemavalidoinnin WSDL-kuvauksen perusteella. Muissa palveluissa skeemavalidointi pitää lisätä erikseen käsittelysääntöihin. Web Service Proxylla on myös mahdollista toteuttaa yhdensuuntainen sanomaprosessointi eli pelkän pyyntösanoman käsittely.

Pääsääntöisesti integraatioissa ovat käytössä Web Service Proxy- ja Multi-Protocol Gateway -palvelut. Perusperiaatteena palvelutyypin valinnassa voi olla se, että jos käytössä on WSDL-tiedosto, luodaan Web Service Proxy. Muissa tapauksissa ja varsinkin, jos käytössä on muu kuin HTTP-protokolla, kannattaa käyttää Multi-Protocol Gateway -palvelua. Palveluobjektin tyypin valinta riippuu yleensä tehtävän integraation luonteesta.

Kuvassa 9 näkyy yleinen näkymä DataPower-palveluobjektin prosessointisäännöstöön (policy). Siitä nähdään myös joitakin käsittelysääntöjä (rule), joille voi määrittää suunnan ja toiminnot eli actionit. Jokaiselle palvelulle määritellään prosessointisäännöstö ja sen sisälle käsittelysäännöt. Säännöt koostuvat erilaisista toiminteista, actioneista, joilla voidaan käsitellä sanomaa. Actioneita on monenlaisia, mutta tässä työssä kuvataan ainoastaan Transform- ja Transform Binary -toiminnot tarkemmin, koska ne hyödyntävät XSLT-muunnoksia.



Kuva 9. Näkymä prosessointisäännöstöön DataPower-laitteen Multi-Protocol Gateway -palvelussa. Janassa näytetään lihavoidun säännön tekemät toiminnot, tässä kuvassa temp_policy_REQUEST-säännön toiminnot. Näkymä on sama DataPower-palveluobjektin tyypistä riippumatta.

5.3 Kontekstit

Sanomien prosessoinnissa käytetään kontekseja. Kontekstien käytöllä voidaan määrittää, mihin tuotetaan jokaisen toiminnon lopputulos tai mistä kontekstista otetaan toiminnolle inputdata. Jokaiselle toiminnolle on määriteltävä sisään menevä input- ja ulos tuleva outputkonteksti. Jokaisella DataPower-palvelulla on seuraavat kontekstit käytettävissä:

- INPUT
- NULL

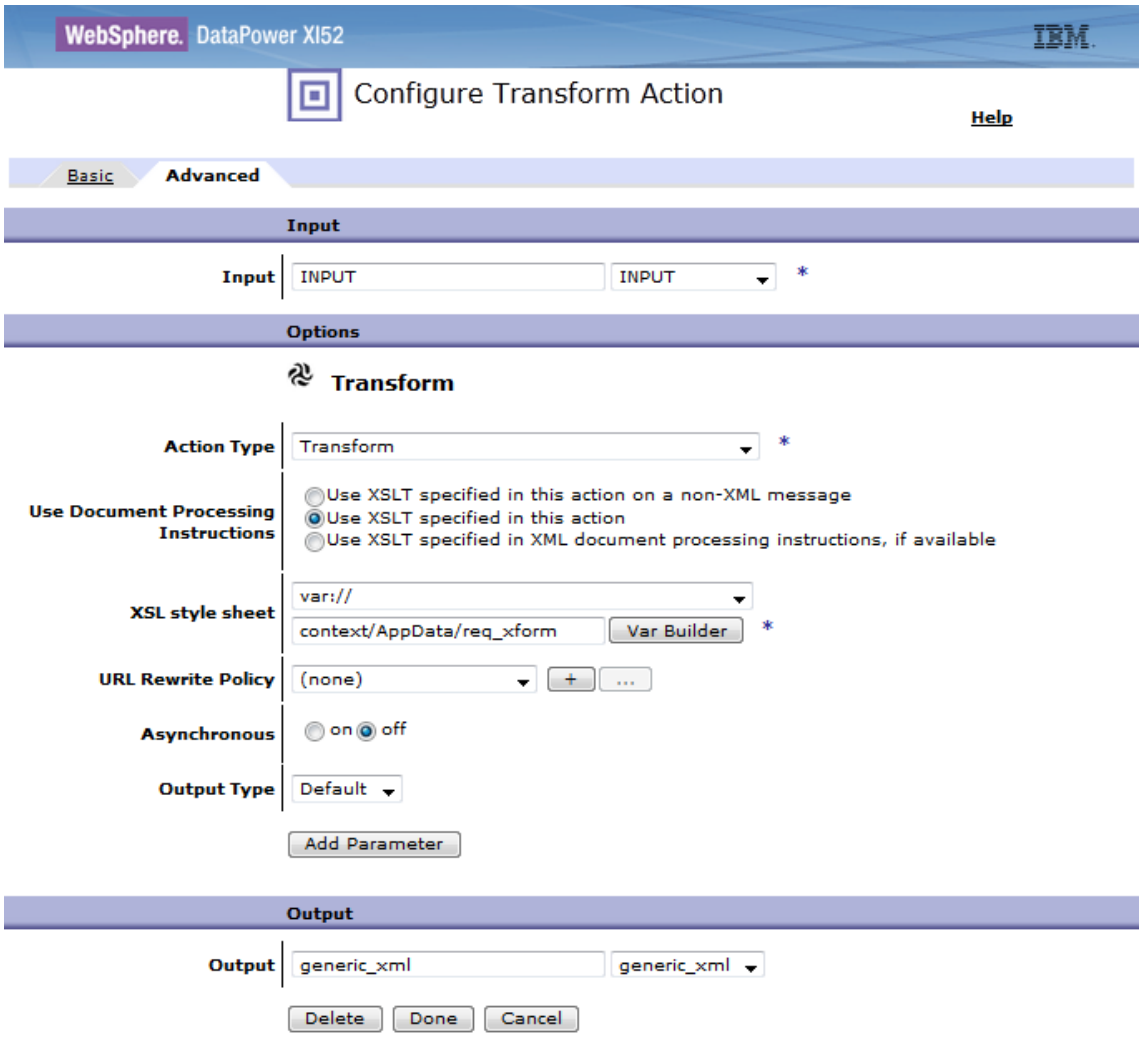
- PIPE
- OUTPUT.

Prosessointisäännön vastaanottama sanoma menee ensimmäisenä INPUT-kontekstiin. NULL-kontekstilla toiminto ei joko käytä mitään inputdataa tai sen outputdata hylätään. Esimerkiksi XSLT-muunnos, joka ainoastaan asettaa muuttujia, mutta jolla ei haluta muokata sanomaa, voi hyödyntää outputkontekstina NULL-kontekstia. PIPE-kontekstia voidaan käyttää välittämään kontekstisisältö seuraavalle toiminnolle. Tässä tapauksessa seuraavan toiminnon on pakko määrittää inputkontekstiksi PIPE-konteksti. OUTPUT-kontekstilla voidaan lähettää toiminnon outputdata suoraan prosessointisäännön loppusovellukselle, joka voi olla taustajärjestelmä tai pyynnön tehnyt kuluttajasovellus.

Näiden lisäksi voidaan luoda omia konteksteja vaihtoehdolla (auto). Omilla konteksteilla voidaan määrittää konteksteja, joita voidaan kutsua myöhemmin. Esimerkiksi oma konteksti voidaan luoda heti ensimmäisen toiminnon jälkeen ja käyttää uudestaan vasta useamman kuin yhden toiminnon jälkeen. Kuvissa 10 ja 11 on määritetty toiminnolle inputkontekstiksi INPUT ja outputkontekstiksi itse luotu konteksti ”generic_xml”.

5.4 Transform-toiminto

Kun halutaan käyttää mitä tahansa XSLT-muunnosta, silloin valitaan käytettäväksi Transform-toiminto. Sen tarkoitus on muuntaa XSLT-muunnoksella sanoman sisältöä ja rakennetta sen mukaan, mitä muunnostiedostossa on määritetty.

