Lappeenrannan teknillinen yliopisto
Teknistaloudellinen tiedekunta
Tietotekniikan osasto
Tietoliikennetekniikan laboratorio

PALVELUVÄYLÄÄ HYÖDYNTÄVÄT JÄRJESTELMÄINTEGRAATIOT

Työn tarkastajana on toiminut professori Jari Porras.

Työn ohjaajana on toiminut diplomi-insinööri Teemu Karvonen.

Aleksi Mustonen
Prikaatintie 4 as. 6
45100 Kouvola

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Teknistaloudellinen tiedekunta
Tietotekniikan osasto
Tietoliikennetekniikan laboratorio

Aleksi Mustonen

Palveluväylää hyödyntävät järjestelmäintegraatiot

Diplomityö

2009

90 sivua, 10 kuvaa ja 1 taulukko

Tarkastajat: professori Jari Porras
diplomi-insinööri Teemu Karvonen

Hakusanat: järjestelmäintegraatiot, palvelukeskeinen arkkitehtuuri, palveluväylä, web-palvelut, web-palveluprotokollat
Keywords: system integrations, service oriented architecture, enterprise service bus, web services, web service protocols

Tässä työssä kuvataan Lahti Fenix Kuntalaistilijärjestelmän ja Tekla Xcity-järjestelmän välille toteutetun järjestelmäintegraation eri vaiheet. Kuntalaistilijärjestelmä on Lahden kaupungin Fenix-hankkeessa kehitteillä oleva sähköinen palvelualusta, jota pitkin kuntalaisille tarjotaan erilaisia kunnallisia palveluja, kuten vastaanottoaikoja hammaslääkärille. Tekla Xcity on kuntien ja kaupunkien käyttöön tarkoitettu järjestelmä, josta on mahdollista hakea esimerkiksi henkilö- ja paikkatietoja.

Aluksi työssä esitellään lyhyesti erilaisia tapoja toteuttaa järjestelmäintegraatioita. Seuraavaksi kiinnitetään erityistä huomiota niin sanottuihin web-palveluihin, joiden etuja ja haittoja arvioidaan käytännön esimerkin kautta. Tässä pidetään viitekehyksenä Kuntalaistilijärjestelmää ja siinä käytettyä palvelukeskeistä arkkitehtuuria. Arkkitehtuurin ja viestiliikenneratkaisujen arvioinnin jälkeen siirrytään käytännön osuuteen, jossa itse järjestelmäintegraatio toteutetaan.

Järjestelmäintegraatio toteutetaan käyttäen avoimen lähdekoodin palveluväylää ja sille saatavissa olevia viestintäkehyksiä. Integraation eri vaiheissa tutustutaan erilaisiin viestiliikenneprotokolliin ja niiden käyttöön valittujen viestintäkehysten kanssa. Kunkin protokollan toimivuus varmennetaan analysoimalla integraatioon liittyvien komponenttien ja päätepisteiden välistä tietoliikennettä.

ABSTRACT

Lappeenranta University of Technology
Faculty of Technology Management
Department of Information Technology
Laboratory of Communications Engineering

Aleksi Mustonen

System integrations taking advantage of an enterprise service bus

Thesis for the Degree of Master of Science in Technology

2009

90 pages, 10 figures and 1 table

Examiners: Professor Jari Porras
Master of Science Teemu Karvonen

Keywords: system integrations, service oriented architecture, enterprise service bus, web services, web service protocols

This work describes different stages involved in an integration between Lahti Fenix Kuntalaistilijärjestelmä (”Citizens’ account”) and Tekla Xcity system. Developed as a part of the Lahti Fenix project, Kuntalaistilijärjestelmä is a platform through which different kinds of municipal services can be electronically offered. These services can include e.g. reservations for dentist. Tekla Xcity is a system that holds, for instance, personal and location based information, and is aimed for municipal use.

At first, this project describes different ways to integrate systems. Then, a special emphasis is given on so called web services, the advantages and disadvantages of which are evaluated using Kuntalaistilijärjestelmä and its service oriented architecture as a reference. After the evaluation of selected architecture and messaging solutions, we move to the practical part of this work. In this part, we carry out the actual integration between the two systems mentioned earlier.

Interoperability between Kuntalaistilijärjestelmä and Tekla Xcity is achieved through the employment of an open-source enterprise service bus and compatible messaging frameworks. In different phases of the integration, various communication protocols are introduced, and we describe how they can be used with selected messaging frameworks. Communications between different components are analyzed to ensure the functionality of each protocol employed.

SISÄLLYSLUETTELO

1 JOHDANTO....8

2 KUNTALAISTILIJÄRJESTELMÄN YLEISKUVAUS....11

2.1 Kuntalaistili....11

2.2 Tilanvarausjärjestelmä....12

2.3 Ajanvarausjärjestelmä....13

3 JÄRJESTELMÄINTEGRAATIOT PALVELUKESKEISESSÄ ARKKITEHTUURISSA....14

3.1 Järjestelmien integrointi eri tasoilla....14

3.2 XML-pohjaisten merkkauskielten hyödyntäminen järjestelmäintegraatioissa....15

3.3 Web-palvelut....18

3.4 SOAP-viestien eri variaatiot....18

3.4.1 SOAP doc/lit-enkoodattuna....20

3.4.2 SOAP rpc/enc-enkoodattuna....20

3.4.3 SOAP rpc/lit- tai doc/lit wrapped -enkoodattuna....21

3.5 Integraatiot olemassa olevia rajapintoja käyttäen....21

4 KUNTALAISTILIJÄRJESTELMÄN TEKNINEN JÄRJESTELMÄ- JA ARKKITEHTUURIKUVAUS....23

4.1 Looginen kerrosarkkitehtuuri....23

4.2 Web-palveluiden granulariteetti Kuntalaistilijärjestelmässä....25

4.3 Järjestelmän palveluväyläarkkitehtuuri ja liitosrajapinnat....26

4.3.1 Järjestelmän julkiset rajapinnat....28

4.3.2 Järjestelmän yksityiset rajapinnat....29

4.3.3 Ulkoisten järjestelmien rajapinnat....29

4.4 Fyysinen arkkitehtuuri 29
4.5 Havaintoja Kuntalaistilijärjestelmän arkkitehtuurista ja sen toteuttamistekniikoista 32
4.5.1 Arkkitehtuuri- ja teknologiaratkaisujen arvioituja etuja 32
4.5.2 Arkkitehtuuri- ja teknologiaratkaisujen arvioituja haittoja 33
4.6 Web-palvelut ja järjestelmän suorituskyky 34
5 KUNTALAISTILIJÄRJESTELMÄÄN TOTEUTETTU YLEINEN PALVELUKUTSUMUODOSTIN 35
5.1 Palvelupyyntöjen välitys palveluväylän läpi 36
5.2 Yleisen palvelukutsumallin laatiminen sisäisissä integraatioissa käytettäväksi 38
5.3 Palvelukutsujen reitittäminen Mule-sovelluskehyksen avulla 39
6 KUNTALAISTILIJÄRJESTELMÄN ULKOISET JÄRJESTELMÄINTEGRAATIOT 41
6.1 Ulkoisten järjestelmien integraatiomalli 41
6.2 Web-palveluprotokollien eroista 43
7 INTEGRAATIO TEKLA XCITY-JÄRJESTELMÄÄN 45
7.1 Integraatiomoduulin käyttöönotto 45
7.2 Palveluiden olemassaolon varmistaminen 48
7.3 Integraatiomoduulin kehittäminen 50
7.3.1 Kuljetusolioiden toteutus 50
7.3.2 Reifikaatio geneeristen elementtien reflektoinnissa 52
7.4 Ulkoisen palvelun kutsuminen järjestelmän läpi 55
7.5 Integraatiopalveluiden reitityskonfiguraatio 56
7.6 Kuljetuskerroksen salaus 64
7.7 Asiakkaan auktorisointi sovelluskerroksella 66
7.8 Web-palvelutason reititys 69

7.9 Web-palvelutason salaus 71
7.10 Kokonaiskuva integraatioarkkitehtuurista 75
8 JOHTOPÄÄTÖKSET 79
LÄHDELUETTELO 82

KUVA- JA TAULUKKOLUETTELO

Kuva 1: Kuntalaistilijärjestelmän looginen kerrosarkkitehtuuri ja kerrosten vastuualueet [Propentus2008] 24
Kuva 2: Järjestelmien välinen federointi palvelualustaa käyttäen [Propentus2008] 25
Kuva 3: Kuntalaistilijärjestelmän yleinen looginen arkkitehtuuri [Propentus2008] 27
Kuva 4: Palveluväyläkerrokselle saapuvat palvelupyynnöt aiheuttavat automaattisen kutsun käyttöoikeuksien tarkastamiseen, kirjausketjun muodostamiseen ja palvelupyyntöstatistiikan ylläpitoon ennen palvelupyynnön ohjaamista eteenpäin [Propentus2008] 28
Kuva 5: Esimerkki Kuntalaistilijärjestelmän fyysisestä palvelinympäristöstä [Propentus2008] 30
Kuva 6: ESB-palvelun läpileikkaus [Propentus2008] 37
Kuva 7: Ulkoisten järjestelmien integraatioarkkitehtuuri 42
Kuva 8: Xcity-integraatioon toteutettu palveluväylätason arkkitehtuuri 76
Kuva 9: Integraatiossa käytettyjen komponenttien vastuualueita protokollatasoittain eroteltuna 77
Kuva 10: Vuokaavio Xcityn henkilöpalvelun kutsumiseen liittyvistä komponenteista ja pisteet, joista liikennettä seurattiin 78

Taulukko 1: Suosituimmat ohjelmointikielet marraskuussa 2008 [TIOBE2008] ja niiden tuki XML-kielelle [useita lähdeviittauksia] 17

LYHENNELUETTELO

ABAP	Advanced Business Application Programming
ACM	Association for Computing Machinery
API	Application Programming Interface
ASMX	Active Server Methods
BL	Business Logic
CBDI	Component-Based Development and Integration
CL-XML	Common Lisp XML
COBOL	Common Business-Oriented Language
CXF	CeltiXFire
DB	Database
doc/enc	Document / Encoded
doc/lit	Document / Literal
E4X	ECMAScript for XML
EAR	Enterprse Archive
ECMA	European Computer Manufacturers Association
ESB	Enterprise Service Bus
html	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP over SSL/TLS
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IETF	The Internet Engineering Task Force
IOC	Inversion of Control
ISO	International Organization for Standardization
ISO/TR	ISO Technical Reference
IT	Information Technology
ITU	International Telecommunication Union
ITU-T	ITU – Telecommunication Standardization Section
J2EE	Java 2 platform, Enterprise Edition

JAX	Java API for XML
JBoss	Java-Based Open Source Server
JCA	J2EE Connector Architecture
JDK	Java Development Kit
JVM	Java Virtual machine
LAN	Local Area Network
LISP	List Processing
MIS	Management Information System
MSDN	Microsoft Developer Network
MVN	Maven
NT	New Technology
NTLM	Windows NT LAN Manager
OASIS	Organization for the Advancement of Structured Information Standards
OSI	Open Systems Interconnection
PHP	PHP: Hypertext Preprocessor
PL/SQL	Procedural Language / Structured Query Language
PSC	Propentus Service Center
RFC	Request For Comments
RMI	Remote Method Invocation
RPC	Remote Procedure Call
rpc/enc	RPC / Encoded
rpc/lit	RPC / Literal
SAS	Statistical Analysis Software
SAX	Simple API for XML
SHA	Secure Hash Algorithm
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol / Internet Protocol
TIOBE	The Importance Of Being Earnest
TLS	Transport Layer Security

TO	Transfer Object
UI	User Interface
UMO	Universal Messaging Object
URL	Uniform Resource Locator
VETUMA	Verkkotunnistaminen ja –maksaminen
VIP	Valtion IT-palvelukeskus
VM	Virtual Machine
VRK	Väestörekisterikeskus
VTJ	Väestötietojärjestelmä
W3C	World Wide Web Consortium
WCF	Windows Communication Foundation
WS	Web Services
WSDL	Web Services Description Language
WWW	World Wide Web
XML	Extensible Markup Language

1 JOHDANTO

Fenix-hanke (myöhemmin Lahti Fenix) on projekti, jonka tarkoituksena on kehittää erilaisia sähköisiä asiointipalveluja Lahden kaupungin ja muiden hankkeessa mukana olevien kuntien ja kaupunkien asukkaiden käyttöön. Hanke on jaettu kolmeen osaan, joita ovat lasten ja nuorten tukemiseen liittyvät palvelut, ajanvaraus- ja resurssienhallintajärjestelmien kehittäminen, sekä sähköisen asioinnin viitekehys. Ajanvaraus- ja resurssienhallintajärjestelmien kehittäminen liittyvät kokonaisuuteen, johon viitataan yleisesti Kuntalaistilijärjestelmä-nimityksellä. Nimensä mukaisesti kokonaisuuteen kuuluu ns. kuntalaistili, joka sisältää muiden järjestelmien yhteisesti tarvitsemat palvelut, jotka keskittyvät käyttäjien eli kuntalaisten tietojen hakemiseen ja käsittelyyn. Myöhemmin kokonaisuuteen voidaan liittää muitakin järjestelmiä.

Kuntalaistiliprojektin sisältämien järjestelmien arkkitehtuuri noudattaa palvelukeskeisen arkkitehtuurin yleisiä toimintamalleja. Palvelukeskeinen arkkitehtuuri on nykyisin yhä useammassa tietoteknisessä järjestelmässä käytetty tapa jakaa järjestelmän toiminnalliset osat itsenäisiin komponentteihin, niiden muodostaman palvelukokonaisuuden perusteella. Eri komponentteja ei liitetä yhdeksi suureksi monoliittiseksi systeemiksi, vaan ne pyritään pikemminkin erottamaan toisistaan mahdollisimman hyvin. Komponenttien välille voidaan tämän jälkeen tarvittaessa toteuttaa löyhä vuorovaikutus jollakin viestinvälitystekniikalla, esimerkiksi niin sanotuilla web-palveluilla. Tällainen modulaarinen malli mahdollistaa teoriassa varsin dynaamisen kehitysprosessin ja mahdollisuuksia lopputuotteen kustomointiin asiakkaan tarpeiden mukaan.

Järjestelmän komponenttien määrän kasvaessa niiden välisten sidosten ja siten ollen viestiliikenteenkin määrä kasvaa – usein jopa eksponentiaalisesti suhteessa komponenttien itsensä määrään. Mikäli keskenään vuorovaikuttavia komponentteja on useita, viestien välitystä varten palvelukeskeisiin järjestelmiin voidaan liittää tähän erikoistunut sovellus, palveluväylä. Keskeisenä osana

Kuntalaistilijärjestelmänkin arkkitehtuuria on palveluväylä, joka toimii reitityspisteenä järjestelmäkokonaisuuden eri osien tarjoamien palveluiden välillä. Palveluväylämallin käyttöönotolla pyritään myös suoraviivaistamaan integraatioiden toteutusta Kuntalaistilijärjestelmän ytimen ja siihen liitettävien oheisjärjestelmien välillä.

Kuntalaistilin teknisestä toteutuksesta vastaa kouvolalainen Propentus oy, joka on erikoistunut yritysten ja yhteisöjen dokumentaation ja käyttäjäoikeuksien hallintaan sekä palvelukeskeisten ratkaisujen tuottamiseen. Yrityksen toimintamalleihin kuuluu avoimen lähdekoodin tuotteiden suosiminen, mikä näkyy myös Lahti Fenix -hankkeen teknisissä ratkaisuissa. Kaikki käytetyt sovellusalustat ovat avoimeen lähdekoodiin perustuvia, palveluväylää myöten.

Tämän työn rajauksena on keskittyä tarkastelemaan tapoja hyödyntää web-palveluita ja avoimen lähdekoodin palveluväylää sisäisissä ja ulkoisissa järjestelmäintegraatioissa, pitäen viitekehyksenä Lahti Fenix -hankkeen Kuntalaistilijärjestelmää. Samalla tutustutaan Kuntalaistilijärjestelmän arkkitehtuuriin ja sen vaikutuksiin järjestelmän toiminnan kannalta. Ulkoisten integraatioiden suhteen työssä liitetään useita erityyppisiä protokollia hyödyntävän ulkoisen Xcity-resurssin tarjoamia palveluita osaksi Kuntalaistilijärjestelmää palveluväylän kautta. Lisäksi tutustutaan järjestelmän sisäisiin integraatioihin, ja mietitään tapoja saada niissä toteutettujen yhteyksien toiminta skaalautumaan suurelle kuormalle.

Tavoitteena on tutustua palveluväylän ominaisuuksiin ja erilaisten protokollien käyttöön ja saada samalla luotua tulevaisuudessakin käytettävä yleinen malli ulkoisten järjestelmien liittämiseen osaksi Kuntalaistilijärjestelmää. Työn aikana ei kuitenkaan aiota keskittyä kaikkien järjestelmän osien yksityiskohtaiseen toimintaan edes palveluväylätasolla, eikä etenkään käyttöliittymään ja tietokantoihin liittyvissä seikoissa. Tällaisia seikkoja ovat esimerkiksi www-sovelluskehykset, Ajax ja EJB3. Työssä ei myöskään aiota opettaa kaikkea valitun palveluväylän käytöstä. Mikäli tavoitteet saavutetaan, on niistä parhaassa

tapauksessa kertynyt tietoa ja taitoa, jota voidaan käyttää myös Lahti Fenix -projektin ulkopuolella.

Työssä kuvataan ensin Kuntalaistilijärjestelmän käyttötarkoitus. Sitten kerrataan yleistä integraatioihin liittyvää teoriaa käytännön palveluorientaatiopainotuksella, esimerkiksi web-palveluihin ja SOAP-kieleen liittyviä seikkoja. Lisäksi tutustutaan palveluväylän merkitykseen palvelukeskeisissä järjestelmissä. Tämän jälkeen tutustutaan Kuntalaistilijärjestelmään ja sen rajapintoihin hieman teknisemmin, pyrkien samalla paikantamaan integraatioiden kannalta merkittäviä pullonkauloja.