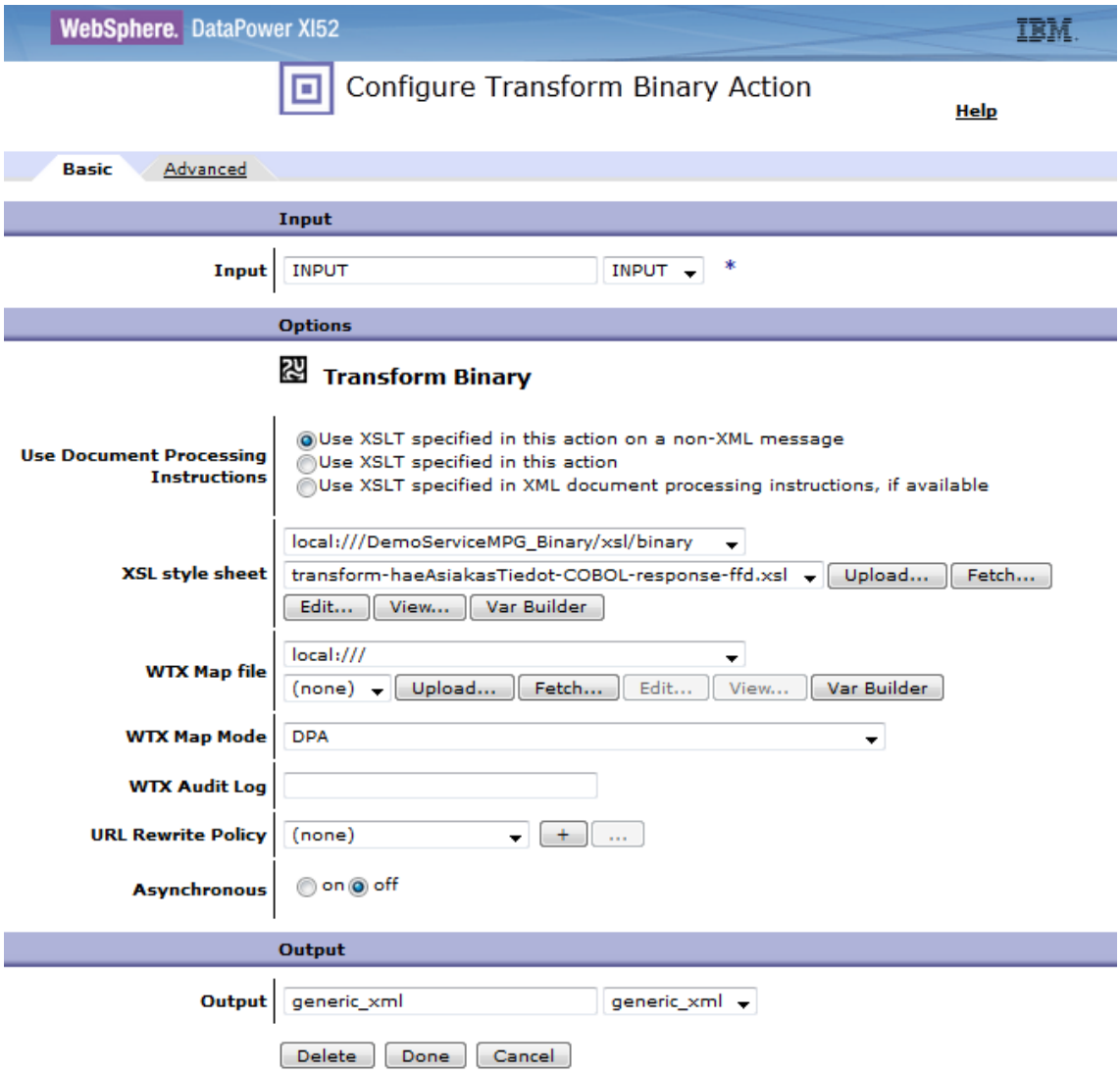


Kuva 10. Transform-toiminnon asetusikkuna. Täällä määritetään, mistä muunnostiedosto haetaan. Kuvan esimerkissä tiedosto haetaan DataPower-kontekstimuuttujasta ”context/AppData/req_xform”. Muunnoksen lopputulos tuotetaan ”generic_xml” kontekstiin.

Transform-toiminnolle voi valita muunnostiedoston eri tavoin, kuten esimerkiksi DataPower-laitteen paikallisesta hakemistosta, http-osoitteesta tai DataPower-muuttujan tiedoista. Toiminnolle voidaan määritellä myös erillisiä parametreja, jotka voidaan hakea muunnostiedoston käyttöön DataPower-laajennuselementillä dp:param. Toiminnolle määritetään, mihin sisään tulevaan dataan muunnos toteutetaan ja mihin tuotetaan lopputulos.

5.5 Transform Binary -toiminto

Transform Binary -toiminnossa määritetään, millä tavoin muunnos binäärimuotoon tehdään. Sille valitaan Transform-toiminnon tavoin muunnostiedosto. XSLT-kielellä toteutettua muunnostiedostoa ei välttämättä määritetä tähän toimintoon, mutta XSLT:llä voidaan muuntaa sanoma väliaikaisesti sellaiseen XML-muotoon, johon Transform Binary -toiminnon määritetty toisenlainen muunnostiedosto kykenee tekemään muunnokset. Transform Binary -toiminnolle voidaan määrittää samalla tavoin erillisiä parametreja ja muita asetuksia kuten Transform-toiminnossa.



Kuva 11. Transform Binary -toiminnon asetusikkuna. Toiminnolle voi määrittää, millä tavoin binäärimuunnos tehdään. Kuvassa binäärimuunnosta varten on määritetty XSLT-tiedosto.

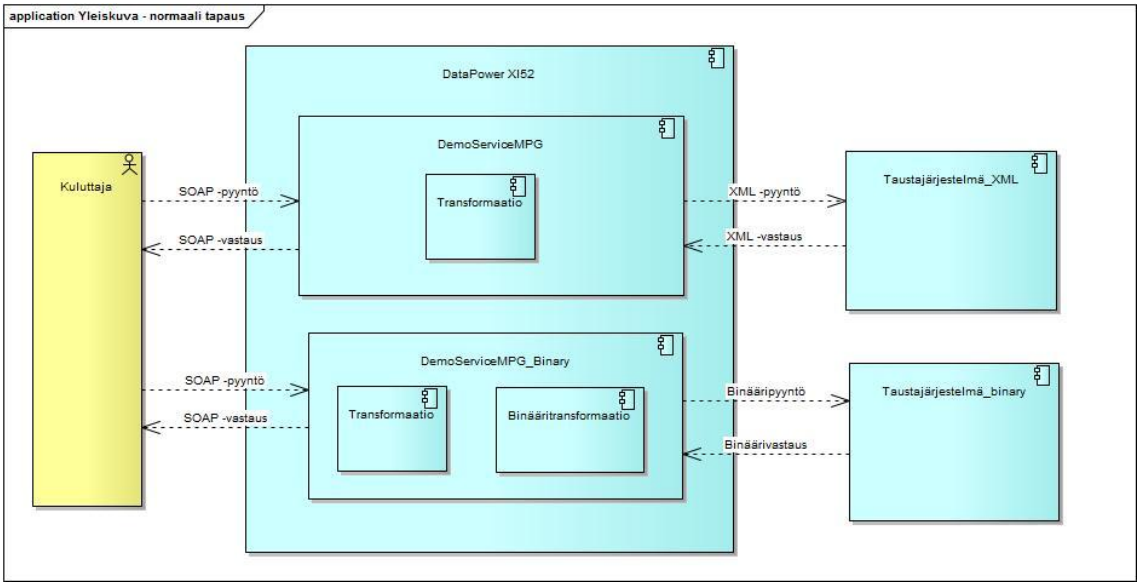
XSLT-muunnoksia ei voi usein käyttää suoraan muuhun kuin XML-muotoiseen dataan. Esimerkiksi COBOL-muotoisia sanomia XSLT ei pysty käsittelemään suoraan. Sen sijaan COBOL-muunnokselle luodaan erillinen muunnostiedosto, joka muuntaa XML-muodon COBOL-muotoon. XSLT:n rooli on muuntaa tuleva sanoma sellaiseen XML-muotoon, jonka COBOL-muunnokseen käytettävä muunnostiedosto kykenee lukemaan ja muuntamaan. Vastaavasti muunnos COBOL:sta johonkin XML-pohjaiseen muotoon toteutuu siten, että COBOL:in muunnostiedosto muuntaa sanoman XML-muotoon, jotta XSLT voi muuntaa sen vielä vastaanottajan vaatimaan muotoon.

5.6 Sanomamuunnokset käytännössä

Sanomamuunnoksien havainnollistamista varten on luotu kaksi DataPower-palvelua: DemoServiceMPG ja DemoServiceMPG_Binary. Näillä voidaan havainnollistaa, miten XSLT-kielellä toteutetut sanomamuunnokset kulkevat DataPower-laitteella. Molemmat palvelut on toteutettu Multi-Protocol Gateway -palveluina, koska käytössä ei ole WSDL-tiedostoa ja tässä tapauksessa kyseinen palvelutyyppi pystyy ajamaan saman asian kuin Web Service Proxy -palvelu.

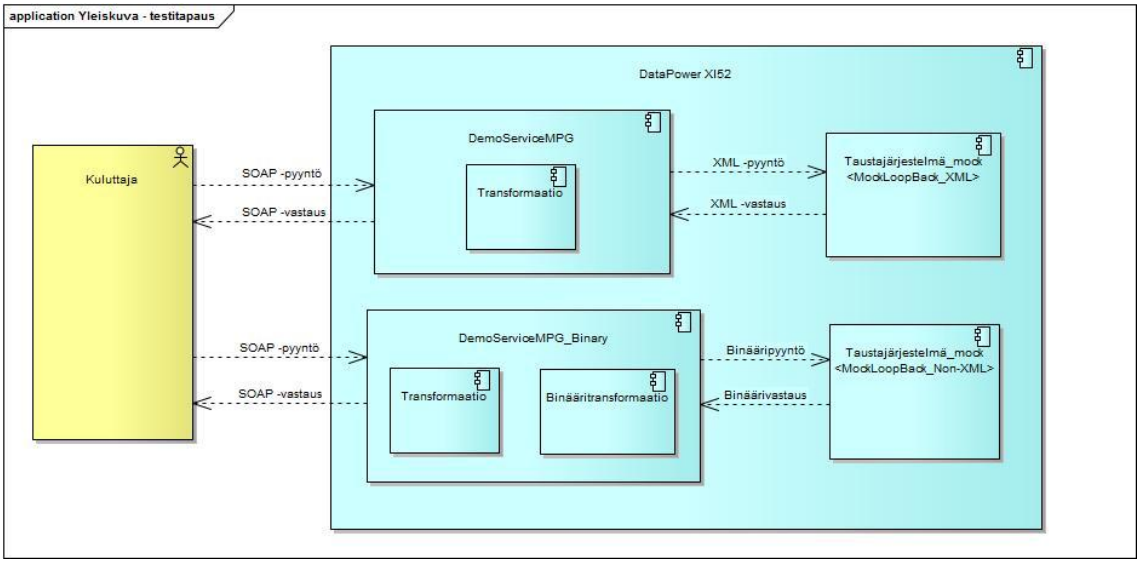
DemoServiceMPG-palvelu käsittelee XML-muotoisia vastauksia ja DemoServiceMPG_Binary binäärimuotoisia vastauksia. Jälkimmäisessä tapauksessa taustajärjestelmän binäärimuotoinen data on positiopohjaisessa muodossa eli tekstimuodossa. Molemmat palvelut vastaanottavat pyynnön SOAP-muodossa. DemoServiceMPG-palvelulla voidaan myös havainnollistaa, miten DataPower-laajennuselementtejä ja -funktioita voidaan käyttää sanomamuunnoksen aikana.

Tositilanteissa taustajärjestelmä on jokin DataPower-laitteen ulkopuolinen järjestelmä tai sovellus, jolloin DataPowerin läpi kulkeva integraatio voisi näyttää kuvan 12 mukaiselta. Riippuen tilanteesta integraatio voi kulkea DataPower-laitteen sisällä usean eri DataPower-palvelun läpi ja mahdollisesti siirtyä johonkin toiseen välikappaleeseen, esimerkiksi MQ-jonoon, ennen kuin sanoma menee taustajärjestelmälle.



Kuva 12. Yksi mahdollinen integraation kulku DataPower-laitteen läpi. Kuluttajasovellus tekee erilaisia pyyntöjä ja riippuen niiden osoitteesta pyynnöt tulevat tietylle DataPower-palvelulle. DataPower-palveluobjektissa toteutetaan käsittelysäännölle tehdyt toiminnot, kuten XSLT-muunnos, ja niiden jälkeen pyyntö välitetään taustajärjestelmälle. Taustajärjestelmä palauttaa vastauksen DataPowerin kautta kuluttajalle ja vastaukselle tehdään myös olennaiset toimenpiteet, kuten muunnos.