Sisäisten integraatioiden kohdalla käydään lyhyesti läpi Java-kielen kehittyneiden ominaisuuksien hyödyntämistapoja palvelukutsujen ohjelmoinnin suhteen. Lopuksi perehdytään yhteen ulkoiseen järjestelmään ja yritetään kehittää jokin tapa käyttää sen tarjoamia palveluita. Tämä edellyttää paitsi protokolliin tutustumista, myös sitkeää yrittämistä, ennen kuin mitään toimivaa saadaan aikaan. Tavoitteena on saada laadittua yleinen integraatiomalli, jolla toimivia yhteyksiä voidaan jatkossakin luoda, sekä saada käsitys tällaisen työn asettamista vaatimuksista mm. työssä käytettävien ohjelmistoversioiden suhteen.

2 KUNTALAISTILIJÄRJESTELMÄN YLEISKUVAUS

Kuntalaistilijärjestelmä on kehitysvaiheessa oleva järjestelmäkokonaisuus, jonka on tarkoitus toimia keskitettynä alustana erilaisten palveluiden tarjoamiseen ja yhteen liittämiseen. Järjestelmä on osa tällä hetkellä kehitysvaiheessa olevaa Lahden kaupungin Lahti Fenix -hanketta, ja sen juuret ovat vuosina 2006 - 2008 läpiviedyssä Kuntien sähköinen palvelualusta -hankkeessa. Kyseisissä hankkeissa Kuntalaistilijärjestelmän avulla toteutetaan muun muassa kuntien julkisten tilojen varauspalvelu ja kansalaisten joukkoseulonnan ajanvarauspalvelu terveydenhuoltoa varten. Myös muunlaisia sosiaalisia palveluita, kuten koulupalveluita ja päivähoitopalveluita, voidaan myöhemmin ottaa osaksi järjestelmää.

Koska Kuntalaistilijärjestelmä toteutetaan palvelukeskeisen arkkitehtuurin periaattein, sitä on myöhemmin mahdollista laajentaa melko helposti, liittämällä uusia sisältömoduuleja osaksi Kuntalaistilijärjestelmää. Tämänhetkisellä kehitysasteella Kuntalaistilijärjestelmä koostuu kolmesta suuresta sisältömoduulista, joita ovat koko järjestelmän ytimenä toimiva kuntalaistili, sekä siihen liitettävät kunnalliset tilanvarausjärjestelmä ja ajanvarausjärjestelmä. Palveluiden tarkoitettuja käyttäjiä ovat kaikki projekteissa mukana olevien kuntien asukkaat.

2.1 Kuntalaistili

Kuntalaistili on yleisluontoinen moduuli, jonka ympärille voidaan rakentaa erillisiä kunnan palveluihin liittyviä sisältömoduuleja, mutta joka toimii siihen liitettävien moduulien suhteen riippumattomasti. Kuntalaistili sisältää yleisesti tarvittavia palveluita ja muiden moduulien palveluiden käyttöön tarvittavia taustatietoja. Yleisesti tarvittavia palveluita ovat esimerkiksi järjestelmään kirjautuminen, käyttäjätietojen hallinta ja sähköiset maksupalvelut, sekä muiden palveluiden käytön kannalta läpinäkyvät taustapalvelut, kuten session hallinta, tietokantamuutosten seuranta ja palvelustatistiikan ylläpito. Taustatietoja ovat esimerkiksi järjestelmässä tapahtuneiden muutosten jäljittämiseen tarvittavat kirjausketjut, käyttäjien henkilötiedot ja yleishyödylliset tiedot, esimerkiksi

kuntiin liittyvät tiedot. Tiedon luonteesta riippuen se voidaan pyytää käyttäjältä, saada taustajärjestelmistä tai luoda automaattisesti järjestelmän sisäisen logiikan toimesta. Taustajärjestelmistä saatavaa tietoa ovat esimerkiksi käyttäjän viralliset henkilötiedot, jotka voidaan hakea Väestörekisterikeskuksen (VRK) hallinnoimasta Väestötietojärjestelmästä (VTJ) ja kuntien hallinnoimista Tekla Xcity -järjestelmistä. Käyttäjältä saatavaa tietoa voi olla esimerkiksi täydentävä, epävirallinen tieto, kuten käyttäjän kutsumanimi. Automaattisesti generoitavaa tietoa on esimerkiksi käyttäjän kutsumanimen viimeisin muokkausajankohta.

2.2 Tilanvarausjärjestelmä

Tilanvarausjärjestelmä on kunnan, kaupungin tai vastaavan julkishallinnollisen elimen käyttöön suunniteltu, kuntalaistiliin liitettävä sisältömoduuli. Sen tarkoituksena on julkisen tahon (kunnan) hallinnoimien tilojen hallinnan helpottaminen ja tehostaminen keskittämällä niihin liittyvät tiedot yhteen järjestelmään. Järjestelmän ylläpitäjät voivat lisätä järjestelmään uusia tiloja ja määritellä niihin liittyviä tietoja ja muita ominaisuuksia, kuten varausaika- tai käyttäjäryhmärajoituksia. Jos järjestelmään lisätään vaikkapa kunnan ylläpitämä uimahalli, voidaan se jakaa erikseen varattaviin osiin esimerkiksi uima-altaiden mukaan. Tilan tai tilan osien voidaan määritellä olevan varattavissa melko monipuolisesti määriteltävin ehdoin, esimerkiksi viikon varoitusajalla, pelkästään maanantaisin kello 10 ja 13 välillä, ja vain mikäli varaaja kuuluu paikalliseen uimaseuraan. Samoin varausten minimiajat, maksujärjestelyt ja vastaavat seikat ovat määriteltävissä.

Järjestelmän kautta asiakaskunnan (kuntalaisten) on mahdollista katsella ja varata heidän käytettävikseen tarjolla olevia, julkisesti hallinnoituja tiloja. Järjestelmä ottaa sekä ennalta määritetyt ehdot että olemassa olevat varaukset huomioon esittäessään vapaita aikoja tilan varaajalle, ja määriteltyjen tietojen perusteella tarjoaa myös tilaan mahdollisesti liittyviä lisäpalveluita, uimahallin tapauksessa esimerkiksi kellukkeita tai uppopalloa. Kaikki tyypillinen tilanvarauksiin liittyvä asiointi, kuten maksut ja peruutukset hoidetaan järjestelmän kautta ilman ihmisen väliintuloa prosessointiketjussa, järjestelmän kutsuessa taustapalveluita automaattisesti niin tarvittaessa (mm. verkkopankkisovellukset).

2.3 Ajanvarausjärjestelmä

Ajanvarausjärjestelmä on pääsääntöisesti henkilöresurssien varaamiseen suunniteltu, tällä hetkellä kehitteillä oleva kunnallinen sisältömoduuli Kuntalaistilijärjestelmään. Se on kuitenkin alusta lähtien pyritty suunnittelemaan melko yleiskäyttöiseksi, joten muunlaistenkin resurssien hallinnointi ja varaaminen voi jossakin määrin olla mahdollista sen kautta. Tilojen varaamisen erityistarpeisiin ajanvarausjärjestelmä ei kuitenkaan sovellu erityisen hyvin. Järjestelmän perusidea vastaa tilanvarausjärjestelmää: siihen voidaan julkishallinnon puolelta, pääkäyttäjien toimesta lisätä erilaisia resurssityyppejä ja niitä vastaavia resursseja, joita kuntalaiset voivat tämän jälkeen selata.

Tyypillisiä ajanvarausjärjestelmän kautta varattavia resursseja voivat olla esimerkiksi terveydenhoitoon liittyvät palvelut. Kuntalainen voi järjestelmän kautta esimerkiksi varata ajan hammaslääkärille, ja järjestelmä allokoi ajan jollekulle hammaslääkärille, jolla kyseisellä hetkellä on kalenterissa vapaata aikaa. Järjestelmän kautta myös tietyissä tehtävissä työskentelevien henkilöiden on mahdollista varata heidän työssään tarvitsemia laitteistoresursseja. Resursseja voidaan ryhmitellä myös erilaisiksi kombinaatioiksi, joita voidaan tämän jälkeen käsitellä samaan tapaan kuin yksittäisiäkin resursseja, jolloin useaan kohteeseen voidaan kohdistaa varauksia yhdellä varauspyynnöllä.

3 JÄRJESTELMÄINTEGRAATIOT PALVELUKESKEISESSÄ ARKKITEHTUURISSA

Palvelukeskeinen arkkitehtuuri eli SOA (Service Oriented Architecture) on viime vuosina paljon huomiota osakseen saanut tapa suunnitella tietojärjestelmän looginen arkkitehtuuri, mikä ilmenee esimerkiksi Googlen tai muiden Internet-hakukoneiden tuloslistausten perusteella. SOA-periaatteisiin kuuluu järjestelmän komponenttien mieltäminen joukkona palveluita, joista kokonaistoiminnallisuus muodostuu, ja näiden palveluiden mahdollisimman suuri toisistaan riippumattomuus [Erl2005]. Kiinteän integraation sijaan suositaan löyhää vuorovaikutteisuutta (interoperability) järjestelmän osien välillä. Myös Lahti Fenix -hankkeen Kuntalaistilijärjestelmä perustuu palvelukeskeisen arkkitehtuurin ja web-palveluiden (WS, web services) käyttöön, mikä mahdollistaa useiden erilaisten palveluiden liittämisen osaksi järjestelmää kulloisenkin tarpeen mukaisesti.

3.1 Järjestelmien integrointi eri tasoilla

Palvelukeskeisessä arkkitehtuurissa järjestelmän komponentit suunnitellaan mahdollisimman autonomisiksi ja itsenäisiksi muista komponenteista. Ideaalisesti, järjestelmän komponenttien ollessa toisistaan erillään, ei niiden ole välttämätöntä olla sidoksissa myöskään keskenään samoihin toteutusteknisiin ratkaisuihin, kuten ohjelmointikieleen, tai edes fyysiseen sijaintiin. Täten SOA-järjestelmä kokonaisuutena voi olla piileviltä teknisiltä ratkaisuiltaan hyvinkin heterogeeninen, joskaan tyypillisissä tilanteissa tällainen ei varmasti ole kovinkaan tarkoituksenmukaista. On selvää, että teknologiariippumattomuus nousee kuitenkin tärkeäksi eduksi rakennettaessa vuorovaikutussuhteita eri tahojen toteuttamien järjestelmien välille, jolloin on äärimmäisen epätodennäköistä, että kaikki tekniset ratkaisut osuvat järjestelmien välillä kohdalleen.

Kun SOA- tai muita järjestelmiä integroidaan, niiden välinen integraatio voidaan toteuttaa useilla eri teknologiapinon kerroksilla, joita järjestelmiin kuuluu.

Esimerkiksi kaksi eri järjestelmää voi käyttää samaa tietokantaa olematta muuten yhteydessä toisiinsa. Tällöin on kyse datakerroksen integraatiosta [Newcomer2005]. Muita mahdollisia tasoja ovat esimerkiksi sovelluskerroksen integraatio ja käyttöliittymäkerroksen integraatio. Kaikissa on omat ongelmansa: Datakerroksella ei ole aina itsestään selvää, miten mikäkin tietueen sarake tulisi tulkita, eivätkä eri järjestelmien tietokanta-arkkitehtuurit muutenkaan välttämättä ole helposti yhteen sovitettavissa. Sovelluskerroksella integraatio edellyttää yhteensopivuutta mm. ohjelmointirajapintojen välillä, mikä ei ole itsestään selvä ominaisuus, eri kielten ollessa kyseessä. Käyttöliittymätasolla voidaan integraatio toteuttaa esimerkiksi ns. ruudun raapimisella (screen scraping), mikä on useasti työlästä ja alttiina hajoamaan, mikäli jonkun integroitavan sovelluksen käyttöliittymän asettelu vähänkin muuttuu. Millään teknologiapinon olemassa olevalla tasolla integraatio ei ole täysin suoraviivaista.

Eric Newcomerin [Newcomer2005] mukaan paras tapa toteuttaa järjestelmäintegraatio on esittelemällä uusi abstraktiokerros pelkästään kyseistä tarkoitusta varten. Kunnollisessa SOA-mallin mukaisessa eli riippumattomuutta korostavassa integraatiossa tämä voikin olla varsin varteenotettava vaihtoehto, sillä muilla kerroksilla keskinäisriippuvuudet nousevat nopeasti hyvin suuriksi. Erillisen integraatiokerroksen kanssa tällaista ongelmaa ei pitäisi esiintyä, sillä vaikka mikään muu seikka ei estäkään sellaisen toteuttamista tekniikkariippuvaisella tavalla, on integraatiokerros tietenkin tarkoituksenmukainen ainoastaan, mikäli sen rajapinta tarjoaa protokollan, joka on riittävissä määrin universaali, ja siten ollen toteutustekniikkariippumaton.

3.2 XML-pohjaisten merkkauskielten hyödyntäminen järjestelmäintegraatioissa

Olivatpa järjestelmät tarkoituksena liittää toisiinsa millä teknologiapinon tasolla tahansa, eri päätepisteiden tulisi ymmärtää samaa tietoliikenneprotokollaa. Määritelmänomaisesti SOA suunnitteluparadigmana ei ota kantaa toteutustekniikoihin. Tästä johtuen tietynasteiseksi ongelmaksi kuitenkin muodostuu, ettei ole olemassa yhtä ja ainoaa protokollaa, jolla jonkin järjestelmän tarjoamien palveluiden voisi automaattisesti olettaa toimivan. Eräs toimiva

ratkaisu ongelmaan on XML-pohjainen (Extensible Markup Language) integraatiokerros [Newcomer2005], joka rakennetaan integroitaviin järjestelmiin. On nähtävissä, että tällainen ratkaisumalli perustuu oletukselle siitä, että XML merkkauskielenä on tuettu useissa historiallisesti merkittävissä ja nykyisin laajalti käytetyissä ohjelmointikielissä joko suoraan tai lisäkirjastojen kautta.

XML-tuen yleisyyden selvittämistä varten tulee saada käsitys yleisimmistä ohjelmointikielistä ja niiden tarjoamasta tuesta. Kielten yleisyyssuhteiden täysin luotettava selvittäminen kautta historian lienee kuitenkin mahdotonta. Esimerkiksi tietyllä kielellä kirjoitettujen lähdekoodirivien määrän vertailu toisen kielen rivimääriin ei ole sinänsäkään luotettava mittari, kielten notaatiotapojen vaihdellessa. Useista käytännön syistä johtuen tilastotieteilijät eivät välttämättä voi muutenkaan päästä tutkimaan minkä tahansa ohjelman lähdekoodeja. Näistä ja muista seikoista johtuen analyysejä joudutaan tekemään joihinkin muihin indikaattoreihin, kuten tietyistä kielistä kertovan kirjallisuuden määrään, perustuen, vaikka niidenkin antamaan dataan tulee ilmiselvästi suhtautua sitä viitteellisenä pitäen. TIOBE Software [TIOBE2008] on laatinut erilaisiin indikaattoreihin perustuvia arvioita eri ohjelmointikielten suhteellisista suosioista vuodesta 2001 lähtien. Ote TIOBEn luettelon perusteella suosituimmista ohjelmointikielistä marraskuussa 2008 löytyy taulukosta Taulukko 1.

Sij.	Ohjelmointikieli	Suosio %	XML-tuki
1	Java	20,299	standardikirjastot (JAX) [Sun2006]
2	C	15,276	lisäkirjastot (Expat) [Expat2007]
3	C++	10,357	lisäkirjastot (Xerces) [Apache2007]
4	(Visual) Basic	9,270	.NET-standardikirjastot (System.XML) [Microsoft2003]
5	PHP	8,940	standardikirjastot (SimpleXML) [PHPGroup2008]
6	Python	5,140	lisäkirjastot (PyXML) [PyXML2008]
7	C#	4,026	.NET-standardikirjastot (System.XML) [Microsoft2003]
8	Delphi	4,006	lisäkirjastot (OmniXML) [Remec2008]
9	Perl	3,876	lisäkirjastot (Perl XML) [McLean2008]
10	JavaScript	2,925	standardikirjastot (E4X) [ECMA2005]
11	Ruby	2,870	lisäkirjastot (XmlSimple) [Schmidt]
12	D	1,442	lisäkirjastot (SimpleXMLD) [DSource2008]
13	PL/SQL	0,939	lisäkirjastot (XML parser for PL/SQL) [Oracle2008]
14	SAS	0,729	standardikirjastot (XML Engine) [SAS2008]
15	ABAP	0,570	standardikirjastot [Keller2004]
16	Pascal	0,511	lisäkirjastot (SAX for Pascal) [Pascal2003]
17	COBOL	0,510	standardikirjastot [TR247162007]
18	ActionScript	0,506	standardikirjastot (E4X) [ECMA2005]
19	Logo	0,489	ei tukea
20	Lua	0,473	lisäkirjastot (LuaExpat) [LuaExpat2007]

Taulukko 1: Suosituimmat ohjelmointikielet marraskuussa 2008 [TIOBE2008] ja niiden tuki XML-kielelle [useita lähdeviittauksia]

Kuten taulukosta 1 voidaan nähdä, 20 suosituinta ohjelmointikieltä kattaa arviolta yli 93 prosenttia kaikkien kielten yhteenlasketusta suosiosta. Taulukkoon on lisäksi kerätty tietoa kunkin mainitun kielen XML-tuesta (lähdeviittaukset taulukossa). Pitäen lukuja suuntaa antavina, voidaan arvioida, että ainakin yli 92 prosentissa kaikista maailman ohjelmista on mahdollisuus joko käyttää XML-kieltä suoraan tai ottaa se käyttöön ilman tarvetta lähteä kehittämään omia tukikirjastoja. Osa kielistä, joita ei taulukossa mainita, sisältää myös XML-tuen, joten todellisuudessa luku lienee vielä suurempi. Historiallisina esimerkkeinä tällaisista kielistä mainittakoon Lisp (CL-XML) [Anderson2003] ja Fortran (Fortran-XML) [Markus2008], joissa molemmissa on XML-tuki. Koska XML näiden lukujen valossa selvästi on universaalisti ymmärretty merkkauskieli

vähintäänkin suosituimpien ohjelmointikielten kontekstissa, on se hyvä perusta toteutustekniikkariippumattomaan viestiliikenteeseen SOA-arkkitehtuureissa.