DemoServiceMPG- ja DemoServiceMPG_Binary-palveluissa taustajärjestelmänä käytetään samaa DataPower-laitetta, mutta eri palveluita kuvan 13 mukaisesti. Taustapalveluina toimivat XML Firewall -palvelut nimeltään MockLoopBack_XML ja MockLoopBack_Non-XML, jotka palauttavat tietynlaisen vastauksen perustuen siihen, millainen pyyntösanoman reititys-URL on. MockLoopBack_XML vastaanottaa ja palauttaa XML-muotoista dataa ja MockLoopBack_Non-XML binäärimuotoista dataa. Molempien mock-palveluiden vastaukset ovat DataPower-laitteella tiedostoina laitteen paikallisessa hakemistossa. Tällä tavoin voidaan testata vastauskäsittelyä, jos on tiedossa, millaisena oikea taustajärjestelmä palauttaa vastauksen.



Kuva 13. Integraation kulku mahdollisessa testitapauksessa. Pyyntö ohjautuu samalla tavoin DataPower-laitteen palveluobjektille kuin kuvassa 12. Testitapauksessa palveluobjekti kuitenkin välittää pyynnön DataPower-laitteen toiselle palvelulle, tässä tapauksessa mock-palvelulle, joka toimii taustajärjestelmänä ja palauttaa valmiiksi määritellyn vastauksen.

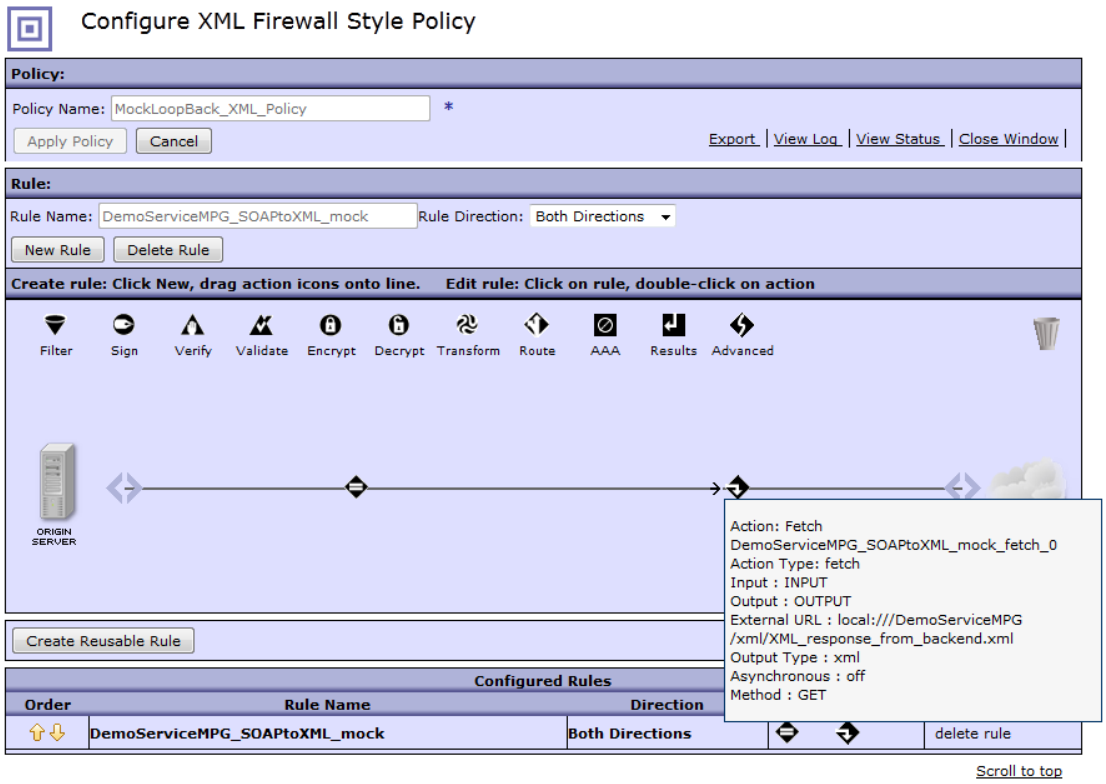
XSLT-sanomamuunnoksia havainnollistetaan molemmissa palveluissa eri tavoin. Luvuissa 5.6.1 ja 5.6.2 kuvataan tarkemmin palvelujen toteutus ja niiden tekemät sanomamuunnokset XSLT-kielellä.

5.6.1 DemoServiceMPG

Tämä DataPower-palvelu käsittelee SOAP-pyyntöjä ja vastaanottaa XML-vastauksia. Kuvan 13 mukaisesti taustajärjestelmänä käytetään mock-palvelua nimeltä MockLoopBack_XML, joka on XML Firewall -tyyppinen palveluobjekti. Palvelun on tarkoitus palauttaa XML-muotoisia vastauksia. Se kuuntelee laitteen sisältä eli osoitteesta 127.0.0.1 tulevia XML-pyyntöjä ja soveltaa käyttöön sitä käsittelysääntöä, johon pyynnön URL-osoite viittaa.

Esimerkiksi DemoServiceMPG-palvelu lähettää pyynnön osoitteeseen http:127.0.0.1:8095/SOAPtoXML. Mock-palvelu kuuntelee kyseiseen porttiin tulevia pyyntöjä ja päättelee URL-päätteen perusteella, mitä käsittelysääntöä käytetään sanoman käsittelyssä. DemoServiceMPG-palvelua varten mock-palvelulle on luotu käsittelysääntö, joka käsittelee pyynnöt, joiden URL-pääte on /SOAPtoXML. Mock-palvelu tekee vain yhden toimenpiteen. Se hakee Fetch-toiminnolla DataPower-laitteen paikal-

lisesta hakemistosta valmiiksi määritellyn tiedoston ja palauttaa sen sisällön vastauksena pyynnön tehneelle palvelulle.



Kuva 14. MockLoopBack_XML-palvelun prosessointisäännöstö. Palvelu hakee Fetch-toiminnolla valitun tiedoston ja palauttaa kutsuvalle palvelulle vastauksensa kyseisen tiedoston sisällön. Hiiri on viety Fetch-toiminnon päälle, jolloin nähdään siihen liittyvät tiedot ja asetukset.

Integraation kulku menee seuraavasti:

1. Kuluttajasovellus tekee SOAP-pyynnön DemoServiceMPG-palvelulle.

2. DemoServiceMPG-palvelu asettaa pyyntösanoman käsittelylle olennaiset muuttujat XSLT-muunnostiedostolla ”set-request-variables” ja ottaa pyyntösanoman SOAP-kuoresta talteen sen nimiavaruuden.

3. DemoServiceMPG-palvelu muuntaa SOAP-pyynnön taustajärjestelmän hyväksymään XML-muotoon.

4. Pyyntö välitetään taustajärjestelmälle.

5. Taustajärjestelmä palauttaa vastauksen XML-muodossa.

6. DemoServiceMPG-palvelu asettaa vastauskäsittelylle olennaiset muuttujat.

7. DemoServiceMPG-palvelu muuntaa XML-vastauksen kuluttajan hyväksymään SOAP-muotoon ja välittää vastauksen kuluttajalle.

5.6.2 DemoServiceMPG_Binary

Tämä DataPower-palvelu käsittelee SOAP-pyyntöjä ja vastaanottaa tekstimuotoisia binäärivastauksia. Kuvan 13 mukaisesti taustajärjestelmänä käytetään mock-palvelua nimeltä MockLoopBack_Non-XML. Palvelun on tarkoitus palauttaa binäärimuotoisia vastauksia. Se kuuntelee laitteen sisältä eli osoitteesta 127.0.0.1 tulevia binääripyyntöjä ja soveltaa käyttöön sitä käsittelysääntöä, johon pyynnön URL-osoite viittaa.

Esimerkiksi DemoServiceMPG_Binary-palvelu lähettää pyynnön osoitteeseen http:127.0.0.1:8095/SOAPtoBinary. Mock-palvelu kuuntelee kyseiseen porttiin tulevia pyyntöjä ja päättelee URL-päätteen perusteella, mitä käsittelysääntöä käytetään sanoman käsittelyssä. DemoServiceMPG_Binary-palvelua varten mock-palvelulle on luotu käsittelysääntö, joka käsittelee pyynnöt, joiden URL-pääte on /SOAPtoBinary. Mock-palvelu tekee vain yhden toimenpiteen. Se hakee DataPower-laitteen paikallisesta hakemistosta valmiiksi määritellyn tiedoston ja palauttaa sen sisällön vastauksena pyynnön tehneelle palvelulle. Toiminta on täysin samanlainen kuin XML-tiedoston palauttavassa mock-palvelussa, mutta tämä mock-palvelu palauttaa binäärimuotoisia vastauksia.

XSLT-muunnoksia ei voida tehdä suoraan binäärimuotoiselle datalle. Binääridataa voidaan kuitenkin käsitellä monilla muilla tavoilla, kuten WTX-kuvaustiedostoilla tai FFD-kuvauksilla (Flat File Descriptor). Sen sijaan XSLT:tä käytetään usein muuntamaan sanoma sellaiseen XML-muotoon, jota binäärimuunnos kykenee käsittelemään. DemoServiceMPG_Binary-palvelussa binäärimuunnosta varten on käytetty FFD-kuvausta, joka muistuttaa XML:ää ja se kertoo DataPowerille, millaisena sanoman pitää tulla käsittelyyn ja millaisena se tuotetaan ulos.

Integraation kulku menee seuraavasti:

1. Kuluttaja tekee SOAP-pyynnön DemoServiceMPG_Binary-palvelulle.

2. DemoServiceMPG_Binary-palvelu asettaa pyyntösanoman käsittelylle olennaiset muuttujat XSLT-muunnostiedostolla ”set-request-variables” ja ottaa pyyntösanoman SOAP-kuoresta talteen sen nimiavaruuden.

3. DemoServiceMPG_Binary-palvelu muuntaa SOAP-pyynnön välilliseen XML-muotoon.

4. Väli-XML muunnetaan FFD-määrityksen mukaisella XSLT-muunnoksella taustajärjestelmän hyväksymään binäärimuotoon.

5. Pyyntö välitetään taustajärjestelmälle.

6. Taustajärjestelmä palauttaa vastauksen binäärimuodossa.

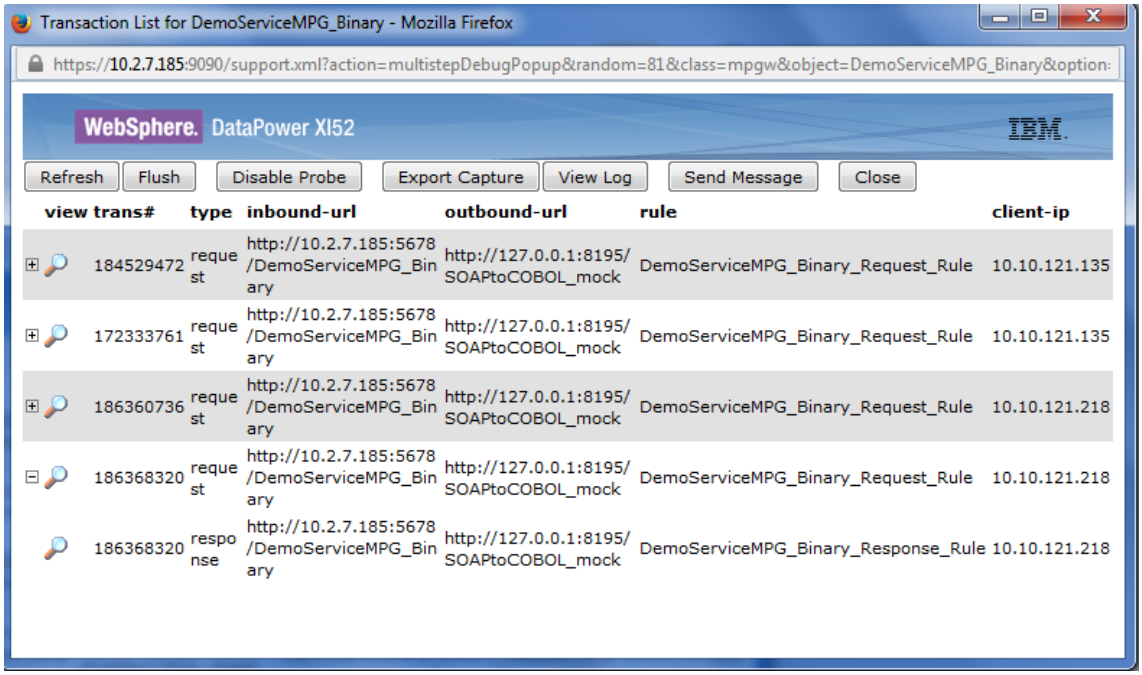
7. DemoServiceMPG_Binary-palvelu muuntaa binäärimuotoisen vastauksen välilliseen XML-muotoon FFD-määrityksen mukaisesti.

8. DemoServiceMPG_Binary-palvelu muuntaa väli-XML-vastauksen kuluttajan hyväksymään SOAP-muotoon ja välittää vastauksen kuluttajalle.

5.6.3 Sanomamuunnoksen testaaminen ja tarkasteleminen

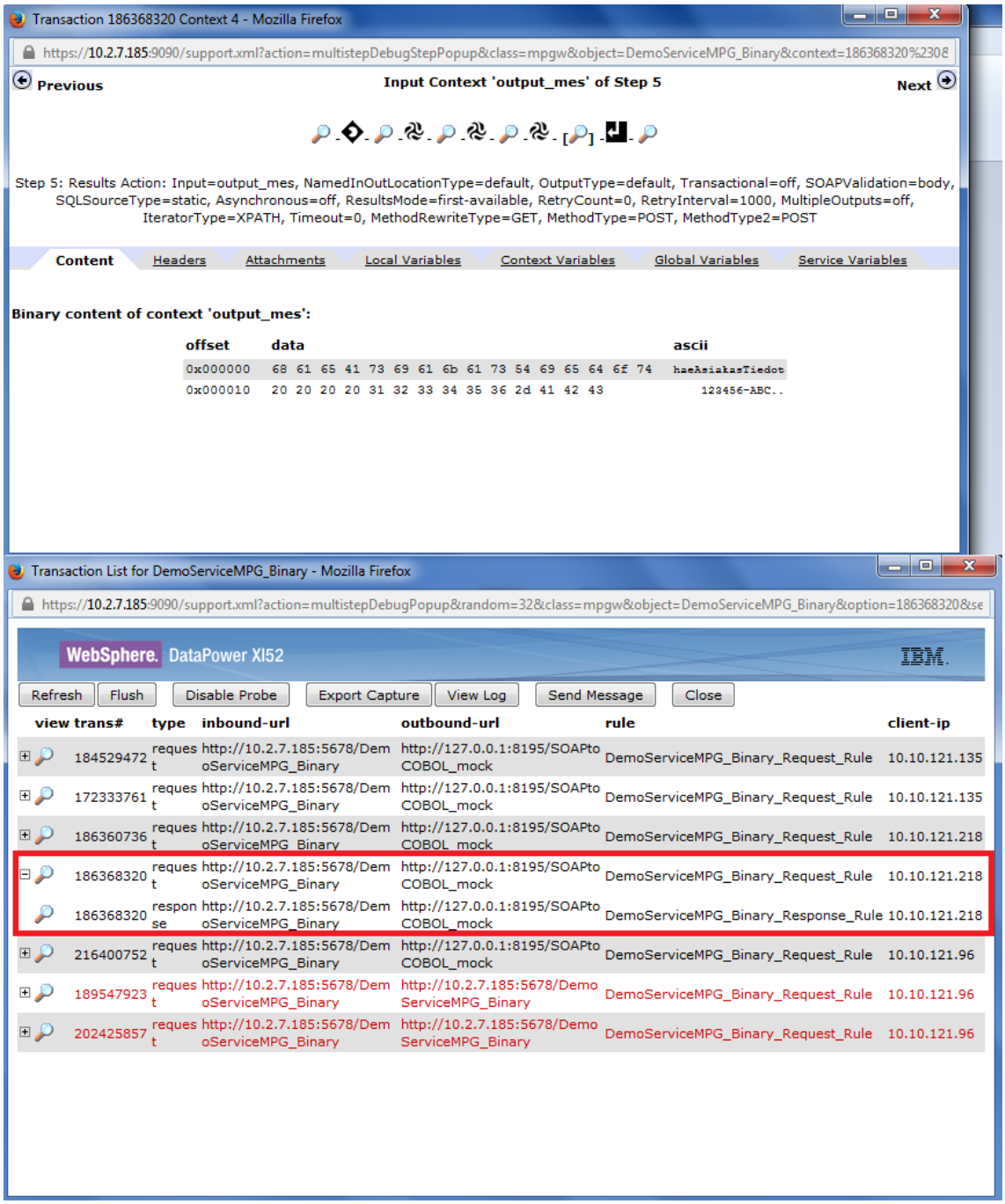
Jokaisella DataPower-palveluobjektilla on käytössä probe-niminen ominaisuus. Vaikka DataPower-laitteen lokeista näkee, miten sanoma on kulkenut ja miten se on onnistunut, proben kautta näkee paremmin, miten itse sanomaprosessointi on mennyt ja kuinka mikäkin prosessointitoiminto on vaikuttanut sanomaan. Jotta palveluobjektille tehdyt pyynnöt näkyisivät probessa, se pitää ensin laittaa käyntiin Enable Probe -painikkeella.

Probe näyttää kuvan 15 mukaisesti 25 viimeisintä pyyntöä, jotka palvelulle on tehty. Jokaisen sanomatransaktion osan, kuten pyynnön, tarkempia tietoja voi tarkastella tarkemmin klikkaamalla vieressä näkyvää suurennuslasia. Transaktion sisältö voidaan avata suurennuslasin viereisestä ikonista, jolloin pyyntöön liittyvät prosessointisäännöt, kuten vastauksen käsittely, tulevat näkyville. Yhteen transaktioon voi liittyä useita prosessointikohtia riippuen siitä, kuinka montaa prosessointisääntöä on kutsuttu transaktion aikana.



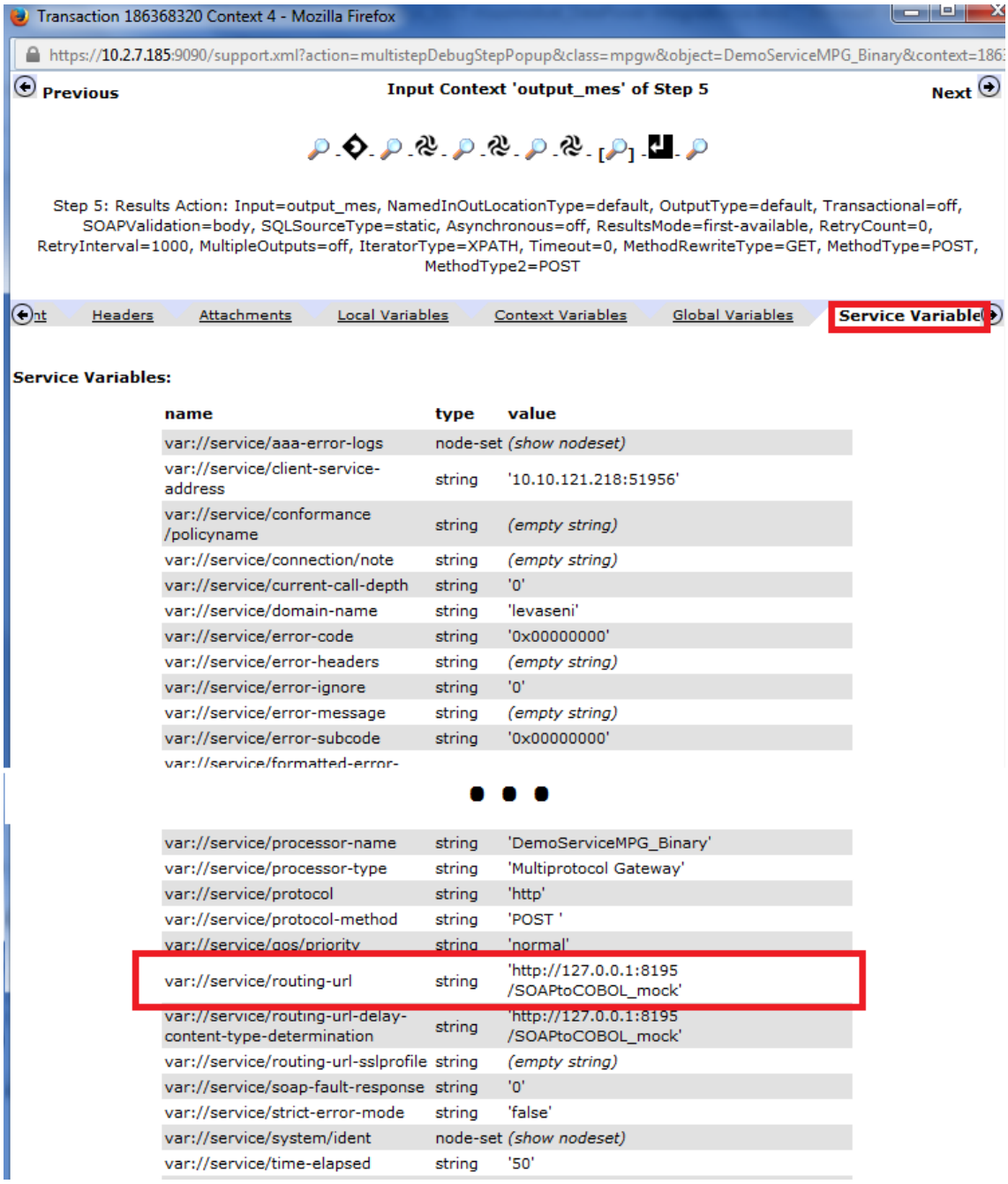
Kuva 15. DataPower probe -näkymä. Tässä näkymässä nähdään transaktioiden pyyntösanomat ja niihin liittyvät muut sanomat. Alin transaktio, joka on tuorein sanoma, on avattu suurennuslasin viereisestä ikonista.