3.3 Web-palvelut

Vaikka SOA-mallin mukaisia järjestelmiä voidaan toteuttaa periaatteessa millä viestiliikenneprotokollalla tahansa [Erl2005], yleiseksi tavaksi on noussut toteutus niin sanottuja web-palveluita käyttäen. Mikä tahansa järjestelmä, joka kykenee vastaanottamaan ja tarjoamaan XML-muotoista dataa HTTP:n (Hypertext Transfer Protocol) yli, on web-palvelukykyinen, sillä yksinkertaisimmillaan web-palvelut ovat pelkästään nämä määritelmät täyttäviä rajapintoja [Newcomer2005]. Koska XML-kieleen perustuvia kuvauskieliä voidaan laatia rajattomasti uusia, näin väljästi määriteltynä web-palvelut voidaan toteuttaa äärettömän monella yhteensopimattomalla tavalla.

Onneksi World Wide Web Consortium (W3C) rajaa web-palvelut huomattavasti tiukemmin [W3C2004] kuin yllä on määritelty: Rajapinnat ja niiden käyttötapa kuvataan erityisellä tähän tarkoitukseen kehitetyllä XML-pohjaisella WSDL-kielellä (Web Services Description Language). Liikennöinti rajapintojen välillä tapahtuu SOAP-kielellä (alun perin Simple Object Access Protocol, nykyisin vailla aukikirjoitettua muotoa [Møller2006]), jota käytetään olioiden serialisointiin XML:ksi. (Mielenkiintoisena seikkana mainittakoon, että W3C:n määritelmä ei edellytä HTTP:n käyttöä.) W3C:n määritelmän ansiosta web-palveluiden toteuttamiseen on olemassa standardinmukainen tapa jo olemassa olevin XML-tekniikoin, mikä siten ollen teoriassa mahdollistaa automaattisen syntaktisen yhteensopivuuden eri järjestelmien välillä, vaikka järjestelmät edustaisivat keskenään muuten täysin erilaisia toteutustekniikoita. SOA/WS-ympäristö on siis kuin alusta lähtien suunniteltu integraatioiden suoraviivaistamiseksi.

3.4 SOAP-viestien eri variaatiot

Vaikka web-palvelut teoriassa yksinkertaistavat järjestelmien yhteenliittämistä radikaalisti, käytännössä ei niidenkään keskinäinen protokollayhteensopivuus ole kirkossa kuulutettu, sillä valitettavasti protokollassa käytetty SOAP voidaan

ymmärtää useammalla kuin yhdellä eri tavalla [Erl2005]. WSDL-kuvauksilla ja SOAP-viesteillä on kaksi attribuuttia, `style` ja `use`, jotka molemmat vaikuttavat siihen, millä tavalla SOAP-viesti muodostetaan. Kumpikin attribuutti voi saada arvonsa kahdesta eri vaihtoehdosta (`style`: `document` tai `RPC` (Remote procedure Call), `use`: `literal` tai `encoded`).

`Style`- ja `use`-attribuuttien avulla voidaan muodostaa kolme keskenään erilaista SOAP-variaatiota: Document/Literal (doc/lit), RPC/Literal (rpc/lit) (Remote Procedure Call) ja RPC/Encoded (rpc/enc). Näiden lisäksi on olemassa vielä neljäs variaatio, Document/Literal wrapped [IBM2003], joka on yleisesti käytössä Apache Axis -alustalla toimivissa web-palveluissa. Document/Encoded olisi vielä viides variaatio, mutta se ei ole [IBM2003] mukaan käytössä missään. Eri variaatiot eivät toimi keskenään yhteen, joten SOAP-viestit tulee muodostaa web-palvelun WSDL-kuvauksen käyttämän variaation mukaan. Eri variaatioiden mukaiset SOAP-viestit on kuvattu listauksissa Listaus 1, Listaus 2, Listaus 3 ja Listaus 4 [IBM2003]. Viestit toteuttavat keskenään saman operaatiokutsun, eli kutsuvat web-metodia `myMethod` parametreilla `int x` ja `float y`.

```
1  <soap:envelope>
2      <soap:body>
3          <xElement>5</xElement>
4          <yElement>5.0</yElement>
5      </soap:body>
6  </soap:envelope>
```

Listaus 1: SOAP doc/lit-enkoodattuna [IBM2003]

```
1  <soap:envelope>
2      <soap:body>
3          <myMethod>
4              <x>5</x>
5              <y>5.0</y>
6          </myMethod>
7      </soap:body>
8  </soap:envelope>
```

Listaus 2: SOAP doc/lit wrapped -enkoodattuna [IBM2003]

```
1  <soap:envelope>
2      <soap:body>
3          <myMethod>
4              <x>5</x>
5              <y>5.0</y>
6          </myMethod>
7      </soap:body>
8  </soap:envelope>
```

Listaus 3: SOAP rpc/lit-enkoodattuna [IBM2003]

```
1  <soap:envelope>
2      <soap:body>
3          <myMethod>
4              <x xsi:type="xsd:int">5</x>
5              <y xsi:type="xsd:float">5.0</y>
6          </myMethod>
7      </soap:body>
8  </soap:envelope>
```

Listaus 4: SOAP rpc/enc-enkoodattuna [IBM2003]

3.4.1 SOAP doc/lit-enkoodattuna

Kuten listauksesta Listaus 1: SOAP doc/lit-enkoodattuna [IBM2003] voidaan havaita, on se muihin verrattuna lyhyempi, mikä on tietysti hyvä asia overheadin kannalta, mutta toisaalta lyhyys perustuu siihen, että siitä puuttuu kutsutun web-metodin nimi, `myMethod`. Näin ollen kutsuttu metodi pitää vastaanottajapäässä päätellä parametrien määrän mukaan, mikä on mahdotonta, mikäli vastaanottajapäässä on useampi kuin yksi metodi, joka vastaanottaa kaksi parametria.

3.4.2 SOAP rpc/enc-enkoodattuna

Toisin kuin doc/lit-enkoodatut SOAP-viestit, rpc/enc-enkoodatut viestit (Listaus 4: SOAP rpc/enc-enkoodattuna [IBM2003]) sisältävät kutsutun metodin nimen, ja lisäksi parametrien tyypit, joten ne ovat kaikista viesteistä täsmällisimpiä. Koska kutsutun metodin nimi kuitenkin jo tiedetään, parametrien tyypit ovat periaatteessa turhia. Niiden aiheuttama haitta on viestin overheadin kasvaminen muita suuremmaksi.

3.4.3 SOAP rpc/lit- tai doc/lit wrapped -enkoodattuna

Listaukset Listaus 2: SOAP doc/lit wrapped -enkoodattuna [IBM2003] ja Listaus 3: SOAP rpc/lit-enkoodattuna [IBM2003] näyttävät syntaktisesti keskenään täsmälleen samanlaisilta viesteiltä. Niiden ero liittyy elementin `<myMethod>` semanttiseen merkitykseen: Rpc/lit-enkoodatussa SOAP-viestissä elementti kuvastaa operaation nimeä, kun taas doc/lit wrapped -enkoodatussa SOAP-viestissä elementin nimi tarkoittaa hyötykuormana lähetyn olion nimeä. Doc/lit wrapped -viesti toisin sanoen ottaa sisäänsä vain yhden XML-skeemoissa `xsd:complexType`-tyyppiseksi määriteltävän parametrin, joka on serialisoitu versio oliosta `myMethod`, jolla on kaksi ominaisuutta, `int x` ja `float y`.

Jos semantiikkaa ei huomioida, sekä rpc/lit että doc/lit wrapped sisältää metodin nimen, mutta ei parametrien tyyppejä, joten ne ovat tietynlaisia doc/lit- ja rpc/enc-viestin hybridejä. Rakenteen vuoksi niiden avulla on mahdollista tunnistaa kutsuttu metodi joka tilanteessa, mutta viestien overhead pysyy silti matalampana kuin rpc/enc-viesteissä. Koska semantiikallakin kuitenkin on toisinaan merkitystä, todettiin doc/lit wrapped -enkoodauksen olevan rpc/lit-enkoodausta suositeltavampi tapa niissä tilanteissa, joissa valinta on mahdollinen, sillä doc/lit wrapped vastaa paremmin SOA-paradigman ideaa, jossa (SOAP-) dokumentteja ajatellaan operaatioiden lopputuloksina eikä välikappaleina.

3.5 Integraatiot olemassa olevia rajapintoja käyttäen

Vaikka web-palvelutyyppisiä rajapintoja voitaisiin teoriassa toteuttaa suureen osaan olemassa olevia järjestelmiä, ei käytännön syistä johtuen moinen ole aina mahdollista. Vanhojen ohjelmien lähdekoodi voi olla hukassa, tai lähdekoodi voi olla niin epäselvää, että luotettavien WS-rajapintojen kirjoittaminen on vaikeaa. Mahdollisesti ohjelmaa tai sen kääntäjää ajetaan alustalla, jolla täytyy käyttää vanhaa versiota ohjelmointikielestä, jolloin uudemmalle versiolle kirjoitettuja XML-kirjastoja ei voidakaan käyttää. Tärkeimpinä syinä ovat tietenkin vanhojen ohjelmien muokkaukseen kuluvat resurssit, aika ja erityisesti raha. Koska vanhojen järjestelmien uudelleenohjelmointi on vaikeaa, hidasta ja kallista, ei sitä haluta tehdä, ellei ole pakko. Käytännössä vanhojen järjestelmien integrointi osaksi muita kokonaisuuksia täytyy siis useimmiten tehdä vanhojen järjestelmien

asettamilla teknisillä ehdoilla eikä toisinpäin. Näissä tilanteissa eri protokollia hallitseva, erityisesti viestiliikenteen reitittämistä varten suunniteltu palveluväylä voi osoittautua hyödylliseksi.

4 KUNTALAISTILIJÄRJESTELMÄN TEKNINEN JÄRJESTELMÄ- JA ARKKITEHTUURIKUVAUS

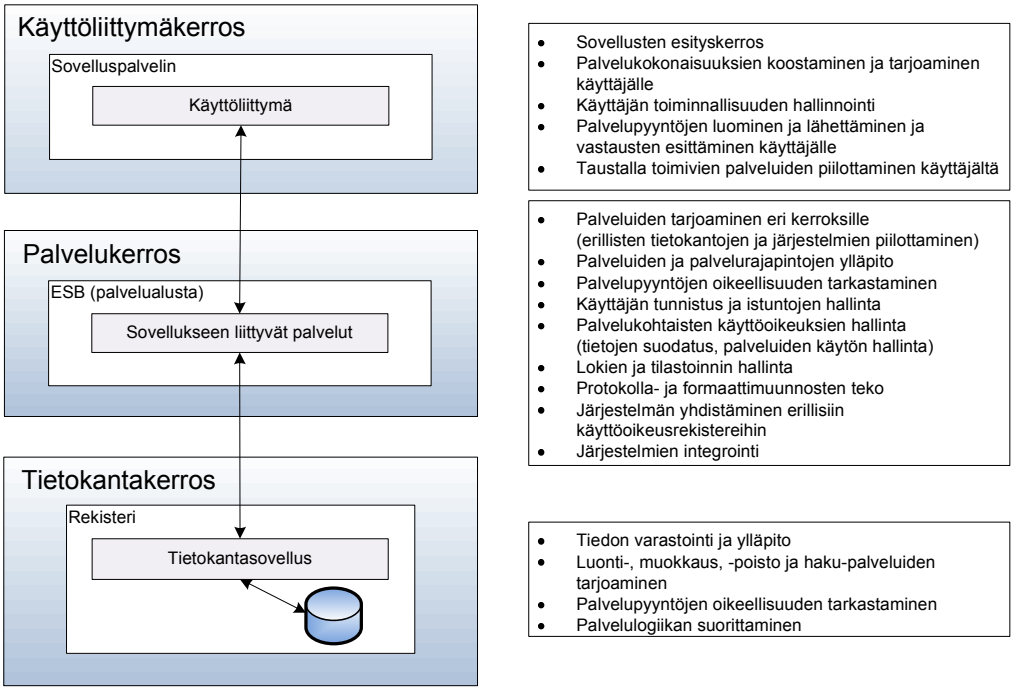
Kuntalaistilijärjestelmä on rakennettu modulaarisesti kokonaisuudeksi, jossa eri osakokonaisuuksien väliset keskinäisriippuvuudet on pyritty pitämään mahdollisimman vähäisinä, ja jossa kukin osakokonaisuus keskittyy omaan loogisesti muiden osien toiminnallisuudesta erilliseen osa-alueeseensa. Korkealla abstraktiotasolla järjestelmä on jaoteltavissa horisontaalisesti sisältömoduuleiksi ja vertikaalisesti tietokanta- palveluväylä- ja käyttöliittymäkerroksiksi. Ajattelutapa heijastuu arkkitehtuurin lisäksi myös järjestelmän käyttöliittymään, jossa kunkin sisältömoduulin palvelut on ryhmitelty moduuleja vastaavien pääotsakkeiden alaisiksi kokonaisuuksiksi. Lisäksi kunkin sisältömoduulin tai vastaavan osakokonaisuuden sisäiset komponenttitason ratkaisut noudattavat riippuvuuksia minimoivaa ja vuorovaikutteisuutta korostavaa mallia tilanteissa, joissa sellainen on katsottu työmäärien ja saavutetun edun kannalta järkeväksi.

Kuntalaistilijärjestelmän toteutusta voidaan sen arkkitehtuurin vuoksi kutsua ideaalisen SOA-mallin pragmaattiseksi sovellukseksi. Ohjelmointikieleksi järjestelmän toteuttamista varten valittiin Java, joka on tulkattava ohjelmointikieli. Tulkattavat ohjelmointikielet ovat tunnetusti riippumattomia käyttöjärjestelmän tai tietokonearkkitehtuurin erityispiirteistä. Koska Sun Microsystemsin Java-virtuaalikone (JVM, Java Virtual Machine) on asennettavissa kaikkiin merkittäviin käyttöjärjestelmiin (Windows, Unix, Linux, Mac OS, Solaris), voidaan Kuntalaistilijärjestelmä asentaa lähes mille palvelimelle tahansa, mikä voi antaa taloudellisuusetuja tai muita synergiaetuja järjestelmän tilaajille. Järjestelmän toteuttava Propentus oy on erikoistunut Java-pohjaisten sovellusratkaisujen tuottamiseen.

4.1 Looginen kerrosarkkitehtuuri

Kuntalaistilijärjestelmän kerrosarkkitehtuuri koostuu kolmesta järjestelmän eri osa-alueita hoitavasta kerroksesta (Kuva 1). Alimpana kerroksena on tietokantakerros. Kerros keskittyy vastaanotettujen palvelupyyntöjen perusteella

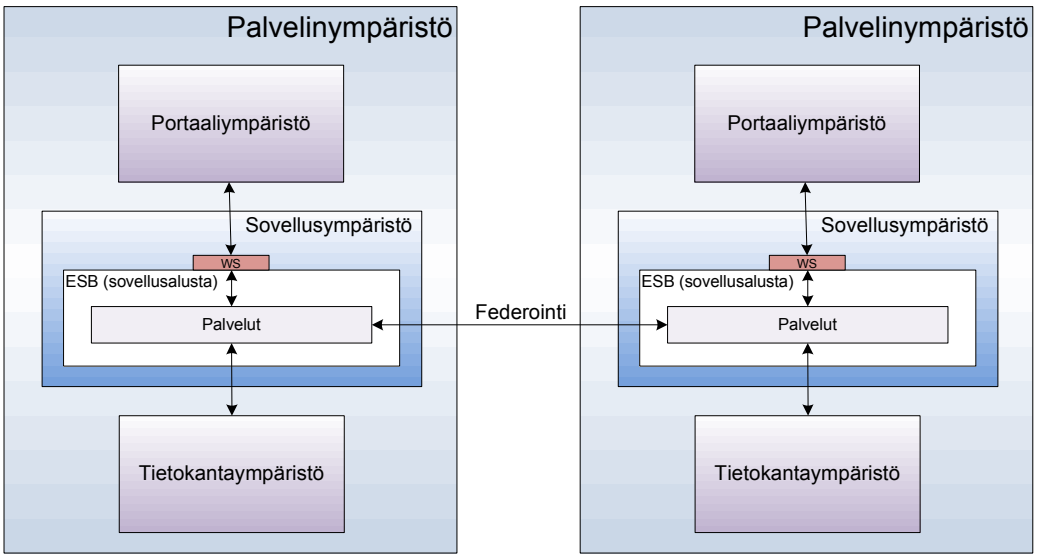
tapahtuviin tietokantaoperaatioihin, eli tiedon hakuun, tallennukseen ja poistamiseen. Tietokantakerros koostuu useammasta tietokannasta, ja jokainen tietokanta voidaan niin haluttaessa jakaa omalle fyysiselle palvelimelleen. Lisäksi kerros suorittaa tietokantoihin kohdistuvia eräajoja, joilla ylläpidetään aikariippuvaista tietoa, esimerkiksi poistetaan menneitä varauksia. Palvelupyynnöt tietokantakerrokselle tulevat poikkeuksetta palveluväyläkerrokselta, joka on järjestelmän keskimmäinen taso. Yhteydenotot muista sijainneista on estetty.



Kuva 1: Kuntalaistilijärjestelmän looginen kerrosarkkitehtuuri ja kerrosten vastuualueet [Propentus2008]

Palveluväyläkerros sisältää suurimman osan järjestelmän bisneslogiikasta. Nimensä mukaisesti kerroksella toimii palveluväylä eli ESB, joka vastaanottaa ja reitittää palvelupyyntöjä asianmukaisille komponenteille. ESB ylläpitää käyttäjäkohtaisia istuntoja, joiden avulla mm. selvitetään eri palveluiden käyttöoikeudet. Tallennus- poisto- ja vastaavissa palveluissa ESB huolehtii myös kirjausketjun muodostamisesta, ja palveluiden käyttöasteiden tilastoinnista kaikkien palveluiden tapauksessa. Myös palveluväylä on mahdollista pilkkoa monelle eri palvelimelle niin sanotun federoinnin avulla. Federoinnissa useampi palveluväylä liitetään loogisesti toisiinsa kuvan Kuva 2 mukaisesti. Federoinnin

avulla voidaan toteuttaa palvelukerroksen integraatioita yli toimialuerajojen. Palvelupyyntöjä kerros voi vastaanottaa mistä tahansa sijainnista, mutta suurin osa pyynnöistä tulee käyttöliittymäkerrokselta.