Transaktion ikkunassa nähdään vaihe vaiheelta, miten sanoma muuttuu. Sen eri välilehdillä nähdään myös, mitä muuttujia on asetettu. Tässä ikkunassa nähdään myös mahdolliset virhesanomat, joita sanomakäsittelyssä on tapahtunut.



Kuva 16. DataPower probe -näkymä alla ja yllä siihen liittyvän pyynnön transaktioikkuna. Transaktioikkunan suurennuslasien kohdalta voidaan tarkastella, miltä sanoma näyttää ja mitä muuttujia on mahdollisesti asetettu.

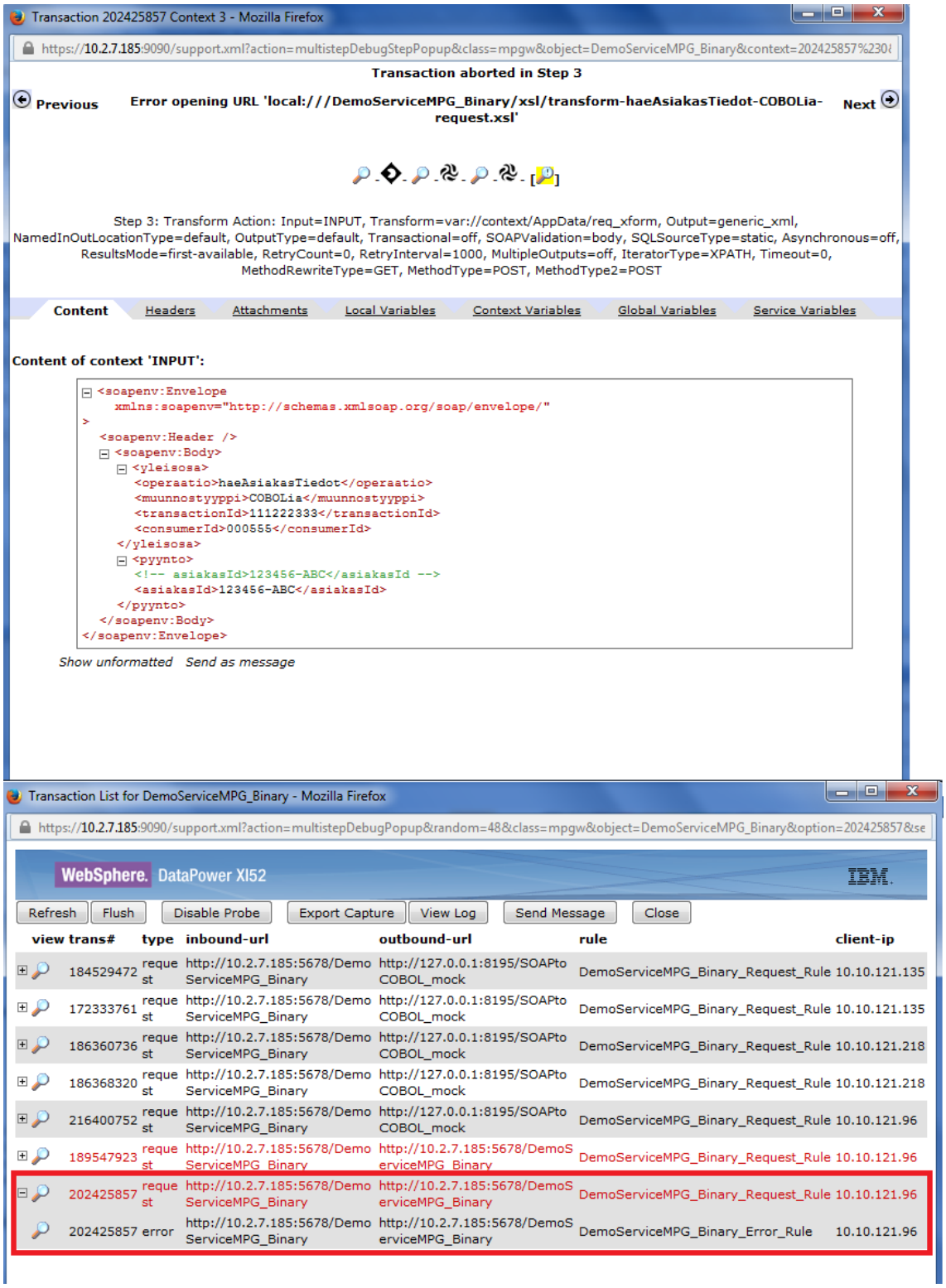
Kuvan 16 mukaisesti nähdään proben avulla onnistunut pyyntösanoman muunnos, jossa SOAP-muotoinen pyyntösanoma on muutettu binäärimuotoon. Proben avulla voidaan vaihe vaiheelta tarkastella, miten sanoma muuttuu ja mitä muuttujia prosessoinnin aikana asetetaan. Tällä tavoin voi kätevästi myös tarkistaa, onko palveluun liittyvät taustajärjestelmän osoitteet asetettu oikein, kuten kuvassa 17 on näytetty.



Kuva 17. DataPower probe -näkymän transaktioikkunan Service Variables -välilehti. Välilehdeltä voi tarkastella palveluun liittyviä muuttujia. Punaisella korostettu muuttuja kertoo sen, minne pyyntö ohjataan.

Kuvan 18 mukaisesti probella näkyy, että pyyntö on mennyt virheeseen ja virheen myötä on kutsuttu erillistä error-tyyppistä prosessointisääntöä. Transaktioikkunassa virhekohta näkyy keltaisella pohjalla olevan suurennuslasin kohdalla. Sitä edeltävässä toiminnossa on tapahtunut virhe. Virhesanoma näkyy transaktioikkunassa lihavoidulla tekstillä. Kuvan esimerkin mukaisesti pyyntösanoman kentässä ”muunnostyyppi” ei ole

kelvollista arvoa ja siksi muunnostiedosto ei ole osannut löytää sanomaan liittyvää muunnostiedostoa DataPowerilta.



Kuva 18. DataPower probe -näkymä alla ja yllä siihen liittyvän pyynnön transaktioikkuna virheti-lanteessa.

6 Best Practices

Tässä luvussa kuvataan joitakin parhaita tai hyväksi todettuja toimintatapoja DataPower-integraatioiden konfigurointiin ja XSLT-ohjelmointiin liittyen. Nämä ovat lähinnä suosituksia eikä niiden käyttö ole välttämätöntä.

6.1 XSLT-ohjelmointi

XSLT-ohjelmoinnissa pätevät monet ohjelmoinnin hyvät käytännöt, kuten koodin sisentäminen ja tarpeeksi kattava kommentointi. Tässä luvussa esitellään nimenomaan XSLT-kieleen liittyviä hyviä käytäntöjä.

XSLT-muunnoksia tehtäessä on parempi käyttää template-elementissä attribuutin name sijasta match-attribuuttia. Nimetyillä templateilla (named templates) voidaan saada koodista järjestyneemmän näköistä, mutta se tuottaa suurissa koodimäärissä ylimääräistä koodia, jonka voi ilmaista yksinkertaisimmin template matchin avulla. Sen sijaan named templates on kätevä erilaisissa sovelluksia ja muunnoksia tukevissa operaatioissa. [XSLT Tips for Cleaner Code and Better Performance 2008.] Tällaisissa tapauksissa named templates toimisi tavallaan funktioina, joita voi käyttää hyödyksi varsinaisen muunnoksen ohjelmoimisessa. Yksittäiset sovelluskohdat, joihin voidaan soveltaa samaa koodipätkää, kannattaa eristää named templatella. Tämä lisää koodin uudelleenkäytettävyyttä. [Bondre.]

XSLT-muuttujia ”element” ja ”attribute” ei kannata käyttää säännöllisesti. Ne vievät enemmän tilaa kuin elementin ja attribuutin kirjoittaminen suoraan [XSLT Tips for Cleaner Code and Better Performance 2008]. Mikäli elementtien nimet ja attribuutit on luotava ajon aikana dynaamisesti, esimerkiksi hakemalla elementtinimet parametriarvoista, silloin voidaan käyttää näitä muuttujia luomaan elementit ja attribuutit [Milowski].

Kun XPathilla valitaan XSLT-muunnoksessa elementtejä ja niiden attribuutteja ja arvoja muokattavaksi, suositeltavaa on välttää juuritasolla käyttämästä ”//” -operaattoria. Isoissa XML-sanomissa kyseinen hakuoperaatio on hyvin raskas ja vie enemmän aikaa, sillä se hakee myös kaikki mahdolliset lapsielementit omineen lapsineen [O’Keefe 2013]. Ylipäätänsä XPathia käytettäessä suositellaan koodin luettavuuden vuoksi käy-

tettäväksi mahdollisimman lyhyitä polkuvalintoja, jotka hakevat vain olennaisimman tiedon.

XML-tiedostot ovat usein kooltaan isoja ja siksi niitä on raskas käsitellä. XSLT-muunnoksissa voidaan ottaa sisennys pois päältä koodipätkällä:

```
<xsl:output method=”xml” indent=”no” />
```

Näin muun muassa XML- ja HTML-sanomien koko pienenee ja sitä myöten sanomien käsittelyyn menee vähemmän aikaa. [O’Keefe 2013.]

6.2 DataPower-integraatioiden konfigurointi

Tässä luvussa kerrotaan yleisesti DataPower-integraatioiden konfigurointiin liittyviä parhaita ja hyviä toimintatapoja. Luvussa ei kuvata kaikkia mahdollisia toimintatapoja, vaan pyritään keskittymään niihin toimintatapoihin, joihin XSLT-muunnokset liittyvät jollakin tavalla. Pääsääntöisesti nämä toimintatavat pyrkivät vähentämään DataPower-laitteen muistin käyttöä.

DataPower laajennusfunktioista joitakin metadata extension functions -tyyppisiä funktioita kannattaa käyttää harkiten. Näistä funktioista esimerkiksi parse(), jota voi käyttää parsimaan sanoma XML-muodossa, kopioi koko sanoman DataPower-laitteen muistiin. Jos sanomat ovat hyvin isoja, se voi alentaa laitteen suorituskykyä. Lisäksi on hyvä huomioida, että DataPower laajennuselementit ja -funktiot olettavat niille annettujen parametrien olevan UTF-8-enkoodattuja. Esimerkiksi ääkköset ISO-8859-1-enkoodatussa XML-rakenteessa voivat aiheuttaa virhesanoman, vaikka parametrina annettu XML-rakenne olisikin rakenteeltaan ja sisällöltään virheetön.

Integraatiopalveluita tehtäessä DataPowerilla ei suositella palveluiden ketjuttamista eli sitä, että DataPowerilla oleva palvelu kutsuu toista laitteen sisällä olevaa palvelua. Joissakin tapauksissa tämä voi olla tarpeellista, jos esimerkiksi halutaan luoda komposiittisia palveluita. Muutoin palveluiden ketjuttamista, varsinkin useamman kuin kahden palvelun ketjuttamista, kannattaa välttää. Tällöin sanomien prosessointia on helpompi tarkastella ja riski sille, että sanomalle tapahtuu käsittelyn aikana jotakin odottamatonta, vähenee. Laitteen muistinkäyttö ja prosessointiteho ovat sitä parempia, mitä vä-

hemmän ketjutettuja palveluita on, koska ketjutettavissa palveluissa syntyy useampi kuin yksi transaktio ja tämän myötä resurssien kulutusta on vaikea ennakoida.

DataPower-muuttujia kannattaa käyttää mahdollisimman vähän, koska niiden käyttö kuluttaa DataPower-laitteen muistia. Mikäli saman toiminnon voi toteuttaa XSLT-muuttujilla, käytetään mieluummin niitä. DataPower-muuttujia kannattaa käyttää lähinnä niissä tapauksissa, kun XSLT-muunnokselta on välitettävä tietoa toiselle XSLT-muunnokselle tai pyynnön käsittelystä pitää välittää tietoa vastauskäsittelylle. Muistin kannalta olisi hyvä välttää tallettamasta isoja XML-rakenteita DataPower-muuttujiin. Sen sijaan niihin kannattaa säilöä lyhyitä merkkijonoja.

Lisäksi samaa DataPower-muuttujaa ei kannata asettaa ja lukea samassa tiedostossa. XSLT-muunnosten prosessointi ei ole suoraviivaista kuten esimerkiksi Java-kielessä ja DataPower-laite optimoi XSLT-muunnoksen suoritusta. Näistä johtuen voi tulla tilanteita, joissa asetettua muuttujaa ei välttämättä olekaan asetettu. Esimerkiksi muuttuja luetaan ennen kuin sille asetetaan arvo. Samasta syystä DataPower-muuttujan ylikirjoittaminen ei ole suositeltavaa. Esimerkiksi saman kontekstimuuttujan asetusta XSLT-muunnoksen for-each-loopissa kannattaa välttää.

Omia konteksteja luotaessa kannattaa itse nimetä konteksti kuin käyttää DataPower-laitteen automaattisesti luomaa kontekstinimeä dpvar. Nämä kontekstit voivat nimensä takia olla ristiriidassa toistensa kanssa ja ylikirjoittaa toistensa arvoja. Tämän seurauksena DataPower ei välttämättä aina osaa tunnistaa, mikä dpvar-konteksti on mikäkin. Kontekstin ylikirjoittamisen vaara pätee mille tahansa kontekstille, joka on samanniminen jonkin toisen kontekstin kanssa. Ylipäätänsä kontekstien käytössä suositeltavaa on käyttää NULL- ja PIPE-konteksteja kuin omia konteksteja. Jokainen itse luotu konteksti vie DataPowerilta muistia, joten niitä kannattaa käyttää vain silloin, kun INPUT-, NULL-, OUTPUT- ja PIPE-kontekstit eivät yksin riitä sanomakäsittelyyn.

Isojen sanomien käsittely voidaan optimoida streamauksen kautta. Streamauksella tarkoitetaan sitä, että sanoma prosessoidaan node kerrallaan. Toimintatapa on samankaltainen kuin SAX-mallin mukaisessa prosessoinnissa, mutta streamauksessa muodostetaan tarvittaessa DOM-mallin mukainen puurakenne niistä osista, joiden käsittely vaatii puurakennetta. Muodostettu puurakenne poistetaan muistista heti, kun sen prosessointi on saatu päätökseen. Kaikkia XSLT-muunnoksia ei kuitenkaan voi streamata. Esimerkiksi sort-tyyppisten operaatioiden ja XPath:n following- ja preceding-akselien

käyttö estävät streamauksen. DataPower-laite käyttää streamausta silloin kun kykenee, ja ne osat, joita XSLT-muunnos ei voi streamata, käsitellään DOM-mallin mukaisesti. Osittainen streamaus on käytössä tilanteissa, joissa käytetään esimerkiksi lauseita xsl:if, xsl:choose tai xsl:when. [Optimizing through Streaming 2009: 1-3.]

Streamauksen hyötynä on tehokkaampi muistin käyttö, koska koko sanomaa ei lueta muistiin ja ne puurakenteet, jotka muodostetaan, poistetaan heti, kun ne on prosessoitu. Osittaisen streamauksen hyötynä on DOM-mallin hyödyntäminen pienemmällä muistinkäytöllä. [Optimizing through Streaming 2009: 1-2.] Streamaus vaatii kuitenkin sanomarakenteen huolellista suunnittelua, sillä mitä valmiimpi rakenne on käsiteltävänä, sitä todennäköisemmin käsittely on tehokas.

7 Yhteenveto

Tässä insinöörityössä oli tavoitteena ensisijaisesti tutkia XSLT-muunnosten käyttöä DataPower-laitteella toteutetussa integraatiossa. Työssä on myös selvitetty yleisiä integraatiokehityksen haasteita ja toimintatapoja, jotta voidaan ymmärtää, mikä merkitys DataPower-laitteella on integraatiokehityksessä ja kuinka se hyödyntää XSLT-muunnoskieltä. Työn käytännön osuus keskittyy kuvaamaan kaksi demointegraatiota, joilla voi havainnollistaa, kuinka XSLT-muunnoksia on käytetty DataPowerilla toteutetussa integraatiossa.

Työn lopputuloksena syntyneet demointegraatiot eivät täysin toteutuneet suunnitelmien mukaisesti, mutta niiden toiminnassa on mukana kaikki olennainen, jota oli tarkoitus havainnollistaa. Monet DataPowerin konfiguraatiot ja niihin liittyvät toimintatavat eivät suoraan liity XSLT-muunnoksiin, joten aiheeseen liittyviä hyviä ja toimivaksi todettuja toimintatapoja oli vaikea löytää. Muutoin työn tavoitteisiin on päästy hyvin.

Lähteet

Bondre, P. XSLT – Efficient Programming Techniques. PDF -verkkodokumentti. <http://www.xml.org//sites/www.xml.org/files/xslt_efficient_programming_techniques.pdf>. Luettu 28.7.2014.

Chase, N. Understanding web services specifications, Part 1: SOAP. Verkkodokumentti. <http://www.ibm.com/developerworks/webservices/tutorials/ws-understand-web-services1/ws-understand-web-services1.html>. Päivitetty 12. toukokuuta 2006. Luettu 4.8.2014.

Curry, E. Message-Oriented Middleware. PDF -verkkodokumentti. <http://www.edwardcurry.org/publications/curry_MfC_MOM_04.pdf>. Luettu 1.8.2014.

Davis, J. 2009. Open Source SOA. Ensimmäinen painos. Manning Publications Co.

Ebbers, Barrus, Bonazebi, Daly, Lee. DataPower Architectural Design Patterns: Integrating and Securing Web Services Across Domains. PDF -verkkodokumentti. <http://www.redbooks.ibm.com/redbooks/pdfs/sg247620.pdf>. Päivitetty lokakuussa 2008. Luettu 1.5.2014.

EXSLT. Verkkodokumentti. EXSLT. <http://www.exslt.org/>. Luettu 29.7.2014.

Extension elements and functions. PDF -verkkodokumentti. IBM Info Center. <http://pic.dhe.ibm.com/infocenter/wsdatap/v6r0m1/index.jsp>. Luettu 5.11.2014.

Introduction. Verkkodokumentti. Enterprise Integration Patterns. <http://www.eaipatterns.com/Introduction.html>. Luettu 31.7.2014.

Introduction to Web Services. Verkkodokumentti. W3 Schools. <http://www.w3schools.com/webservices/ws_intro.asp>. Luettu 4.8.2014.

Introduction to WSDL. Verkkodokumentti. W3 Schools. <http://www.w3schools.com/webservices/ws_wsdl_intro.asp>. Luettu 29.7.2014.

Kaplan, Bechtold, Dickerson, Kinard, Mitra, Mota, Shute, Walczyk. DataPower SOA Appliance Administration, Deployment and Best Practices. PDF-verkkodokumentti. <http://www.redbooks.ibm.com/redbooks/pdfs/sg247901.pdf>. Päivitetty kesäkuussa 2011. Luettu 31.7.2014.

Kay, M. 2000. XSLT Programmer's Reference. Ensimmäinen painos. Wrox Press.

Kodali, R. What is service-oriented architecture? An introduction to SOA. Verkkodokumentti. <http://www.javaworld.com/article/2071889/soa/what-is-service-oriented-architecture.html>. Päivitetty 13.6.2005. Luettu 29.7.2014.