Kuva 2: Järjestelmien välinen federointi palvelualustaa käyttäen [Propentus2008]

Ylimpänä kerroksena Kuntalaistilijärjestelmän kerrosarkkitehtuurissa toimii käyttöliittymäkerros. Kerros huolehtii www-sivujen välittämisestä käyttäjien selaimille. Sivujen muodostaminen tapahtuu ajonaikaisesti jokaisen sivun kohdalla, sillä järjestelmän sisältämä, käyttöliittymällä esitettävä data on järjestelmän käyttötarkoituksen vuoksi pääosin dynaamista. Käyttöliittymäkerros huolehtii palvelupyyntöjen välittämisestä palveluväyläkerrokselle ja näkymättömän metadatan liittämisestä pyyntöihin. Käyttöliittymäkerroksenkin voi eriyttää omalle, tietokanta- tai palveluväyläkerroksesta erilliselle palvelimelleen.

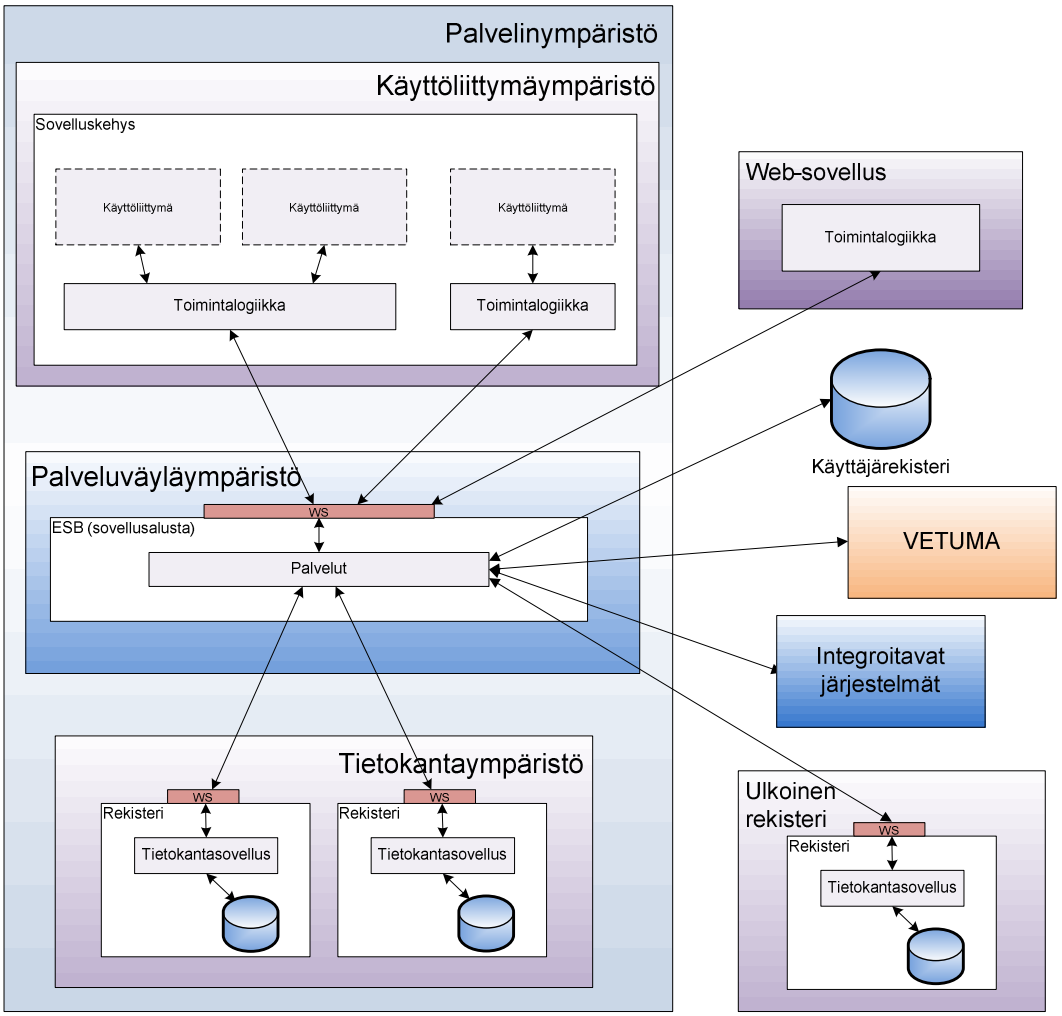
4.2 Web-palveluiden granulariteetti Kuntalaistilijärjestelmässä

Web-palveluita suunniteltaessa eräs huomionarvoisimmista suunnittelijan päätettäväksi jäävistä seikoista on palvelun granulariteetti, millä tarkoitetaan tässä yhteydessä sitä, mitä kaikkia matalan tason toimenpiteitä yksi palvelukutsu aiheuttaa. Tarjottavien palveluiden sopivalle karkeudelle ei ole olemassa yksiselitteistä kaavaa [Erl2005]. Kuntalaistilijärjestelmän kehityksessä lähestymistapana on yleisesti ottaen ollut, että julkisista rajapinnoista tarjottavat

palveluoperaatiot vastaavat granulariteetiltaan yksittäistä liiketoimintaprosessia. Tällöin palveluoperaation raekoosta tulee tietenkin melko karkea. Ajattelumallia voidaan kuitenkin pitää sikäli loogisena, että yhden uuden palvelun lisääminen järjestelmään tarjoaa palvelun kutsujalle keskimäärin yhden uuden atomisena pidettävän toiminnon. Näin toiminnallinen ja lähdekoodillinen koheesio saadaan pysymään korkeana, mikä helpottaa komponenttien testausta ja muutosten tekemistä prosesseihin. Etuna on myös viestiliikenteen määrän vähäisyys suhteessa hienommiksi jaettuihin palveluihin. Aina tällainen lähestymistapa ei kuitenkaan ole mahdollinen, eivätkä yksittäisen liiketoimintaprosessin rajatkaan välttämättä ole itsestään selviä. Totta kai myös palveluiden uudelleenkäytettävyys muissa kuin alkuperäisessä kontekstissa kärsii granulariteetin ollessa karkea. Tämän vuoksi rajoitusta ei järjestelmän yksityisissä rajapinnoissa ole joka tilanteessa katsottu aiheelliseksi noudattaa.

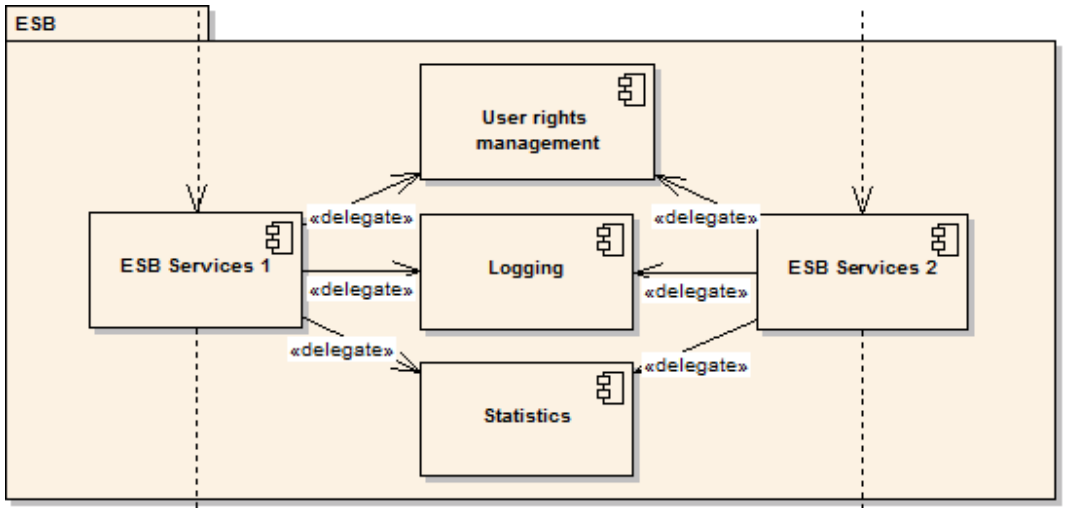
4.3 Järjestelmän palveluväyläarkkitehtuuri ja liitosrajapinnat

Kuntalaistilijärjestelmän sisältämä bisneslogiikka sisältyy palveluihin, joita järjestelmässä on sisäisesti ja ulkoisesti tarjolla. Kaikki järjestelmän sisäiset, käyttöliittymäympäristön ja tietokantaympäristön väliset palvelupyynnöt kulkevat läpi palveluväyläympäristön, joka reitittää pyynnöt järjestelmän eri osiin. Palveluväyläkerros on siis paitsi järjestelmän keskimmäinen, myös keskeisin taso, sillä sen tehtävä on nivoa kaikki järjestelmän eri osat yhteen. Kuten Kuntalaistilijärjestelmän yleinen looginen arkkitehtuurikaavio kuvassa Kuva 3 paljastaa, kaikkien järjestelmän sisäisten rajapintojen lisäksi palveluväyläkerros tarjoaa myös järjestelmän ulkoiset rajapinnat, joita pitkin voidaan kutsua muita järjestelmiä, tai joita pitkin muut järjestelmät voivat kutsua Kuntalaistilijärjestelmää. Näin ollen suurin osa järjestelmän bisneslogiikasta löytyy palveluväyläkerrokselta. Yksinkertaisimmissa tilanteissa palveluväylän bisneslogiikaksi riittää, että se reitittää vastaanottamansa pyynnöt oikeisiin osoitteisiinsa, mutta useimmiten palveluprosessiin joudutaan osallistumaan aktiivisemmin.



Kuva 3: Kuntalaistilijärjestelmän yleinen looginen arkkitehtuuri [Propentus2008]

Palveluväylän suorittamiin perustoimenpiteisiin yksinkertaisimmissakin reitityspyynnöissä sisältyy käyttöoikeuksien tarkistaminen, kirjausketjun täyttäminen ja tilastointi. Kukin toimenpiteistä muodostaa oman loogisen komponenttinsa, joka automaattisesti sisältyy kutsuttuun web-palveluun (Kuva 4). Monissa tilanteissa yksi palvelupyyntö voi vaatia (em. peruskomponenttien automaattisen kutsumisen lisäksi) useamman kuin yhden järjestelmän komponentin kutsumista onnistunutta suoritusta varten. Näissä tilanteissa palveluväylällä olevan logiikan pitää lähettää kutsuja useille eri tahoille, odottaa vastauksia, yhdistää vastaanottamansa tiedot ja muokata niitä uudelleen seuraavan vastaanottajan ymmärtämään muotoon.



Kuva 4: Palveluväyläkerrokselle saapuvat palvelupyynnöt aiheuttavat automaattisen kutsun käyttöoikeuksien tarkastamiseen, kirjausketjun muodostamiseen ja palvelupyyntöstatistiikan ylläpitoon ennen palvelupyynnön ohjaamista eteenpäin [Propentus2008]

4.3.1 Järjestelmän julkiset rajapinnat

Kuten yleisestä arkkitehtuurikaaviosta (Kuva 3) ilmenee, kaikki palveluväylän julkiset (käyttöliittymältä tai ulkoisesta web-sovellukselta kutsuttavat) rajapinnat on toteutettu web-palveluina. Julkisten rajapintojen kautta kutsuttavat palvelut toteuttavat aina sellaisen toimenpidejoukon, jonka lopputulosta voi luonnehtia lopulliseksi. Esimerkiksi, jos toimenpiteiden tarkoituksena on suorittaa tilaan kohdistuva varaus, operaation loputtua varaus on tehty, ja siihen on samalla liitetty kaikki oleelliset tiedot varaajasta, ilman erillistä käyttöliittymältä tulevaa kutsua varaajan tietojen liittämiseksi varaukseen. Julkisia rajapintoja käyttäen osa järjestelmän palveluista voidaan liittää johonkin toiseen järjestelmään, tai järjestelmälle voidaan rakentaa kokonaan vaihtoehtoinen käyttöliittymä kolmannen osapuolen toimesta. Koska käyttöliittymäkerroksen ja palveluväyläkerroksen välinen viestiliikenne on tämän dokumentin kirjoittamiseen mennessä tapahtuneen testauksen perusteella sujunut järjestelmän kuormituksen ja yleisen toimivuuden kannalta pääosin moitteettomasti, koettiin WS-protokollan käyttö julkisissa rajapinnoissa toteutustavaksi, jota ei ole tarvetta muuttaa.

4.3.2 Järjestelmän yksityiset rajapinnat

Tietokantakerroksen rajapinnat ovat järjestelmäkokonaisuuden sisäisiä, yksityisiä rajapintoja, jotka ovat tarjolla vain palveluväyläkerrokselta kutsuttuina. Kirjoitushetkellä kaikki tietokantakerrokselta tarjotut palvelut käyttävät WS-protokollaa liikennöintiin. Koska yksi käyttöliittymätasolta tullut sisällöltään yksinkertainenkin palvelupyyntö voi liipaista useita, jopa kymmeniä palveluväylältä tietokantakerrokselle kohdistuvia palvelupyyntöjä, jotka sisältävät monimutkaisia olioita parametreina, voi huonosti valittu protokolla suurilla liikennemäärillä vaikuttaa järjestelmän kokonaistehokkuuteen huomattavasti julkisten rajapintojen protokollavalintaa enemmän. Web-palveluiden käyttämä SOAP ei XML-sovelluksena ole mikään tiivis tapa enkoodata viestiliikennettä, joten tietokantarajapintoja on syytä harkita uudelleen, mikäli järjestelmä suurella kuormituksella rupeaa tuntumaan kohtuuttoman hitaalta.

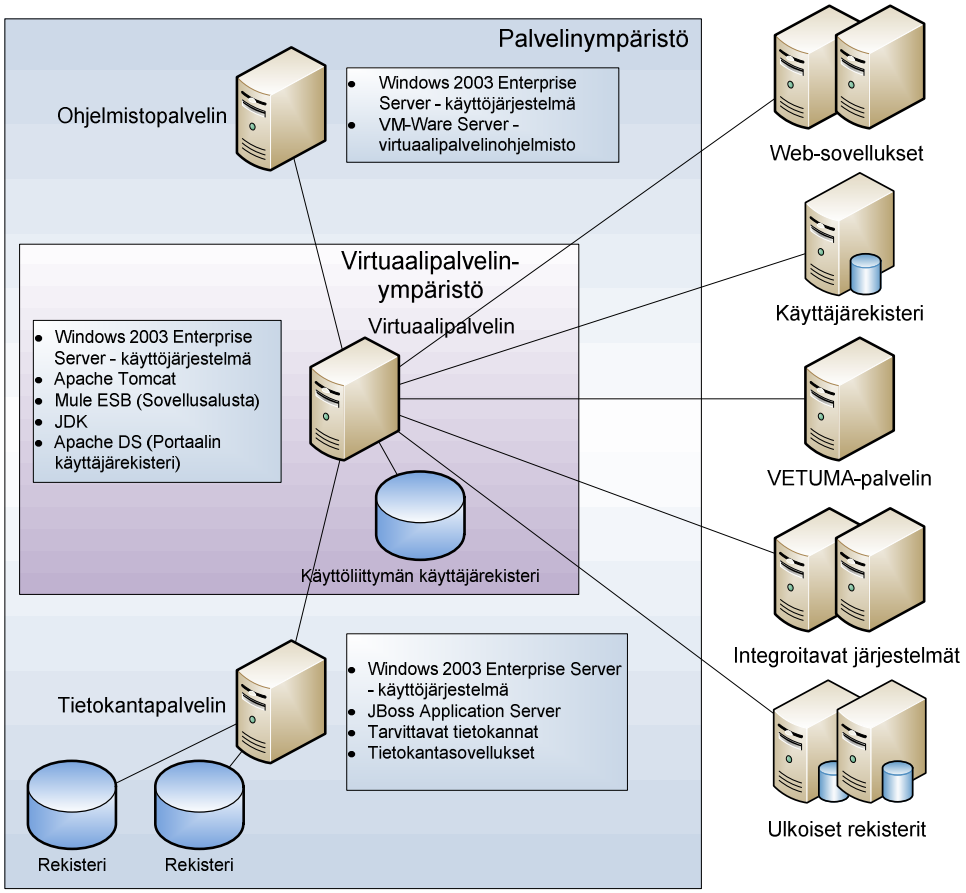
4.3.3 Ulkoisten järjestelmien rajapinnat

Kaikki palvelut, joita Kuntalaistilijärjestelmään tehdään tai liitetään, ovat saatavilla ainoastaan palveluväyläkerroksen julkisten rajapintojen kautta. Osa palveluista käyttää hyväkseen jo olemassa olevia palveluita, joita jokin ulkoinen järjestelmä tarjoaa. Tällaisia ovat esimerkiksi Tekla XCity, Itella iPost ja Commit; MIS (Management Information System). Useimmissa tapauksissa palveluväylälle tulleet palvelupyynnöt liipaisevat vain järjestelmän sisäisiä tietokantapalveluita, jolloin käytetään yllä (kappale 4.3.2) määritettyjä yksityisiä rajapintoja ja niiden myötä yhdenmukaista protokollaa, mutta mikäli tieto tai kutsuttava toimenpide on saatavilla jostakin Kuntalaistilijärjestelmän ulkopuolisesta palvelusta, käytettävät protokollat voivat vaihdella. Muun muassa tästä syystä palveluväyläkerros rakentuu ESB-sovellusalustan ympärille. ESB-sovellusalustat on suunniteltu ymmärtämään useita eri protokollia ja tarjoamaan valmis tuki protokollien yksinkertaista käyttöä varten.

4.4 Fyysinen arkkitehtuuri

Kuntalaistilijärjestelmä noudattaa fyysiseltä arkkitehtuuriltaan jossain määrin kappaleessa 4.1 esiteltyä loogista arkkitehtuuria. Kukin looginen kerros pyörii oman palvelinohjelmistonsa päällä. Yksinkertaisimmillaan kaikki

palvelinohjelmistot voidaan asentaa yhdelle palvelintietokoneelle. Koska kerrosten integraatio ei ole monoliittinen vaan hajautetusti vuorovaikutteinen, voidaan kutakin palvelinohjelmistoa ajaa omassa virtuaalipalvelimessaan, mikä saattaa helpottaa järjestelmän ylläpitoa tai parantaa tietoturvaa. Fyysinen arkkitehtuuri voidaan loogisen arkkitehtuurin joustavuuden ansiosta mukauttaa myös suurta kuormaa vastaavaksi kokonaisuudeksi. Kukin looginen kerros voidaan jakaa omalle palvelimelleen, ja monipalvelinympäristön käyttöönotto onnistuu myös loogisten kerrosten sisäisesti. Kokoonpano, jonka pitäisi riittää keskimääräiselle kuormalle, koostuu kahdesta tietokoneesta, joista toinen ajaa pelkästään tietokantasovelluksia. Toiselle palvelimelle on tällöin asennettu sekä sovelluspalvelin että käyttöliittymäpalvelin joko sellaisenaan tai omaan virtuaalipalvelinympäristöönsä. Tällaista oletuskokoonpanoa vastaava kokoonpano esitetään kuvassa Kuva 5.



Kuva 5: Esimerkki Kuntalaistilijärjestelmän fyysisestä palvelinympäristöstä [Propentus2008]

Kukin Kuntalaistilijärjestelmän looginen osa vaatii oman palvelinohjelmistonsa toimiakseen. Esimerkkikuvassa (Kuva 5) kuntalaistilijärjestelmä käyttää palvelimilla maksullista Windows 2003 Server -käyttöjärjestelmää, mutta se voidaan korvata muillakin vaihtoehdoilla. Esimerkki vapaan lähdekoodin ohjelmistoihin perustuvasta kokoonpanosta voi olla esimerkiksi Centos Linux -käyttöjärjestelmälle rakentuva ohjelmistokokonaisuus: Tietokantakerros toimii omana palvelimenaan, jolle asennetaan Red Hatin nykyisin omistaman JBoss-palvelimen ilmainen community-versio. Lisäksi käytetylle tietokantapalvelimelle on asennettava JDK (Java Development Kit) sekä tietysti asianmukaiset tietokannat, joiden määrä riippuu siitä, mitkä kaikki sisältömoduulit Kuntalaistilijärjestelmän järjestelmäkokonaisuuteen on asiakaskohtaisessa konfiguraatiossa otettu mukaan. Palvelin- ja käyttöliittymäkerroksia voidaan myös ajaa Centos Linuxin tai vastaavan järjestelmän päällä. Molempien kerrosten palvelimille on asennettava tietokantakerrosta vastaavasti JDK. Tämän lisäksi palvelukerrospalvelimelle on asennettava Mule ESB -palveluväyläohjelmisto. Käyttöliittymäkerrokselle on asennettava Apache Directory Server -hakemistopalvelin sekä Apache Tomcat -sovelluspalvelin, jonka päällä varsinainen käyttöliittymä toimii.

Kaikkien kolmen loogisen kerroksen toteutus on edellisen esimerkin tavoin mahdollista pelkkiä avoimen lähdekoodin ohjelmistoja käyttäen. Avoimen lähdekoodin edut tulevat esille jo järjestelmäkehityksen aikana, jolloin koodin virheenjäljityksen (debuggaus) yhteydessä nähdään, miten sovellusalustat suorittavat toimenpiteitään. Myöhemmin avoimuus voi ilmetä esimerkiksi kyseisten palvelinohjelmistojen nopeina virhekorjauksina. Lisäksi kaikki mainitut avoimen lähdekoodin sovellukset ovat myös ilmaisia, joten Kuntalaistilijärjestelmän käyttöönotto on jossakin määrin edullisempaa, kuin se olisi suljetun lähdekoodin vastineita käyttäen. Haittana avoimen lähdekoodin ohjelmistoista ja niiden ilmaisuudesta voi olla, ettei ongelmatilanteissa välttämättä ole asiakastukea tai vastaavaa tahoa, jonka puoleen kääntyä ongelmatilanteissa, vaan ongelmat pitää osata ratkaista itsenäisesti.

4.5 Havaintoja Kuntalaistilijärjestelmän arkkitehtuurista ja sen toteuttamistekniikoista

Kuntalaistilijärjestelmä noudattaa suunnittelumalliltaan hyvin paljon ideaalista SOA-mallia. Valittua mallia voidaan puolustaa tiettyihin käytännön syihin vedoten: Se mahdollistaa loogisen kokonaisuuden osalta järjestelmän tavanomaista helpomman laajennettavuuden, moduuli- ja komponenttikohtaisen modifioinnin muuttuvien tarpeiden mukaiseksi koko järjestelmän eheyttä vaarantamatta, ja periaatteessa melko selkeän, järjestelmän tarjoamaa toiminnallisuutta seurailevan arkkitehtuurin. Komponenttien rajapintojen toteuttaminen web-palveluina tuo järjestelmään teknologiariippumattomuutta. Arkkitehtuuri ja toteutusratkaisut tuovat mukanaan sekä etuja että haittoja, mutta Kuntalaistilijärjestelmän tapauksessa edut arvioitiin haittoja suuremmiksi.

4.5.1 Arkkitehtuuri- ja teknologiaratkaisujen arvioituja etuja

Kuntalaistilijärjestelmään valitun arkkitehtuurin positiiviset vaikutukset kohdistuvat tekniseen projektinhallintaan, ohjelmistosuunnitteluun ja ohjelmistotototeutukseen, sekä asiakkaan etuihin. Projektinhallinnallisia vaikutuksia arkkitehtuurin valinnalla on muun muassa projektirakenteeseen, joka seuraa lähes orjallisesti sisältömoduuleja ja palvelukokonaisuuksia horisontaalisessa suunnassa, sekä ohjelman kerrosmallia vertikaalisessa suunnassa. Tällaista projektirakennetta voidaan käyttää suoraan ohjelman lähdekoodien ryhmittelyssä ohjelmointiympäristön projekteihin, mikä taas helpottaa lähdekoodin ylläpitoa ja projektin etenemisen seurantaa.

Toteutustekniseltä kannalta palveluperustainen rakenne yksinkertaistaa metodien suunnittelua ja toteutusta, sillä käyttöliittymältä tuleva palvelupyyntö kulkee ”putkimaisesti” tietokannalle ja takaisin, lähes aina samaa kaavaa noudattaen, mikä mahdollistaa koodin kierrättämisen. Käyttöliittymäkerrokselle on nopeaa tuottaa yksinkertaisia integraatiotestejä, joiden palvelupyynnöt kulkevat automaattisesti samojen rajapintojen kautta kuin oikeatkin pyynnöt. Asiakkaalle SOA-mallista on etuna mahdollisuus myöhemmin tilata jokin järjestelmän osa toiselta toimittajalta, koska rajapinnat ovat avoimia ja standardinmukaisia web-palveluita, eivätkä täten sido käyttämään yhtä toimittajaa.

4.5.2 Arkkitehtuuri- ja teknologiaratkaisujen arvioituja haittoja

SOA-mallin, palveluväylän ja web-palveluiden käytöstä voidaan arvioida etujen lisäksi olevan tiettyjä haittoja ohjelmiston käytännön toteutukseen ja tehokkuuteen. Web-palvelurajapintojen kirjoittaminen vaatii oman aikansa ja lisää ohjelmakoodin määrää, jolloin oleellinen logiikka voi olla hieman vaikeampaa löytää kaiken koodin seasta. Hieman enemmän harmia aiheutuu siitä, että rajapinnat ovat tietynlaisia ”epäjatkuvuuspisteitä” ohjelmakoodin seassa. Käytetty kehitysympäristö osaa alleviivata väärinkirjoitetut metodikutsut, joten syntaksivirheitä ei usein pääse syntymään. WS-rajapinnoissa metodikutsut kuitenkin esitetään merkkijonoina, jolloin automaattinen syntaksivirheiden tarkistus ei toimi. Tämä johtaa toisinaan vaikeasti löydettäviin virheisiin koodissa. Sama ongelma tulee esille myös silloin, jos olemassa olevia web-metodeja muokataan. Palveluiden välittämien olioiden serialisointi ja deserialisointi XML-muotoon ei myöskään aina suju ongelmitta. Rajapinnat ovat kuitenkin useimmiten hyvin yksinkertaisia, joten niiden hidastava vaikutus koko projektin etenemiseen on vähäisempää kuin arkkitehtuurin kokonaisuutta yksinkertaistavista vaikutuksista saatava nopeusetu projektin toteuttamisaikataulussa.

Projektin etenemiseen kohdistuvia vaikutuksia merkittävämpänä uhkana koettiin SOA-WS-yhdistelmän vaikutus valmiin järjestelmän tehokkuuteen. SOA-mallissa yleiset palvelukutsut WS-rajapintojen yli luovat verkkoon paljon liikennettä, sillä XML-muotoiset viestit sisältävät tyypillisesti overheadia huomattavasti enemmän kuin hyötykuormaa. Kolmikerroksisena arkkitehtuurina Kuntalaistilijärjestelmässä jokainen käyttöliittymältä ESB:n läpi tietokannalle tuleva palvelukutsu serialisoidaan ja deserialisoidaan kahteen kertaan. Suuri osa käyttöliittymältä kutsuttavista palveluista aiheuttaa useita, jopa kymmeniä palveluväylän tuottamia täydentäviä palvelukutsuja. XML-liikenteen aiheuttama tietoliikennekuorma yhdessä serialisointi-deserialisointiproseduurien kanssa voi suurilla käyttäjämäärillä aiheuttaa pullonkaulan järjestelmään. Erittäin suuria rasituskokeita järjestelmälle ei tämän työn aikana tehty. Pienet rasituskokeet eivät lopullista asennusympäristöä tehottomammalla kehityspalvelimella kuitenkaan aiheuttaneet ongelmia.

4.6 Web-palvelut ja järjestelmän suorituskyky

Vaikka tehokkuuteen kohdistuvia uhkia onkin järjestelmäarkkitehtuurin vuoksi löydettävissä, vastatoimia potentiaaliselle web-palveluiden aiheuttamalle ylikuormitukselle on silti kuviteltavissa. SOA-arkkitehtuurin viestiliikenne voi aiheuttaa ylimääräistä kuormaa, mutta samalla se mahdollistaa loogisen kokonaisuuden hajauttamisen usealle fyysiselle palvelimelle kuormituksen hallitsemiseksi. Jos palvelimet sijaitsevat fyysisesti samoissa tiloissa, voidaan palveluväyläpalvelimien ja tietokantapalvelimien välille asentaa nopea fyysinen tietoliikenneyhteys, kuten 1 gigabitin Ethernet tai Infiniband. Parhaat tulokset todennäköisesti saavutetaan kuitenkin oireiden hoitamisen sijaan poistamalla niiden aiheuttaja, eli tietoliikenteen overhead. Koska oletuskokoonpanossa sekä tietokantapalvelimen että palveluväylän sovellukset on toteutettu Java-tekniikalla, voidaan niiden välisten HTTP/SOAP-viestien sijaan ottaa käyttöön jokin Javan omista oliopohjaisista tietoliikenneprotokollista.

Hajautetuissa järjestelmissä, joissa viestiliikennettä on paljon, valittu protokolla voi vaikuttaa järjestelmän kokonaistehokkuuteen huomattavasti. Javan RMI-protokolla (Remote Method Invocation) on huomattavasti HTTP/SOAP-perustaista WS-protokollaa nopeampi [Juric2005], ja siten erittäin hyvä vaihtoehto web-palveluille, mikäli ohjelmointikieliriippumattomuudesta ollaan valmiita luopumaan. Juric et al. [Juric2005] ovat tutkimuksissaan havainneet dramaattisia, yli kymmenkertaisia nopeuseroja viestiliikenteessä web-palveluiden ja RMI:n välillä. Sekä web-palvelurajapinnat että RMI-rajapinnat voivat olla käytössä samanaikaisesti, joten mikäli tehontarve havaitaan vasta, kun järjestelmä on jo tuotantokäytössä, voidaan nopeammat RMI-yhteydet ottaa myös asteittain käyttöön. Järjestelmän sisäisissä integraatioissa on siis kirjoitushetkellä paljonkin potentiaalia suorituskyvyn kasvattamiseen luopumalla WS-protokollasta ja siirtymällä RMI:hin. Ulkoisten järjestelmien integroinnissa RMI-protokollan käyttäminen ei kuitenkaan usein ole mahdollista, vaan pääsääntöisesti joudutaan tyytymään valmiiksi tarjottaviin protokolliin. Suurin vaikutus järjestelmän kokonaistehokkuuteen on kuitenkin valtaosan toiminnallisuudesta muodostavilla sisäisillä palveluilla.

5 KUNTALAISTILIJÄRJESTELMÄÄN TOTEUTETTU YLEINEN PALVELUKUTSUMUODOSTIN

Lahti Fenix -projektin Kuntalaistilijärjestelmä jäsentyy rakenteellisesti kolmikerrosarkkitehtuuriksi, jossa ylin kerros huolehtii käyttöliittymästä, alin kerros tietokannoista ja keskimmäinen yleisluontoisesta (ei tietokantaan tai käyttöliittymään liittyvästä) bisneslogiikasta. Keskimmäinen eli palveluväyläkerros toimii lisäksi integraatiopisteenä, joka huolehtii käyttöliittymäkerroksen, tietokantakerroksen sekä järjestelmän ulkoisten osien toisiinsa liittämisestä. Kun Kuntalaistilijärjestelmää käytiin toteuttamaan, valittiin palveluväyläratkaisuksi avoimeen lähdekoodiin perustuva Mule ESB, jonka uusin versio tuolloin oli 1.4.2. Myöhemmin Mule päivitettiin versioon 1.4.3, joka oli käytössä myös tämän työn aloittamisen aikaan.

Suurin osa palveluväylän välittämistä palvelupyynnöistä kulkee käyttöliittymäkerrokselta tietokantakerrokselle. Ne eivät tule järjestelmän ulkopuolelta, eivätkä ne missään vaiheessa käy Kuntalaistilijärjestelmän ulkopuolella, joten niitä kutsutaan tässä yhteydessä sisäisiksi palvelukutsuiksi ja niiden toteutusta kokonaisuutena järjestelmän sisäisiksi integraatioiksi. Järjestelmän jokaisessa web-palvelussa toimii useita web-metodeja, ja jokainen yksittäinen metodi vaatii oman palvelukutsunsa käyttöliittymältä palveluväylälle ja palveluväylältä tietokantasovellukselle.

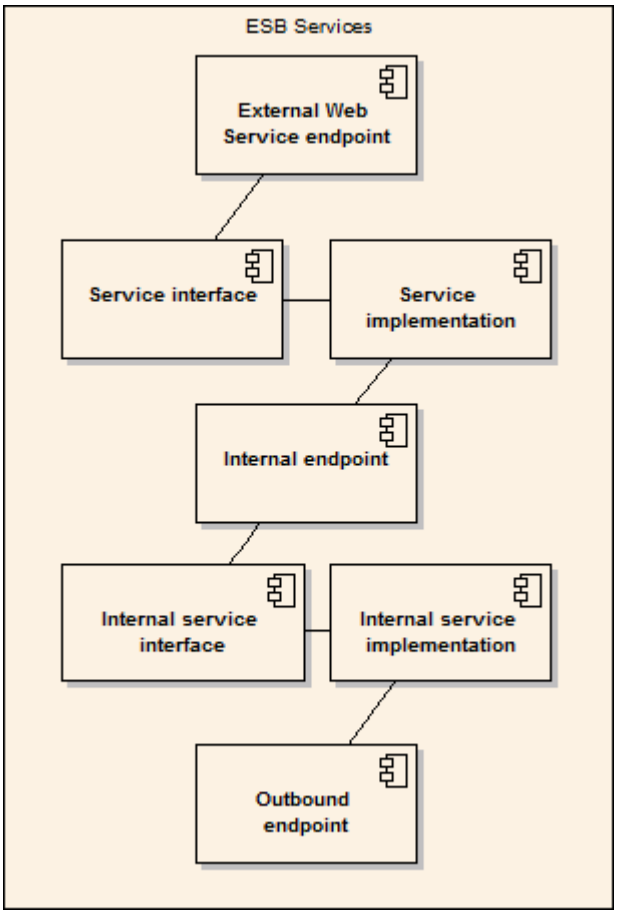
Sisäisiä integraatioita vastaavasti Kuntalaistilijärjestelmään kuitenkin joudutaan toteuttamaan myös ulkoisia järjestelmäintegraatioita ulkoisten järjestelmien tarjoamien tarpeellisten ominaisuuksien käyttöä varten. Tällaisia ominaisuuksia voisivat olla esimerkiksi verkkopankin käyttäminen uimahallivuoron varausmaksun maksamiseksi tai henkilötietojen hakeminen VRK:n VTJ-järjestelmästä. Kuntalaistilijärjestelmän arkkitehtuurin vuoksi tällaiset ulkoiset integraatiot on järkevää toteuttaa samalla periaatteella kuin sisäisetkin integraatiot, eli palveluväylän kautta. Niitä käyttäessä ei kuitenkaan palvelupyyntöä ohjatakaan

ensisijaisesti tietokantatasolle, vaan integroitavan ulkoisen järjestelmän tarjoamaan julkiseen rajapintaan.

Kunkin palvelukutsun muodostaminen edellyttää, että ohjelmistosuunnittelija ohjelmoi siihen tarvittavan logiikan. Web-metodien määrä nousee Kuntalaistilijärjestelmässä useisiin satoihin, joten palvelukutsujen ohjelmointiin menevällä ajalla on huomattava merkitys projektin työmääriin. Tämän vuoksi palvelukutsujen muodostamisen tulisi olla mahdollisimman suoraviivaista ja yhdenmukaista joka tilanteessa, jotta ohjelmistosuunnittelijat voisivat käyttää suuremman osan ajastaan erityislogiikan suunnitteluun ja toteuttamiseen. Tässä osiossa kuvataan, kuinka palvelupyyntöjen välitys järjestelmän sisällä yleisesti ottaen tapahtuu.