Lorenz, M. IBM Certified SOA Solution Designer certification prep, Part 1: SOA best practices. Verkkodokumentti. <http://www.ibm.com/developerworks/webservices/tutorials/ws-soacert1/ws-soacert1.html> Päivitetty 10.10.2006. Luettu 29.10.2014

Milowski, A. There are Monsters in My Closet or How Not to Use XSLT. Verkkodokumentti. <http://www2.sims.berkeley.edu/academics/courses/is290-8/s04/lectures/5/dragons/allslides.html>. Luettu 28.7.2014.

O'Keefe, R. XSLT Best Practices. Verkkodokumentti. <http://superdevelopment.com/2013/08/23/xslt-best-practices/>. Päivitetty 23.8.2013. Luettu 28.7.2014.

Optimizing through Streaming. PDF-verkkodokumentti. <ftp://ftp.software.ibm.com/software/integration/datapower/library/prod_docs/Misc/OptimizingThroughStreaming-v1.pdf> Päivitetty toukokuussa 2009. Luettu 3.11.2014.

J. Rodriguez, Adiraju, Gauci, Grohmann, Holmes, Moody, Muralidharan, Ramirez, A. Rodriguez. IBM WebSphere DataPower SOA Appliances Part I: Overview and Getting Started. PDF -verkkodokumentti. <http://www.redbooks.ibm.com/redpapers/pdfs/redp4327.pdf>. Päivitetty huhtikuussa 2008. Luettu 25.4.2014.

Rouse, M. Service-oriented architecture (SOA). Verkkodokumentti. <http://searchsoa.techtarget.com/definition/service-oriented-architecture>. Päivitetty elokuussa 2008. Luettu 29.7.2014.

Request/Response. Verkkodokumentti. Service Design Patterns. <http://www.servicedesignpatterns.com/ClientServiceInteractions/RequestResponse>. Luettu 31.7.2014.

Snell, Tidwell, Kulchenko. Programming Web Services with SOAP. 2002. Ensimmäinen painos. O'Reilly Media.

SOAP Introduction. Verkkodokumentti. W3 Schools. <http://www.w3schools.com/webservices/ws_soap_intro.asp>. Luettu 4.8.2014.

Solving Integration problems with Patterns. Verkkodokumentti. Enterprise Integration Patterns. <http://www.eaipatterns.com/Chapter1.html>. Luettu 31.7.2014.

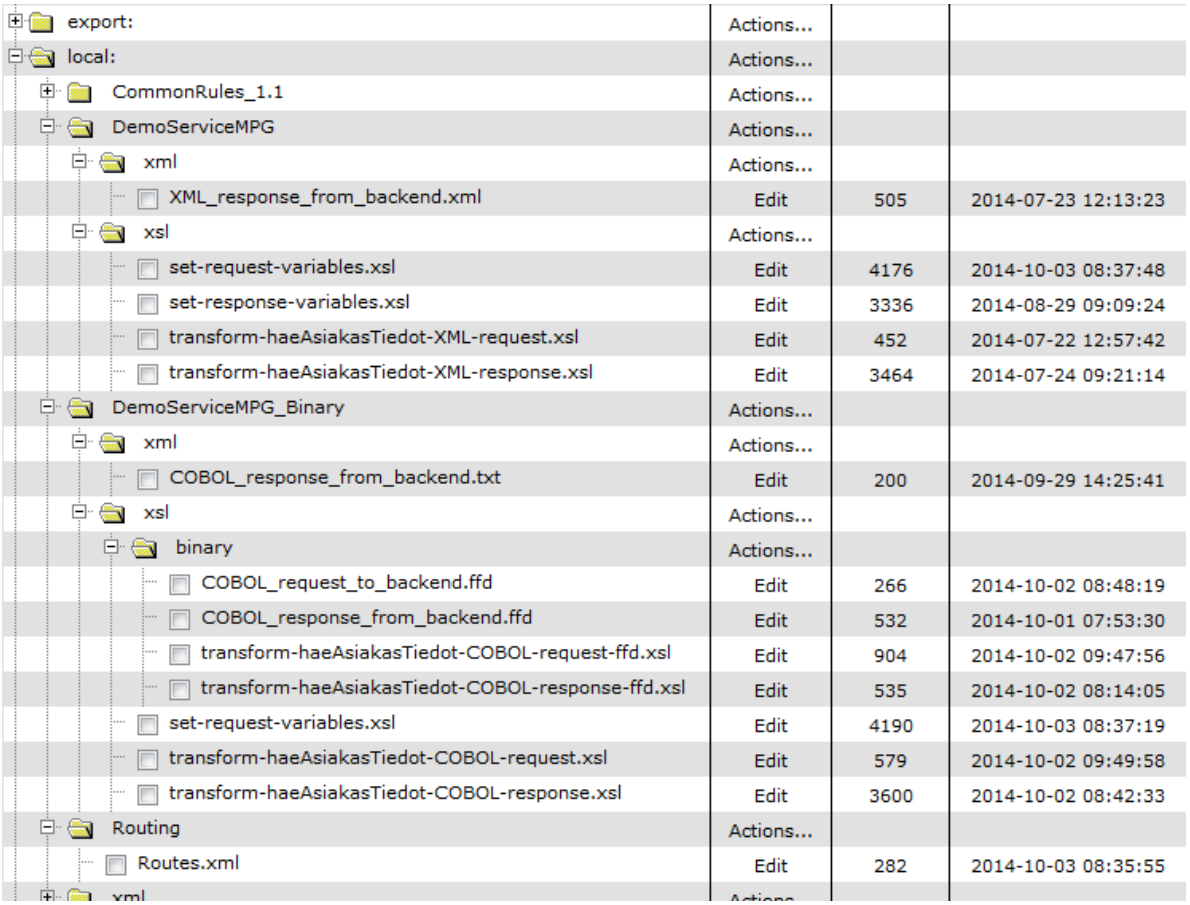
Tidwell, D. 2008. XSLT, Second Edition. Toinen painos. O'Reilly Media.

What are Web Services. Verkkodokumentti. Tutorials Point. <http://www.tutorialspoint.com/webservices/what_are_web_services.htm>. Luettu 4.8.2014.

XSLT Tips for Cleaner Code and Better Performance. Verkkodokumentti. Onenaught. <http://www.onenaught.com/posts/23/xslt-tips-for-cleaner-code-and-better-performance>. Päivitetty syyskuussa 2008. Luettu 28.7.2014.

XSL Transformations (XSLT) Version 2.0. Verkkodokumentti. W3C. <http://www.w3.org/TR/xslt20/>. Luettu 25.7.2014.

Liite 1: DemoService-integraatioiden hakemistorakenne DataPower-laitteella



export:	Actions...		
local:	Actions...		
CommonRules_1.1	Actions...		
DemoServiceMPG	Actions...		
xml	Actions...		
XML_response_from_backend.xml	Edit	505	2014-07-23 12:13:23
xsl	Actions...		
set-request-variables.xsl	Edit	4176	2014-10-03 08:37:48
set-response-variables.xsl	Edit	3336	2014-08-29 09:09:24
transform-haeAsiakasTiedot-XML-request.xsl	Edit	452	2014-07-22 12:57:42
transform-haeAsiakasTiedot-XML-response.xsl	Edit	3464	2014-07-24 09:21:14
DemoServiceMPG_Binary	Actions...		
xml	Actions...		
COBOL_response_from_backend.txt	Edit	200	2014-09-29 14:25:41
xsl	Actions...		
binary	Actions...		
COBOL_request_to_backend.ffd	Edit	266	2014-10-02 08:48:19
COBOL_response_from_backend.ffd	Edit	532	2014-10-01 07:53:30
transform-haeAsiakasTiedot-COBOL-request-ffd.xsl	Edit	904	2014-10-02 09:47:56
transform-haeAsiakasTiedot-COBOL-response-ffd.xsl	Edit	535	2014-10-02 08:14:05
set-request-variables.xsl	Edit	4190	2014-10-03 08:37:19
transform-haeAsiakasTiedot-COBOL-request.xsl	Edit	579	2014-10-02 09:49:58
transform-haeAsiakasTiedot-COBOL-response.xsl	Edit	3600	2014-10-02 08:42:33
Routing	Actions...		
Routes.xml	Edit	282	2014-10-03 08:35:55
xml	Actions...		

Liite 2: Pyynnön muuttujien asetus DataPowerilla

set-request-variables.xsl (DemoServiceMPG_Binary -hakemistossa)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
            xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            xmlns:dp="http://www.datapower.com/extensions"
            xmlns:dpconfig="http://www.datapower.com/param/config"
            extension-element-prefixes="dp dpconfig"
            exclude-result-prefixes="dp dpconfig">

<!--
* Maaritetaan DataPower Stylesheet -parametrit, jotka on annettu DataPower -
palveluobjektille valilehdella "Stylesheet Parameters" Talla maarityksella
selvitetaan, minkalaisena tehdaan binaarimuunnos, mikali sellaista tullaan
tekemaan.
* Tama on esimerkkina siita, miten XSLT pystyy DataPower -lisaosien kautta
hakemaan palveluobjektille maaritettyja parametreja.
-->
<xsl:param name="dpconfig:req-xFormbin-type" />
<dp:param name="dpconfig:req-xFormbin-type" type="'dmString'">
     <display>Request Binary Transform Type</display>
     <description>
            Kertoo, millaista binaarimuunnosta kaytetaan pyynnossa.
     </description>
     <default></default>
</dp:param>

<xsl:template match="/">
     <!-- Otetaan yleisosa -elementti talteen ja tallennetaan se DataPower
     -kontekstimuuttujaan. -->
     <xsl:variable name="yleisosa">
         <!-- copy-of kopioi elementin sisaltoineen, siksi sita on
         kaytetty tassa -->
         <!--xsl:copy-of select="/*[local-name()='Envelope']/*[local-name()
         = 'Body']/*[local-name() = 'yleisosa']"/-->
         <xsl:copy-of select="//yleisosa"/>
     </xsl:variable>
     <dp:set-variable
         name= "'var://context/AppData/yleisosa'"
         value= "$yleisosa"/>

     <!-- Otetaan yleisosasta talteen operaation nimi ja muunnostyyppi. -->
     <xsl:variable name= "operaatio" select= "$yleisosa/yleisosa/operaatio"/>
     <dp:set-variable name= "'var://context/AppData/operaatio'"
         value= "$operaatio"/>
     <xsl:variable name= "muunnostyyppi"
         select= "$yleisosa/yleisosa/muunnostyyppi"/>
     <dp:set-variable name= "'var://context/AppData/muunnostyyppi'"
         value= "$muunnostyyppi"/>