5.1 Palvelupyyntöjen välitys palveluväylän läpi

Jokaista palveluväylälle tulevaa palvelupyyntöä kohden suoritetaan tyypillisesti kappaleessa 4.3 ja kuvassa Kuva 4 kuvattu joukko vakiotoimenpiteitä: Pyynnön lähettäjän oikeudet pyynnöstä seuraaviin toimenpiteisiin tarkistetaan ja niiden perusteella joko jatketaan prosessia tai keskeytetään se, muodostetaan kirjausketju tapahtumista ja ylläpidetään palvelutilastoja. Koska näiden toimenpiteiden lisäksi suoritetaan kuitenkin myös kyseiseen palveluun erikseen ohjelmoitavaa bisneslogiikkaa, on palveluketjussa kaksi komponenttia, joihin ohjelmistosuunnittelija voi koodiaan liittää. Palvelupyynnön kulkema reitti palveluväylän läpi kuvataan alla tarkemmin (Kuva 6: ESB-palvelun läpileikkaus).



Kuva 6: ESB-palvelun läpileikkaus [Propentus2008]

Kaikki palveluväylälle ohjautuneet pyynnöt saapuvat kuvan Kuva 6 mukaisesti palveluväylän niin sanottuun ulkoiseen päätepisteeseen (External Web Service endpoint), josta ne ohjataan sopivan Java-rajapinnan (Service interface) toteuttavalle luokalle (Service implementation), jossa palvelupyyntö päätyy oikeaan metodiin. Metodi toteuttaa granulariteetiltaan karkean palvelun. Tähän asti kaikki tapahtuu automaattisesti Mule-sovelluskehyksen ohjaamana. Toteuttavalle luokalle saavuttuaan sovelluskehys antaa ohjakset ohjelmistosuunnittelijan kirjoittamalle bisneslogiikalle (BL, business logic). Tähän erikseen kirjoitettavaan bisneslogiikkaan tulee sisältyä aiemmin mainittu käyttöoikeuksien tarkistus. Sovelluskehys ei siis toteuta niitä automaattisesti.

Toteuttavan luokan BL kutsuu käyttämiänsä, granulariteetiltaan itseään hienompia palveluita ja jää tarvittaessa odottamaan niiltä vastausta. Mule havaitsee palvelukutsun ja siirtyy taas ohjaamaan prosessia. Ulkoisen rajapinnan

toteuttavalta luokalta palvelupyynnöt lähtevät Mulen sisäiseen päätepisteeseeen (Internal endpoint), josta ne ohjautuvat taas sopivaan, Mulen ns. sisemmän rajapinnan (Internal service interface) toteuttavaan luokkaan (Internal service implementation). Kuten ulkoistenkin luokkien tapauksessa, suoritusvastuu siirtyy taas ohjelmistosuunnittelijan toteuttamalle logiikalle.

Sisäisissä luokissa voidaan toteuttaa vielä ”mikrotason” bisneslogiikkaa, mutta joka tapauksessa niiden tehtäväksi jää huolehtia kirjausketjun muodostamisesta ja tilastoinnista. Sisemmän rajapinnan palvelut voivat joko palauttaa vastauksen suoraan ulommalle rajapinnalle, tai vielä kutsua esimerkiksi tietokantapalveluita, jolloin Mule tarttuu taas ohjaksiin, ja palvelupyyntö ohjataan sen ulospäin suuntautuvaan päätepisteeseen (Outbound endpoint), mistä se päätyy kutsutulle tietokantapalvelulle, Mulen vaikutuspiirin ulkopuolelle. Synkronisten palvelukutsujen tapauksessa Mule jää odottamaan vastausta, joka välitetään takaisin alkuperäisen, palveluväylälle pyynnön lähettäneelle asiakkaalle kulkemalla yllä kuvattu polku takaperin. Asynkronisten palvelukutsujen tapauksessa vastausta ei jäädä odottamaan. Muutamaa poikkeusta lukuun ottamatta kaikki Kuntalaistilijärjestelmän palvelukutsut ovat synkronisia, joten synkronisuus voitiin yleistää oletustapaukseksi suunniteltaessa palvelukutsujen yleistä ohjelmointimallia.

5.2 Yleisen palvelukutsumallin laatiminen sisäisissä integraatioissa käytettäväksi

Koodin muokkaaminen käsin kutakin toteutettavaa web-metodia vastaavaksi on hidasta ja virhealtista, eikä lopputuloskaan ole siistiä. Mikäli jossakin vaiheessa kaikkiin toteutettuihin web-metodikutsuihin haluttaisiin lisätä jokin yleisluontoinen lisätoimenpide, jouduttaisiin joka ikinen web-metodikutsu käymään läpi, mikä on paitsi hidasta, myös puuduttavaa. Hyödyntämällä Java-kielen kehittyneitä ominaisuuksia, kuten reflektiivistä ohjelmointia [Sun1998] ja Javan versiossa 5 esiteltyä Generics-ohjelmointitapaa [Sun2004], toteutettiin työssä [Mustonen2009a] dynaamiset metodit, joiden avulla minkä tahansa web-metodin kutsuminen suoraviivaistui huomattavasti.

Palveluväylän ulomman ja sisemmän rajapinnan toteutukset muistuttivat lähtötilanteessa toisiaan. Kun niiden suhteen ruvettiin laatimaan dynaamista toteutusmallia käyttöliittymätason palvelukutsumuodostimen tapaan, yritettiin tästä yhdennäköisyydestä johtuen keksiä tapa yhdenmukaistaa molempien rajapintojen kutsut keskenään täsmälleen samanlaisiksi, jotta koodin uudelleenkäyttö olisi mahdollisimman tehokasta. Hyödyntämällä geneerisyyttä ja reflektiota sekä Javan metodipinoa, saatiin aikaan yleiskäyttöinen palvelukutsulogiikka, joka esitellään tarkemmin työssä [Mustonen2009a] ja jonka käyttöesimerkki nähdään alla olevassa listauksessa (Listaus 5):

```
1  public ResponseTO saveResource(Resource resource) {
2       return request (resource);
3   }
```

Listaus 5: Request-logiikalla toteutettu saveResource-palvelukutsu palveluväylän ulommalta rajapinnalta sisemmälle rajapinnalle tai sisemmältä rajapinnalta tietokannalle – sekä ulompi että sisempi rajapintakutsu näyttää yksinkertaisimmillaan täysin samalta

5.3 Palvelukutsujen reitittäminen Mule-sovelluskehyksen avulla

Työssä [Mustonen2009a] laadittu palvelukutsumuodostin (käyttöesimerkkinä Listaus 5, yllä) suunniteltiin käytettäväksi ensisijaisesti järjestelmän sisäisissä integraatioissa. Sen toiminta perustuu pohjimmiltaan järjestelmän luokkien, rajapintojen ja metodien yhdenmukaiseen nimeämiskonventioon. Käytettyjen Java-ratkaisuiden ja niitä varten sovitun nimeämiskonvention avulla muodostin osaa kutsua Mule-sovelluskehykseltä kutsuvan metodin nimeä vastaavaa päätepistettä, tapahtuipa kutsu sitten palveluväylän ulommalta tai sisemmältä rajapinnalta. Kun käyttöliittymätasolta kutsutaan tietyn nimistä palvelua, kysely ohjautuu web-palvelupyyntönä Mule-sovelluskehyksen ulompaan rajapintaan, samannimiseen päätepisteeseen. Mule on konfiguroitu siten, että web-metodin toteutuksesta vastaa täsmälleen samanniminen Java-metodi. Samalla tavalla kutsu ohjataan Mule-konfiguraation avulla ulommalta rajapinnalta sisemmälle tai sisemmältä vielä eteenpäin, oletetulle tietokantatason palvelulle.

Palvelukutsujen reitittäminen perustuu siis sille oletukselle, että Mule-sovelluskehys on konfiguroitu siten, että sen tarjoamat web-metodit ovat samannimisiä kuin niitä kutsuvat Java-metodit, ja että myös web-metodien Java-toteutukset ovat samannimisiä kuin itse web-metodit. Esimerkiksi yllä olevassa listauksessa (Listaus 5) olevan Java-kielisen `saveResource`-metodin esittely julkiseksi, ulkoisen rajapinnan toteuttavaksi web-palveluksi tapahtuu kirjoittamalla Mule-konfiguraatiotiedostoon alla oleva, XML-muotoinen määrittely (Listaus 6):

```
1  …
2  <mule-descriptor name="ResourceUMO"
   implementation="com.propentus.psc.rema.services.ResourceImpl">
3  …
4  <router
   className="org.mule.routing.outbound.FilteringOutboundRouter">
5      <endpoint
   address="vm://ResourceManagementInternal?method=saveResource">
6      </endpoint>
7      <filter expression="remoteMethodName=saveResource"
   className="org.mule.routing.filters.MessagePropertyFilter" />
8  </router>
9  …
```

Listaus 6: saveResource-metodin ulomman rajapinnan määrittely

Listauksen Listaus 6 rivillä kaksi määritetään itse web-palvelun nimeksi `ResourceUMO` (missä UMO on lyhenne sanoista Universal Messaging Object eli ”yleinen viestintäolio”, Mule-projektin versiossa 1.4 suosittelema nimitys web-palveluille) ja kerrotaan, että `ResourceImpl`-niminen Java-luokka toimii web-palvelun toteuttavana luokkana. Riveillä 4-8 määritellään `saveResource`-metodin kannalta oleellinen reititystietokokonaisuus: Rivillä seitsemän kerrotaan, että kyseinen reititystieto koskee kaikkia palvelupyyntöjä, joissa kutsutaan palvelua `saveResource`. Rivillä viisi kerrotaan, että palvelupyyntö ohjataan suorituksen myöhemmässä vaiheessa `ResourceManagementInternal`-nimisen web-palvelun web-metodille `saveResource`. Samantyyppisillä XML-muotoisilla reititysohjeilla määritellään myös sisemmän rajapinnan web-metodit.

6 KUNTALAISTILIJÄRJESTELMÄN ULKOISET JÄRJESTELMÄINTEGRAATIOT

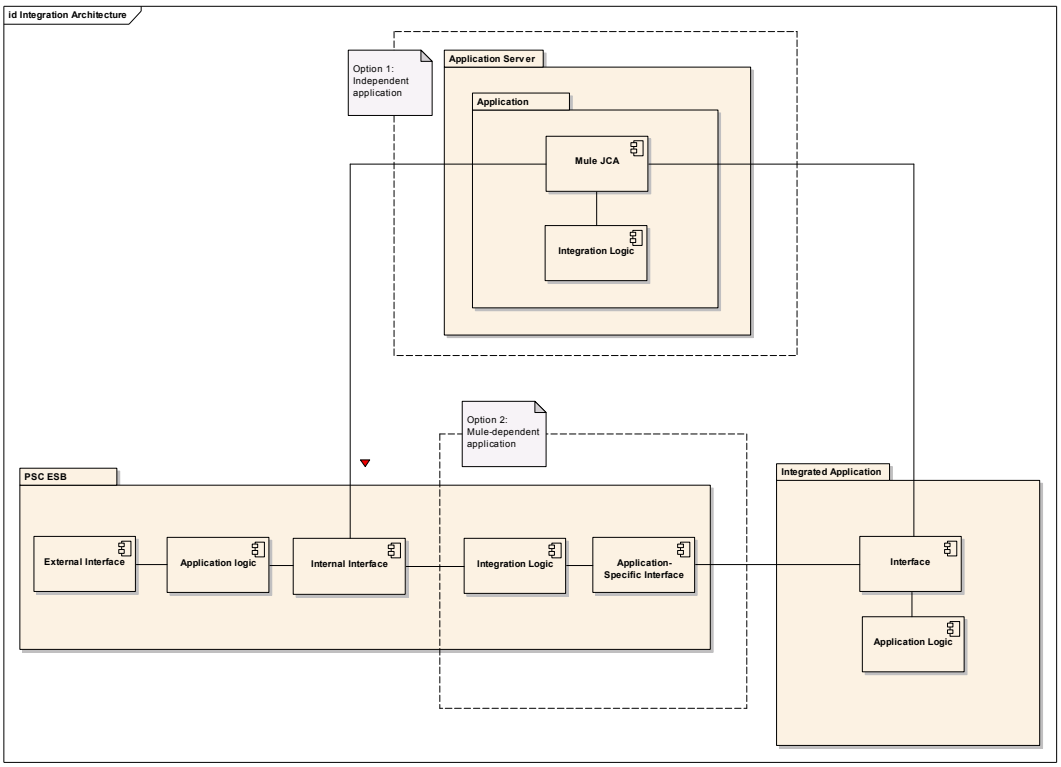
Käyttötarkoituksensa vuoksi Kuntalaistilijärjestelmään toteutetaan integraatioita useisiin ulkoisesti tarjolla oleviin palveluihin. Esimerkiksi joissakin tilanteissa kuntalaisen henkilötiedot voi olla tarpeen hakea jostakin ulkoisesta, kunnallisesta tai valtiollisesta rekisteristä. Samoin henkilön vahvaan tunnistamiseen voidaan käyttää siihen valmiiksi olevaa, luotettavan tahon tarjoamaa palvelua, esimerkiksi Valtiokonttorin alaisuudessa toimivan Valtion IT-palvelukeskuksen (VIP) VETUMA-järjestelmää (Verkkotunnistaminen ja -maksaminen). Myös muita ulkoisia palveluita voidaan ja aiotaan liittää osaksi järjestelmää.

6.1 Ulkoisten järjestelmien integraatiomalli

Vaikka suurin osa Kuntalaistilijärjestelmän palvelupyynnöistä kulkee järjestelmän sisäisesti käyttöliittymältä palveluväylän läpi tietokantatasolle, osa palveluväylälle tulleista pyynnöistä kuitenkin reititetään sellaisille komponenteille, jotka joko luovat uusia palvelupyyntöjä tai ohjaavat niitä järjestelmän ulkopuolelle erilaisia protokollia käyttäen. Monet tahot tarjoavat nykyään julkisia rajapintoja tietojärjestelmiinsä Internetin yli, ja SOA-periaatteiden mukaisesti niiden kautta tarjolla olevia palveluita käytetään myös Kuntalaistilijärjestelmässä niissä tilanteissa, joissa ulkoinen palvelu tarjoaa valmiiksi riittävän hyvin Kuntalaistilijärjestelmän tarpeisiin soveltuvia toimintoja.

Ulkoisten järjestelmien integraatiot voivat vaatia omaa, ulkoisen järjestelmän käyttämään protokollaan tai muihin seikkoihin perustuvaa logiikkaansa, joten integraatioihin liittyvät metodit on järkevää eristää omiksi komponenteikseen, vaikka ne toimivatkin palveluväylätasolla. Eristäminen mahdollistaa mm. erilaisten kuljetusolioiden (Transfer Object, TO) käyttämisen järjestelmän sisäisesti ja järjestelmien välillä. Se taas voi omalta osaltaan lisätä vikasietoisuutta rajapintamuutosten tai järjestelmän sisäisten muutosten yhteydessä. Erillisten komponenttien käytön ansiosta ulkoisten järjestelmien integraatiologiikka ei myöskään sekoitu järjestelmän sisäiseen reitityslogiikkaan. Lisäksi eristämisen myötä on mahdollista jättää ulkoiset integraatiot kokonaan pois joistakin

asennuskonfiguraatioista ilman suuria muutoksia järjestelmän varsinaiseen logiikkaan. Näistä syistä palveluväyläarkkitehtuuria laajennettiin ulkoisten integraatioiden osalta vastaamaan alla olevaa kuvaa (Kuva 7).



Kuva 7: Ulkoisten järjestelmien integraatioarkkitehtuuri

Yllä olevassa kuvassa (Kuva 7) nähdään katkoviivoilla rajattuna kaksi vaihtoehtoista tapaa toteuttaa ulkoisen integraatiologiikan erottaminen kappaleessa 5 kuvatusta sisäisestä logiikasta ja reitityksestä. Palveluväylälle (PSC ESB, Propentus Service Center ESB) saapuva viesti kulkee ylempänä kuvatulla tavalla ulkoiselta rajapinnalta sisemmälle rajapinnalle, mutta sisemmältä rajapinnalta sitä ei ohjatakaan tietokantatasolle, vaan jommallekummalle mainituista ulkoisten integraatiologiikoiden toteutuksista. Riippumatta valitusta toteutuksesta, viesti ohjataan logiikalta ulkoiseen järjestelmään, sieltä tuleva vastaus ohjataan integraatiokomponenteilta palveluväylän sisäiseen rajapintaan ja sieltä edelleen oikeaan paluuosoitteeseensa.

Vaihtoehdoista alemmalla kuvataan tilannetta, jossa integraatiologiikkaan liittyvät komponentit sijaitsevat samalla loogisella palveluväylällä järjestelmän muiden

palveluväylätason osien kanssa, vaikka olisivatkin omassa moduulissaan projektihierarkisesti. Tällaisen rakenteen etuna on, että koko Tilanvarausjärjestelmän käyttöön riittää yksi looginen palveluväyläprosessi. Näin integraatiot on mahdollista toteuttaa ilman tarvetta erillisen palvelimen konfigurointiin, mikä nopeuttaa ja helpottaa järjestelmän ylläpitoa ja vähentää sovelluskehittäjän erehtymisen todennäköisyyttä sovelluskehyksestä toiseen liikuttaessa.

Ylemmässä vaihtoehdossa integraatiologiikka on erotettu omaksi, sovelluspalvelimella erillisesti toimivaksi palveluväyläprosessikseen, johon otetaan yhteyttä järjestelmän yleisen palveluväylän sisäisestä rajapinnasta federointiperiaatteella. Kyseisellä tavalla integraatiologiikan rajapinta on toteutettu ns. Mule JCA-yhdistimenä (J2EE Connector Architecture). Yhdistimen (connector) käyttö mahdollistaa palvelinväyläprosessin liittämisen johonkin jo valmiiksi olemassa olevaan palvelinprosessiin, esimerkiksi samalle JBoss-palvelimelle tietokantatason toimintojen kanssa. Näin ollen muista täysin erillinen palvelinväyläprosessikaan ei välttämättä tarvitse omaa, muista täysin erillistä palvelinprosessia, mikä voi säästää hieman resursseja itsenäisen stand-alone-palvelimen pystyttämiseen verrattuna. JCA-yhdistimen käyttö ei silti estä ajamasta integraatiologiikan sisältävää palvelinväylää omana loogisena palvelimenaankaan. Haittana JCA-mallissa on lisääntynyt palvelinkonfiguroinnin tarve, mutta toisaalta etuna mahdollisuus käyttää täysin halutunlaista palveluväylää: Vaikka Tilanvarausjärjestelmän varsinainen palveluväylä on Mule 1.4.3, voidaan integraatioväylänä käyttää esimerkiksi Mulen versiota 2.0, tai jopa kokonaan eri valmistajan palveluväylää, kuten Apache ServiceMixiä [ServiceMix2009].