     <!--
     Haetaan operaation muunnostyypin peruteella taustajarjestelman osoite ja
     asetetaan se DataPower -palvelumuuttujaan. Reititystiedot saadaan
     erillisestä tiedostosta.
     -->
     <xsl:variable name= "reittifile"
         select= "string('local:///Routing/Routes.xml')"/>
     <xsl:variable name= "reititykset" select= "document($reittifile)"/>
     <dp:set-variable name= "'var://service/routing-url'"
         value= "$reititykset/Routes/Operation/type[@name=$muunnostyyppi
         /text()]/route/text()"/>

     <!-- Asetetaan muunnostiedosto SOAP-to-XML -muunnokselle. Tama tehdaan
      seka SOAP-to-XML -tapauksissa ja SOAP-to-COBOL -tapauksissa. -->
     <xsl:variable name="xform-path">
```

```
            <xsl:value-of select=
               "concat('local:///DemoServiceMPG_Binary/xsl/transform-',
               $operaatio, '-', $muunnostyyppi, '-request.xsl')"/>
         </xsl:variable>
         <dp:set-variable name="'var://context/AppData/req_xform'"
         value="string($xform-path)"/>

         <!-- Tarkistetaan, kaytetaanko binaarimuunnosta. Jos kaytetaan, valitaan
      binaarimuunnoksen tyypin perusteella muunnostiedosto. -->
         <xsl:variable name="xformbin-path">
            <xsl:choose>
               <!-- Kaytetaan .dpa -tiedostoa binaarimuunnosta varten. -->
               <!--xsl:when test="string($dpconfig:req-xFormbin-type) = 'DPA'">
               <xsl:value-of select=
               "concat('local:///DemoServiceMPG/xsl/binary/transform-',
               $operaatio, '-', $muunnostyyppi, '-request.dpa')"/>
               </xsl:when-->
               <!-- Kaytetaan .ffd -maarityksen mukaista XSLT -tiedostoa binaari-
               muunnosta varten. -->
               <xsl:when test="string($dpconfig:req-xFormbin-type) = 'FFD'
               and $muunnostyyppi='COBOL'">
                  <xsl:value-of select=
                  "concat('local:///DemoServiceMPG_Binary/xsl/binary/transform-',
                  $operaatio, '-', $muunnostyyppi, '-request-ffd.xsl')"/>
               </xsl:when>
               <xsl:otherwise>
                  <xsl:message>
                  Binaarimuunnosta ei tehda pyyntosanomalle.
                  </xsl:message>
               </xsl:otherwise>
            </xsl:choose>
         </xsl:variable>
         <dp:set-variable name="'var://context/AppData/req_xformbin'"
         value="string($xformbin-path)"/>

         <!--
         * Mikali binaarimuunnos tehdaan, nama elementtitiedot luodaan ja niita
         kaytetaan DataPower -palveluobjektissa hakemaan haluttu binaarimuunnos-
         toiminto.
         -->
         <actions>
            <xsl:if test= "string-length($dpconfig:req-xFormbin-type) != 0
            and $muunnostyyppi='COBOL'">
               <xFormbin>
                  <xsl:value-of select="$dpconfig:req-xFormbin-type"/>
               </xFormbin>
            </xsl:if>
         </actions>
      </xsl:template>
      </xsl:stylesheet>
```

Liite 3: Vastauksen muunnos DemoServiceMPG-integraatiopalvelussa

transform-haeAsiakasTiedot-XML-response.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
            xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            xmlns:dp="http://www.datapower.com/extensions"
            xmlns:date="http://exslt.org/dates-and-times"
            extension-element-prefixes="dp date"
            exclude-result-prefixes="dp date"
>
<!-- excluse-result-prefixes estaa ylimaaraisten namespace -maaritysten kopi-
oimisen vastaussanomaan -->

<xsl:output  encoding="UTF-8"  indent="yes"  omit-xml-declaration="yes"  meth-
od="xml" version="1.0"/>

<xsl:template match="/">

     <!--xsl:variable
     name="soap-namespace"
     select="'http://schemas.xmlsoap.org/soap/envelope'"/-->

     <!--
     * SOAP -namespace on asetettu pyyntosanoman kasittelyssa DataPower
     -palveluobjektissa. Nyt tuo arvo haetaan vastaussanomalle, jotta vastaus-
     sanoma palautuu samalla SOAP -namespacella kuluttajalle.
     -->
     <xsl:variable
            name="soap-namespace"
            select="dp:variable('var://context/AppData/soap_namespace')"/>

     <!-- Luodaan SOAP -kuoret sanomasisallon ymparille. -->
     <xsl:element name="soapenv:Envelope" namespace="{$soap-namespace}">
     <xsl:element name="soapenv:Header" namespace="{$soap-namespace}"/>
     <xsl:element name="soapenv:Body" namespace="{$soap-namespace}">
     <!-- Kopioidaan yleisosa DataPower -kontekstimuuttujasta. -->
     <xsl:copy-of select="dp:variable('var://context/AppData/yleisosa')"/>
     <!-- kasitellaan vastaussanoman sisalto -->
     <vastaus>
         <asiakasId><xsl:value-of select="//asiakastunnus"/></asiakasId>
         <asiakasEtunimi><xsl:value-of select="//etunimi"/></asiakasEtunimi>
         <asiakasSukunimi>
            <xsl:value-of select="//sukunimi"/>
         </asiakasSukunimi>
         <asiakasOsoite><xsl:value-of select="//lahiosoite"/></asiakasOsoite>
         <asiakasPostiNro>
            <xsl:value-of select="//postinumero"/>
         </asiakasPostiNro>
         <asiakasPostiToimPaik>
            <xsl:value-of select="//postitoimipaikka"/>
         </asiakasPostiToimPaik>
         <!--
         Tassa elementissa on hyodynnetty DataPower -funktiota "decode", jolla
         dekoodataan vastauksessa tuleva tietosisalto kuluttajan puolelle.
         Loytyy myos funktio "encode", joka tekisi enkoodauksen.
         -->
         <asiakasLisaTiedot>
            <xsl:value-of
            select="dp:decode(//enkoodattulisatieto,
                        'base-64')"/>
         </asiakasLisaTiedot>
         <!--
         Alla oleva template -kutsu luo liittymisPvm -elementin, jos sita on.
         Se myos konvertoi paivamaaran toiseen muotoon. Valmis elementti
         nayttaisi esim. talta:
            <liittymisPvm>2012-12-17</liittymisPvm>
         -->
```

```
            <xsl:call-template name="createDateToConsumerIfExists">
               <xsl:with-param name="value">
                  <xsl:value-of select="//liittymispaiva"/>
               </xsl:with-param>
               <xsl:with-param name="to">liittymisPvm</xsl:with-param>
            </xsl:call-template>
      </vastaus>
      </xsl:element> <!-- soapenv:Body -->
      </xsl:element> <!-- soapenv:Envelope -->
</xsl:template>

<!-- Erillinen template, jolla muunnetaan paivamaaratiedot muodosta YYYYMMDD
muotoon YYYY-MM-DD -->
<xsl:template name="createDateToConsumerIfExists">
      <xsl:param name="value"/>
      <xsl:param name="to"/>
      <!-- Example input:            20021213-->
      <!-- Example output:           2002-12-13 -->
      <xsl:if test="string-length() &gt; 0">
            <xsl:variable name="xformedDate">
               <xsl:value-of select="concat(substring($value, 1, 4),
                  '-', substring($value, 5, 2), '-',
                  substring($value, 7, 2))"/>
            </xsl:variable>
            <xsl:variable name="outputDate">
               <xsl:value-of select="date:date($xformedDate)"/>
            </xsl:variable>
            <xsl:element name="{$to}">
               <xsl:value-of select="$outputDate"/>
            </xsl:element>
      </xsl:if>
</xsl:template>

</xsl:stylesheet>
```

Liite 4: Binäärimuunnokset

transform-haeAsiakasTiedot-COBOL-request-ffd.xsl (väliXML-muodosta binäärimuotoon)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
            xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            xmlns:dp="http://www.datapower.com/extensions"
            extension-element-prefixes="dp"
>

     <dp:output-mapping
href="local:///DemoServiceMPG_Binary/xsl/binary/COBOL_request_to_backend.ffd"
type="ffd"/>

     <xsl:template match="/">
         <xsl:copy-of select="."/>
     </xsl:template>

</xsl:stylesheet>
```

transform-haeAsiakasTiedot-COBOL-response-ffd.xsl (binäärimuodosta väliXML-muotoon)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
            xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            xmlns:dp="http://www.datapower.com/extensions"
            extension-element-prefixes="dp"
>

<xsl:output method="xml" encoding="UTF-8" version="1.0"/>

<dp:input-mapping href="COBOL_response_from_backend.ffd" type="ffd"/>

<xsl:template match="/">
     <xsl:copy-of select="."/>
</xsl:template>

</xsl:stylesheet>
```

Liite 5: Binääripyynnön ja -vastauksen ffd-tiedostot

COBOL_Request_to_backend.ffd

```
<?xml version="1.0" encoding="UTF-8"?>
<File version="1.1" name="PYYNTO" syntax="syn">
     <Syntax name="syn" encoding=" UTF-8"/>
     <Field name="OPERAATIO" length="20"/>
     <Group name="ASIAKASTIEDOT-INPUT">
         <Field name="ASIAKASTUNNUS" length="10"/>
     </Group>
</File>
```

COBOL_Response_from_backend.ffd

```
<?xml version="1.0" encoding="UTF-8"?>
<File version="1.1" name="VASTAUS" syntax="syn">
     <Syntax name="syn" encoding=" UTF-8"/>
     <Group name="ASIAKASTIEDOT-OUTPUT">
         <Field name="ASIAKASTUNNUS" length="10"/>
         <Field name="ETUNIMI" length="10"/>
         <Field name="SUKUNIMI" length="10"/>
         <Field name="LAHIOSOITE" length="20"/>
         <Field name="POSTINUMERO" length="5"/>
         <Field name="POSTITOIMIPAIKKA" length="15"/>
         <Field name="LIITTYMISPAIVA" length="10"/>
         <Field name="ENKOODATTULISATIETO" length="120"/>
     </Group>
</File>
```