6.2 Web-palveluprotokollien eroista

Useimmissa Kuntalaistilijärjestelmän yhteyteen liitettävistä ulkoisista palveluista on tarjolla jonkinlainen web-palvelutyyppinen rajapinta, minkä markkinointihenkisissä puheissa usein luvataan mahdollistavan plug-and-play-tyyppisen järjestelmien välisen yhteensopivuuden ja käyttöönoton helppouden. Esimerkiksi konsulttiyritys Infosys [Infosys] kertoo seuraavaa:

”Web services-based solutions deployed across the enterprise, especially banks, not only allow the developers to leverage plug-and-play logic across platforms quickly but also reduce the total cost of development.”

– [Infosys]

Valitettavasti WS-rajapintojen käyttö ei silti välttämättä ole aivan niin helppoa ja yhteensopivaa, kuin yllä olevassa lainauksessa annetaan ymmärtää. Suurin osa erilaisista nykyisin käytössä olevista WS-protokollista on vielä niin uutta, ettei niiden käytöstä ole vielä vakiintuneita käytäntöjä. Suurin osa ei ole vielä edes valmiin standardin asteella [CBDI2009], ja siten alustavien dokumenttien perusteella joitakin protokollia onkin voitu tulkita eri tavoin eri yrityksissä.

Eroja WS-protokollien toteutuksesta löytyy jopa yleiseen käyttöön vakiintuneista protokollatoteutuksista: Muun muassa oletusarvoisesti käytettävässä SOAP-kehyksessä voi olla eroja riippuen siitä, onko palvelu toteutettu Microsoftin .NET-sovelluskehyksen WCF-tekniikalla (Windows Communication Foundation) tai ASMX-tekniikalla (Active Server Methods) vai Sunin Java-sovelluskehystä käyttäen. Tämän lisäksi erilaisissa autentikointi- ja salausprotokollissa on eroja – Microsoftin tekniikkoina WCF- ja ASMX-pohjaiset palvelut voivat hyvinkin usein käyttää Windows-sidonnaisia protokollia, kuten NTLM-protokollaa (Windows NT LAN Manager) autentikointiin [MSDN2009], kun taas Java-ympäristössä alustariippumattomuuden vuoksi pyritään yleisluontoisempiin standardeihin. Erot tulevat käytännössä esiin seuraavassa osiossa, jossa toteutetaan integraatio .NET-pohjaiseen Tekla Xcity –järjestelmään Java-pohjaista Mule-palveluväylää ja erilaisia web-palvelutekniikoita käyttäen.

7 INTEGRAATIO TEKLA XCITY-JÄRJESTELMÄÄN

Kappaleen 6 johdanto-osassa mainittu valtiollinen henkilötietorekisteri on Väestötietojärjestelmä, johon on tallennettu tiedot kaikista suomalaisista. Sen kunnallinen vastine on Teklan Xcity-järjestelmä, jonka Tekla itse luonnehtii [Tekla2009] olevan *”nykyaikainen ja monipuolinen suomalainen tietojärjestelmä kuntien ja kaupunkien paikka- ja perusrekisteritietojen hallintaan”*. Kunnan tai kaupungin väestötietojen lisäksi Xcity sisältääkin tietoa muun muassa kunnallistekniikasta ja kiinteistönhallinnasta, sekä karttatietoja, joiden avulla voidaan määritellä esimerkiksi eri kiinteistöjen välisiä etäisyyksiä. Kuntalaistilijärjestelmään suunniteltujen palveluiden kannalta näistä palveluista oleellisia ovat ensisijaisesti väestötiedot, ja jossakin määrin myös koulujen karttatiedot. Tekla toteutti Lahti Fenix -projektin yhteydessä Xcity-järjestelmäänsä uuden koulupalvelun ASMX-tyyppisenä web-palveluna. Tilanvarausjärjestelmään toteutettiin rajapinnat paitsi kyseistä koulupalvelua, myös entuudestaan olemassa olevaa WCF-tyyppistä henkilöpalvelua varten.

7.1 Integraatiomoduulin käyttöönotto

Lahti Fenix -projektissa käytettävä, Lahden kaupungin Xcity-järjestelmä sijaitsee palvelimella, johon viitataan tässä työssä nimellä `teky01`. Palvelimelle laadittiin Kuntalaistilijärjestelmän kehitystä varten testirajapinnat Xcity-järjestelmän henkilö- ja koulupalveluihin osoitteisiin `http://teky01/TeklaXcityPersonWebServiceTest/person.svc` ja `http://teky01/TeklaXcityWebServiceTest/School.asmx`. Rajapintojen käyttöä varten palvelimen palomuurin asetuksista sallittiin yhteydenotot Propentus oy:n palvelimelta `teky01`-palvelimelle sopivaan porttiin.

Koska sekä koulu- että henkilöpalvelurajapinnat ovat web-palvelutyyppisiä, sisältävät ne molemmat WSDL-kielisen listauksen metodeistaan. Täten rajapintoihin tutustuminen voitiin aloittaa listauksia lukemalla. Näin palvelurajapinnoista saatiin selvitettyä monia sellaisia yksityiskohtia, joista ei ollut dokumentaatiota tai muuta tietoa saatavilla. Aloittamisvaiheessa oleellisia havaintoja olivat muun muassa SOAP-viestien muodostamiseen liittyvät

yksityiskohdat: SOAP-viestien hyötykuormana välitettävistä kuljetusolioista havaittiin niiden usein olevan `complexType`-tyyppisiksi serialisoitavia (Listaus 7). SOAP-viestien enkoodauksen osalta havaittiin käytettävän tyypin olevan document/literal wrapped (Listaus 7 `ja` Listaus 8).

```
1  …
2  <wsdl:types>
3      <s:schema elementFormDefault="qualified"
   targetNamespace="http://tekla.com/webservices/">
4        <s:element name="GetSchoolDistrict">
5          <s:complexType>
6            <s:sequence>
7              <s:element minOccurs="0" maxOccurs="1" name="Key"
   type="s:string" />
8          …
9            </s:sequence>
10         </s:complexType>
11       </s:element>
12      …
13     </s:schema>
14   </wsdl:types>
15 …
```

Listaus 7: Lahden Xcityn koulupalvelun listauksesta nähdään, että osassa viesteistä käytetään complexType-tyyppisiksi serialisoitavia olioita.

```
1  …
2  <wsdl:binding name="ISchool" type="tns:ISchool">
3      …
4      <soap:operation
   soapAction="http://tekla.com/webservices/GetSchoolDistrict"
   style="document" />
5        <wsdl:input>
6          <soap:body use="literal" />
7        </wsdl:input>
8        <wsdl:output>
9          <soap:body use="literal" />
10       </wsdl:output>
```

```
11      </wsdl:operation>
12  …
```

Listaus 8: WSDL-kuvauksesta havaitaan, että Lahden Xcity-rajapinnoissa on käytössä document/literal wrapped -enkoodaus

Aluksi integraatio Xcityyn aiottiin toteuttaa kuvan Kuva 7 esittämällä varsinaiseen palveluväylään sidonnaisella tavalla. Kuntalaistilijärjestelmässä käytetty palveluväylä on Mule 1.4.3, ja siinä on käytetty viestintäkehyksenä Apache Axis versiota 1.4. Vaikka tällainen yhdistelmä tukeekin document/literal wrapped -enkoodausta [MuleSource2008], ei sitä kovasta yrittämisestä huolimatta saatu toimimaan Xcity-palveluiden kanssa. Internetin keskusteluryhmistä selvisi, että muilla on ollut samanlaisia ongelmia Axis-viestintäkehyksen ja .NET-palveluiden kanssa. Mule 1.4.3. tukee myös Apache XFire -viestintäkehystä, mutta tälläkään yhdistelmällä mm. paluuviestien deserialisointi ei onnistunut.

Koska mitään ratkaisua Xcity-integraatioiden kanssa ilmenneisiin ongelmiin ei muuten löytynyt, päätettiin lopulta ottaa käyttöön uusi Apache CXF-viestintäkehys (CeltiXFire) [CXF2009], joka on yhdistelmä vanhoista Apache Celtix – ja Apache XFire -viestintäkehyksistä [XFire2008]. CXF:n käyttö vaatii uudempaa Mule-versiota, joten integraatiomalliksi valittiin kuvan Kuva 7 yläosassa kuvattu, federointiperiaatteella toimiva vaihtoehto. Uusin saatavilla ollut Mule-jakeluversio oli 2.20, josta asennettiin JCA-yhdistimenä toimiva versio Kuntalaistilijärjestelmän JBoss-palvelimeen, jota aiemmin käytettiin pelkästään tietokantatason toimintoihin.

Kuntalaistilijärjestelmän JBoss-palvelin tarjosi jo entuudestaan palveluväylätasolta kutsuttavia web-palveluyhteyksiä tietokantaoperaatioihinsa. Palvelut tarjottiin Mule 1.4.3-JCA-yhdistimen kautta, joten Mule 2.20 JCA-yhdistimen asennuksen jälkeen palvelin tuli konfiguroida siten, ettei kahden eri version samanaikainen käyttö aiheuta ristiriitoja palvelimen toimintaan. JBoss-palvelimessa voidaan käyttää ns. luokkalataajan eristämistä (classloader isolation) tämän tavoitteen saavuttamiseksi. Laaditun ulkoisen integraatiomallin mukaisesti integraatiologiikkaa varten laadittiin oma erillinen moduuliinsa, josta tehtiin

EAR-tyyppinen (Enterprise Archive) projektipaketti JBoss-palvelimelle. EAR-paketin sisältä viitattiin uuteen Mule 2.20 JCA:han. Samankaltainen pakettirakenne on käytössä myös vanhan Mule 1.4.3-JCA:n kanssa. Pakettien muodostamisen jälkeen JCA-yhdistimiin viittaaviin EAR-projektipaketteihin asetettiin seuraavanlainen, XML-muotoinen JBoss-konfiguraatio luokkalataajan ohjaamiseen (tiedostossa `jboss-app.xml`):

```
1  <loader-repository>org.mule:loader=PSCIntegrationEAR.ear
2          <loader-repository-config>
3              java2ParentDelegation=false
4          </loader-repository-config>
5  </loader-repository>
```

Listaus 9: Luokkalataajan eristäminen JBoss-palvelimella

7.2 Palveluiden olemassaolon varmistaminen

Kun perustavanlaatuiset palvelinkonfiguraatiot saatiin kuntoon, varsinaisen integraation kehittäminen aloitettiin testaamalla Xcity-palveluita ilmaisen Eviware SoapUI-ohjelman [Eviware2009] kautta. SoapUI:ta käyttäen on helppoa muodostaa SOAP-muotoisia viestejä lähetettäväksi haluttuun kohteeseen, ja ohjelmasta näkee myös vastauksen SOAP-muodossa. SoapUI osaa muodostaa oletusmuotoiset viestit automaattisesti palvelun WSDL-tiedostoista, mutta viestejä voi muokata jälkikäteen vapaasti. Yhteydenotto koulupalvelun testirajapintaan sujui ongelmitta, ja palvelu palautti odotetusti SOAP-muotoisen vastauksen SOAP-muotoiseen, WSDL:n perusteella automaattisesti muodostettuun viestiin.

Henkilöpalvelun suhteen yhteydenotto ei sujunut aivan yhtä yksinkertaisesti kuin koulupalvelun suhteen. Koska SoapUI:n mukaan kysely ei palauttanut mitään vastausta, seurattiin OSI-mallin (Open Systems Interconnection) kuljetuskerroksen päällä kulkevaa tietoliikennettä Mule-asennuspaketin mukana tulevalla TCP Monitor -ohjelmalla. TCP Monitorin avulla havaittiin, jälleen odotetusti, varsinaisen TCP/IP-yhteyden toimivan sen ylimmällä tasolla eli sovelluskerroksella asti, ja samalla saatiin paikannettua varsinainen ongelma: `Teky01`-palvelin ei hyväksynyt palvelupyynnön HTTP-otsikkoa (Listaus 10 ja

Listaus 11), vaikka se oli standardinmukainen ja muodostettu suoraan kutsutun henkilöpalvelun WSDL-kuvauksesta.

```
1  POST /TeklaXcityPersonWebServiceTest/Person.svc HTTP/1.1
2  Content-Type: application/soap+xml;charset=UTF-
   8;action=http://tekla.com/webservices/Person/GetPersonsByIdentity
   Code
3  User-Agent: Jakarta Commons-HttpClient/3.0.1
4  Host: teky01:9876
5  Content-Length: 381
```

Listaus 10: Xcityn henkilöpalvelun WSDL-kuvauksesta muodostetun viestin HTTP-otsikko oletusmuodossaan

```
1  HTTP/1.1 415 Cannot process the message because the content type
   'application/soap+xml;charset=UTF-
   8;action=http://tekla.com/webservices/Person/GetPersonsByIdentity
   Code' was not the expected type 'application/soap+xml;
   charset=utf-8'.
2  Date: Fri, 20 Mar 2009 11:10:27 GMT
3  Server: Microsoft-IIS/6.0
4  X-Powered-By: ASP.NET
5  X-AspNet-Version: 2.0.50727
6  Cache-Control: private
7  Content-Length: 0
```

Listaus 11: Xcityn henkilöpalvelun vastaus oletusmuotoiseen viestiin

Kun listauksen Listaus 10 riviä 2 muutettiin siten, että `action`-parametri jätettiin pois kokonaan, saatiin Xcityn henkilöpalvelultakin SOAP-tyyppinen paluuviesti, joka esitetään alla olevassa listauksessa (Listaus 12). Vaikka paluuviesti ilmoittaakin virheestä palvelupyynnössä, toimii se silti osoituksena siitä, että itse palveluun on saatu jonkinlainen yhteys. Koska näin kokeilemalla saatiin todettua, että sekä henkilöpalvelu että koulupalvelu on olemassa, voitiin siirtyä Kuntalaistilijärjestelmän integraatiomoduulin kehittämiseen ja yhteydenoton yrittämiseen Mule-palveluväylän kautta.

```
1  HTTP/1.1 500 Internal Server Error
```

```
2  ...
3  <s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
   xmlns:a="http://www.w3.org/2005/08/addressing">
4  <s:Header><a:Action
   s:mustUnderstand="1">http://www.w3.org/2005/08/addressing/soap/fa
   ult</a:Action></s:Header>
5  <s:Body><s:Fault>
6      <s:Code><s:Value>s:Sender</s:Value>
7          <s:Subcode>
8              <s:Value>a:DestinationUnreachable</s:Value>
9          </s:Subcode>
10     </s:Code>
11     <s:Reason><s:Text xml:lang="fi-FI">The message with To ''
   cannot be processed at the receiver, due to an AddressFilter
       mismatch at the EndpointDispatcher.  Check that the sender
   and receiver's EndpointAddresses agree.</s:Text></s:Reason>
12  </s:Fault></s:Body>
13 </s:Envelope>
```

Listaus 12: Xcityn henkilöpalvelulta saatu vastaus muokatun HTTP-otsikon sisältävään palvelupyyntöön

7.3 Integraatiomoduulin kehittäminen

Kuten ylempänä kuvataan, ulkoisia järjestelmäintegraatioita varten laaditussa arkkitehtuurimallissa integraatiologiikka haluttiin eristää kokonaan Kuntalaistilijärjestelmän muusta arkkitehtuurista. Mallia noudattaen Xcity-integraatioita varten laadittiin oma projektipakettinsa, joka kapseloitiin luokkaeristettyyn EAR-moduuliin. Lisäksi integraatiopalveluväylästä käytettiin huomattavasti järjestelmän pääpalveluväylää uudempaa versiota, joka sisältää kohtalaisen suuria muutoksia palveluväylän kehittämisen kannalta. Em. seikoista johtuen yleiseen käyttöön aiemmin laaditut metodit eivät olleet saatavilla tai muutenkaan käyttökelpoisia Xcity-integraatiologiikan kehittämiseen.

7.3.1 Kuljetusolioiden toteutus

Koska integroinnissa haluttiin huomioida mahdollisimman suuri virheensietokyky ja joustavuus rajapintamuutosten suhteen, integraatiologiikasta järjestelmän sisälle päin näkyvässä rajapinnassa päätettiin käyttää eri TO-olioita, kuin järjestelmästä

ulospäin, ja sovittaa näiden TO-olioiden tiedot soveltuvilta osin dynaamisesti yhteen itse integraatiologiikassa. Erilliset oliot sisäiseen ja ulkoiseen käyttöön auttavat myös piilottamaan jommankumman käytön kannalta epäoleelliset tiedot, mikä ilmenee esimerkiksi listauksista Listaus 13 ja Listaus 14. Ulkoiseen käyttöön tarkoitetut kuljetusoliot toteutettiin ns. sopimus ensin -periaatteella (contract first) suoraan `teky01`-palvelimen WSDL-kuvauksista käyttäen apuna CXF-asennuspaketin mukana tulevaa WsdlToJava-työkalun CXF-versiota [CXF2009]. Sisäiset kuljetusoliot toteutettiin ulkoiseen käyttöön tarkoitettujen olioiden pohjalta. Olioita vertailemalla voidaan havaita, että CXF vaatii serialisointiin huomattavasti enemmän Java-annotaatiomuotoista ohjausdataa kuin järjestelmän sisällä haluttiin käyttää.

```
1  package com.tekla.webservices;
2  …
3  @XmlAccessorType(XmlAccessType.FIELD)
4  @XmlType(name = "", propOrder = {
5      "key",
6      "identityCode"
7  })
8  @XmlRootElement(name = "GetPersonsByIdentityCode")
9  public class GetPersonsByIdentityCode {
10     @XmlElementRef(name = "Key", namespace =
   "http://tekla.com/webservices/", type = JAXBElement.class)
11     protected JAXBElement<String> key;
12     @XmlElementRef(name = "IdentityCode", namespace =
   "http://tekla.com/webservices/", type = JAXBElement.class)
13     protected JAXBElement<String> identityCode;
14     public JAXBElement<String> getKey() {
15         return key;
16     }
17     public void setKey(JAXBElement<String> value) {
18         this.key = ((JAXBElement<String> ) value);
19     }
```

Listaus 13: Xcityn suuntaan käytettävä, WsdlToJava-työkalulla toteutettu kuljetusolio

```
1  package com.propentus.psc.integration.xcity.transferobjects;
2  …
3  public class XcityPersonSearchTO implements Serializable {
4  private static final long serialVersionUID = -
   6931657134977241539L;
5   private String key;
6   private String identityCode;
7   public String getKey() {
8       return key;
9   }
10  public void setKey(String key) {
11      this.key = key;
12  }
13 …
```

Listaus 14: Kuntalaistilijärjestelmän sisäisesti käytettävä, listausta Listaus 13 vastaava kuljetusolio

7.3.2 Reifikaatio geneeristen elementtien reflektoinnissa

Kuten listaukset Listaus 13 ja Listaus 14 paljastavat, ulkoisissa kuljetusolioissa käytetään toisinaan `JAXBElement<String>`-tyyppisiä elementtejä, joita vastaavat `String`-tyyppiset ominaisuudet sisäisissä kuljetusolioissa. Sama tilanne voi toisinaan esiintyä myös `Integer`-, `Float`-, tai minkä tahansa ominaisuuden suhteen. Tästä johtuen dynaamiseen konversiologiikkaan haluttiin toteuttaa geneerinen, mitkä tahansa `JAXBElement<T>`-tyyppiset elementit reflektiivisesti `T`-tyyppisiksi muuntava metodi. Sinänsä yksinkertaiselta kuulostavaa toimenpidettä on kuitenkin yleisesti pidetty mahdottomana toteuttaa geneeristen elementtien ns. reifikaation eli esineellistämisen puutteen takia, mistä esimerkkinä alla oleva lainaus Sun Microsystemsin javac-kääntäjän kehitystiimin projektipäällikön (Ahé, Peter von der) haastattelusta vuodelta 2007 [Heiss2007]:

> *Generics and Lack of Reification*
>
> *[haastattelija]: In your blog, you have identified problems with generics related to the lack of runtime type information about generic types. You point*

out that at runtime, you cannot tell an ArrayList<String> from an ArrayList<Integer>. How would you address this?

Ahé: Let me provide some background on generics. When we added generics to the Java programming language in 2004, it was based on a technique called erasure, which had been selected many years before because it was the only proposal that dealt with migration compatibility.

As of 2004, after more than five years of work in academia and at Sun, there was no known solution that could solve all the compatibility requirements. Jon Gibbons joined the compiler team around the time we shipped JDK 5.0 and offered a solution -- or at least a restatement of the problem. Rather than trying to provide reified types for all instances, why not simply accept that some instances do not have reified type information, because they are compiled with a 1.4.2 compiler or for other reasons?

This turns reification into a best-effort problem: If you use a raw type, then no type information is reified, but if you avoid raw types, you can take advantage of the additional reified type information. It's worth noting that a precondition for Jon's idea is that it is worth having erasure. Today's generic types would not exist without erasure.

- [Heiss2007]

Onneksi tähän aluksi mahdottomalta vaikuttaneeseen tilanteeseen on tarjolla ratkaisu, reifikaation puutteesta huolimatta. Ian Robertson [Robertson2007] esittelee tavan, jolla ongelman voi yleistettyjen listojen tapauksessa kiertää. Muokkaamalla Robertsonin esimerkkiä, laadittiin listauksen Listaus 15 mukainen metodi, jota käyttämällä saadaan selvitettyä minkä tahansa tyyppisiä geneerisiä luokkia käyttävien getter- ja setter-metodien palauttama yleinen `T`:n elementtiluokka. Soveltamalla laadittua metodia esimerkiksi `JAXBElement<T>`-elementteihin listauksessa Listaus 16 esitetyllä tavalla, Kuntalaistilijärjestelmä-

Xcity-välin kuljetusolioiden konversiologiikka, eli muuntaminen muodosta toiseen, saatiin täysin dynaamiseksi.

```
1   private static Class<?> resolveElementClass(Method method) {
2       Type type = null;
3       if (method.getName().startsWith("get")) {
4           type = method.getGenericReturnType();
5       }
6       if (method.getName().startsWith("set")) {
7           type = method.getGenericParameterTypes()[0];
8       }
9       if (type instanceof ParameterizedType) {
10          ParameterizedType parameterizedType = (ParameterizedType)
    type;
11          Type[] parameterArgumentTypes =
    parameterizedType.getActualTypeArguments();
12          return (Class<?>) parameterArgumentTypes[0];
13      }
14      return null;
15  }
```

Listaus 15: Metodi, joka palauttaa annetun getter- tai setter-metodin käyttämän geneerisen luokan tyypin

```
1   private static <T, U> void elementToFieldCopy(T sourceObject,
    Method sourceMethod, U targetObject, Method targetMethod) throws
    RemoteException {
2       Class<?> elementClass = resolveElementClass(sourceMethod);
3       if
    (!elementClass.equals(targetMethod.getParameterTypes()[0])) {
4           return;
5       }
6       try {
7           JAXBElement<?> element = (JAXBElement<?>)
    sourceMethod.invoke(
8               sourceObject, (Object[]) null);
9           targetMethod.invoke(targetObject, element.getValue());
10      } catch (Exception e) {
```

```
11          throw new RemoteException(
12              "Could not copy value from source element to target
    field!");
13      }
14  }
```

Listaus 16: Metodi, joka kopioi JAXBElement<T>-tyyppisen elementin arvon sen T-tyyppiseksi vastineeksi

7.4 Ulkoisen palvelun kutsuminen järjestelmän läpi

Xcity-integraatiot eivät kooditasolla liity järjestelmän pääpalveluväylään, joten palvelukutsuja varten toteutettiin oma Mule 2.20 -yhteensopiva Xcity-palvelukutsumuodostimensa, joka toteutettiin kappaleessa 5 esitellyn yleisen palvelukutsumuodostimen tapaan geneerisin mallein. Lisäksi järjestelmän sisäisesti päätettiin käyttää eri kuljetusolioita kuin ulkoisesti. Tällaisessa mallissa konversio sisäisestä esitysmuodosta ulkoiseen tapahtuu juuri ennen varsinaista palvelukutsun lähettämistä, ja ulkoisesta muodosta sisäiseen vastaavasti heti palveluvastauksen saamisen jälkeen. Jotta automaattisen konvertoinnin sijaan jossakin kohdassa voitaisiin antaa ohjelmistokehittäjälle mahdollisuus tiettyjen arvojen eksplisiittiseen syöttämiseen kummassa tahansa tilanteessa, konversiologiikka toteutettiin varsinaisesta palvelukutsumuodostimesta erillisinä metodeina.

Koska integraatiopalveluväylää ajetaan omana prosessinaan, integraatiomallin käyttöönoton jälkeen palveluväylätasolla toimii kaksi palveluväylää. Tämä ei kuitenkaan tuota mitään ongelmia palveluiden kutsumiseen, sillä molemmat palveluväylät voidaan määrittää vastaanottamaan palvelupyyntöjä eri porttien kautta. Erilaisin konstein myös palveluiden näkyvyys saadaan rajattua halutulle alueelle. Integraatiopalveluiden kutsuminen esimerkiksi käyttöliittymältä tapahtuukin järjestelmän pääpalveluväylän kautta. Listauksissa Listaus 17 ja Listaus 18 esitetään yksinkertaisin mahdollinen palveluväylätason ohjelmallinen kutsuminen. Ensin pääpalveluväylän ulkoisesta rajapinnasta kutsutaan sisäistä rajapintaa ja sisäiseltä rajapinnalta integraatiopalveluväylän integraatiorajapintaa. Integraatiorajapinnalla ensin muunnetaan palvelupyynnön kuormana toimiva

kuljetusolio sisäisestä esitysmuodosta ulkoiseen, lähetetään palvelupyyntö ulkoiseen palveluun, ja muunnetaan palveluvastaus ulkoisesta muodosta sisäiseen esitysmuotoon ennen sen palauttamista takaisin järjestelmän sisäiselle rajapinnalle ja siitä eteenpäin.

```
1  public List<XcityPersonTO> findPersonList(XcityPersonSearchTO
   searchParams) {
2       return request(searchParams);
3   }
```

Listaus 17: Pääpalveluväylällä voidaan integraatiorajapintaa kutsuessa käyttää luvussa 5 laadittua yleistä palvelukutsumuodostinta sekä sisäisellä että ulkoisella rajapinnalla

```
1  public List<XcityPersonTO> findPersonList(XcityPersonSearchTO
   searchParams)
2       throws RemoteException {
3       Object requestObject =
   TeklaClassConversionUtil.convertToTeklaObject(searchParams);
4       PersonSearchResult response =
   TeklaMessagingUtil.teklaRequest("vm://TeklaXcity-
   GetPersonsByIdentityCode", requestObject);
5       return
   TeklaClassConversionUtil.invertToPropentusObject(response);
6   }
```

Listaus 18: Integraatiorajapinta huolehtii paitsi yhteydenotosta ulkoiseen järjestelmään, myös kuljetusolioiden muuntamisesta esitysmuodosta toiseen

7.5 Integraatiopalveluiden reitityskonfiguraatio

Listauksen Listaus 18 rivit 3 ja 5 huolehtivat kuljetusolioiden muuntamisesta ja rivi 4 niiden välissä ulkoisen palvelun kutsumisesta. Riviltä 4 havaitaan helposti, että ulkoisia Tekla Xcity -järjestelmän palveluita kutsuttaessa kutsumetodin argumenttina annetaan URL-tyyppinen (Uniform Resource Locator) osoite. Osoite ei kuitenkaan osoita mihinkään Internetistä löytyvään palveluun, vaan se on Mule-palveluväylän sisäinen ”virtuaalikoneosoite”, jota käytetään tässä

piilottamaan kutsun kannalta epäoleelliset yksityiskohdat, kuten käytettävä protokolla ja palvelun tarkka polku.

Kuten jo aiemmin havaittiin, ASMX-palveluna toteutettu Xcityn koulupalvelu toimii hieman eri tavalla kuin WCF-palveluna toteutettu henkilöpalvelu. Tästä syystä myös niitä kutsuvan asiakaspään konfiguroinneissa voidaan odottaa palvelukohtaisia eroja. Muun muassa tästä syystä konfiguraatiot haluttiin erottaa omiksi XML-tiedostoikseen. Konfiguraatiot ja kutsuttavan palvelun ominaisuudet ovat kuitenkin Kuntalaistilijärjestelmän ohjelmistosuunnittelijan kannalta epäoleellista tietoa, joten niiden ei haluttu näkyvän palvelukutsun osoitteessa. Siksipä palvelukutsuja varten laadittiin vielä oma koontitiedostonsa, joka kääntää kutsut oikeisiin konfiguraatiotiedostoihin. Osa koontitiedoston reitityslistaa nähdään listauksessa Listaus 19.

```
1       <service name="GetDistance2">
2           <inbound>
3               <vm:inbound-endpoint address="vm://TeklaXcity-
    GetDistance2" />
4           </inbound>
5           <outbound>
6               <pass-through-router>
7                   <vm:outbound-endpoint
    address="vm://TeklaXcitySchoolService-GetDistance2"
    synchronous="true"/>
8               </pass-through-router>
9           </outbound>
10      </service>
11  ...
12      <service name="GetPersonPropertiesByIdentityCode">
13          <inbound>
14              <vm:inbound-endpoint address="vm://TeklaXcity-
    GetPersonPropertiesByIdentityCode" />
15          </inbound>
16          <outbound>
17              <pass-through-router>
```

```
18                  <vm:outbound-endpoint
    address="vm://TeklaXcityPersonService-
    GetPersonPropertiesByIdentityCode" synchronous="true"/>
        </pass-through-router>
19          </outbound>
20      </service>
```

Listaus 19: Osa Xcity-palveluiden koontitiedoston reitityslistaa

Listaus 19 osoittaa, kuinka Xcity-palveluiden koontitiedosto ohjaa esimerkiksi `TeklaXcity-GetDistance2`-osoitteeseen tulleen pyynnön eteenpäin osoitteeseen `TeklaXcitySchoolService-GetDistance2`. Tästä osoitteesta voidaan jo päätellä, että kyseessä on koulupalveluun liittyvä reititystieto, ja sen yksityiskohdat löytyvätkin koulupalvelun omasta konfiguraatiotiedostosta (Listaus 21). Mutta koontitiedostolla on toinenkin tarkoituksensa, jolla on muutakin kuin hierarkista merkitystä: se määrittelee protokollan, osoitteen, portin ja viestintäkehyksen, jolla palveluita voidaan ulkoapäin kutsua, ja luokan, joka palvelurajapinnan erikoistuksen toteuttaa. Esimerkiksi listauksen Listaus 18 metodi löytyy konfiguraatiossa ilmoitetusta `XcityManagementImpl`-luokasta. Koska järjestelmän pääpalveluväylällä on pyritty käyttämään Axis-kehystä joka tilanteessa, myös integraatioväylä konfiguroitiin tarjoamaan omat palvelunsa pääpalveluväylälle Axis-tyyppisinä. Koontitiedoston ominaisuudet esitetään listauksessa Listaus 20:

```
1   <service name="XcityIntegrationUMO">
2           <inbound>
3               <inbound-endpoint
    address="axis:http://localhost:8091/service" synchronous="true”/>
4           </inbound>
5           <component class="com.propentus.psc.integration.xcity.
                            service.XcityManagementImpl" />
6       </service>
```

Listaus 20: Xcity-palveluiden koontitiedoston alussa määritellään osoite, josta integraatioväylän palvelut ovat saatavilla, ja luokka, joka palvelut toteuttaa

```
1   <cxf:endpoint
```

```
2       name="SchoolService-CxfOut"
3       applyTransformersToProtocol="true"
4   address="http://teky01/TeklaXcityWebServiceTest/School.asmx?wsdl"
5       synchronous="true"
6    wsdlLocation="http://teky01/TeklaXcityWebServiceTest/School.asmx
    ?wsdl"
7       wsdlPort="ISchool"
8       clientClass="com.tekla.webservices.SchoolService"
9       encoding="utf-8"
10      mtomEnabled="false"/>
11    <model name="XcityIntegration-SchoolService">
12      <service name="SchoolService-GetDistance2">
13          <inbound>
14              <vm:inbound-endpoint
    address="vm://TeklaXcitySchoolService-GetDistance2" />
15          </inbound>
16          <outbound>
17              <pass-through-router>
18                  <cxf:outbound-endpoint ref="SchoolService-CxfOut"
    operation="GetDistance2" />
19              </pass-through-router>
20          </outbound>
21      </service>
22  ...
23  </model>
```

Listaus 21: Xcity-koulupalvelun reititys ja konfiguraatio Kuntalaistilijärjestelmässä

Listauksessa Listaus 21 nähdään osa Kuntalaistilijärjestelmän integraatioväylään toteutettua Xcityn koulupalvelukonfiguraatiota. Konfiguraation riveillä 12-21 esitellään palvelu, jota voidaan kutsua Mulen sisäisesti osoitteesta `vm://TeklaXcitySchoolService-GetDistance2`. Palvelun ulkoinen osoite kerrotaan rivillä 18, kertomalla kutsuttavan web-metodin nimi ja viittaamalla nimettyyn päätepisteeseen `SchoolService-CxfOut`. Kyseinen päätepiste ja siihen sovellettavat asetukset esitellään riveillä 1-10. Näistä asetuksista nähdään vihdoin lopullinen reititystieto: palvelupyyntö reititetään Kuntalaistilijärjestelmän

ulkopuolelle, `teky01`-palvelimella sijaitsevaan Xcity-järjestelmään, osoitteeseen `http://teky01/TeklaXcityWebServiceTest/School.asmx?wsdl`. Muita asetuksista nähtäviä seikkoja ovat mm. käytetty merkistöenkoodaus ja WSDL-portti, josta kutsuttava web-metodi web-palvelussa löytyy. Lisäksi käytetystä XML-nimiavaruudesta riveillä 1 ja 18 selviää, että Xcityyn lähetettävä viesti muodostetaan Apachen CXF-viestintäkehystä käyttäen.

Edellä mainittujen asetusten kirjoittamisen jälkeen integraatiomoduuli ajettiin ylös ja yhteyttä testattiin käyttöliittymätasolle kirjoitetun JUnit-testin avulla. Integraatiorajapinta konvertoi olion haluttuun muotoon ja lähetti sen oikein serialisoituna Xcity-palvelulle, joka palautti samanlaisen vastauksen kuin sitä aiemmin SoapUI:n kautta testatessa. Vastaus deserialisoitiin onnistuneesti ja konvertoitiin ulkoisesta kuljetusoliomuodosta sisäiseen muotoon ennen sen palauttamista kutsun muodostaneeseen JUnit-testitapaukseen. Näin ollen integraatio koulupalveluun havaittiin kaikilla eri protokollatasoilla onnistuneeksi. Kuten voidaan päätellä, järjestelmäintegraation toteutus Kuntalaistilijärjestelmästä Xcityn koulupalveluun ei vaatinut erityisen suurta konfigurointia. CXF-viestintäkehys osaa huolehtia HTTP-protokollan ja SOAP-viestien oikeanlaisesta muodostamisesta sekä kuljetusolioiden serialisoinnista ja deserialisoinnista Java-olioista XML:ksi ja takaisin.

WCF-tyyppinen henkilöpalvelu osoittautui kaikkine WS-protokollineen ASMX-tyyppistä koulupalvelua huomattavasti vaikeammaksi tapaukseksi. Henkilöpalvelua kutsuvaa integraatiorajapintaa lähdettiin kehittämään samalla tavalla kuin edellä esiteltyä koulupalvelun integraatiorajapintaa, mutta alustavasti yhteyttä JUnit-testitapauksen avulla testattaessa Mule tai CXF tuntui jäätyvän deadlockiin Xcityn vastausta odotellessa. Koska ongelma ei vaikuttanut olevan reitityskonfiguraatioissa tai koodilistauksessa, siirryttiin tarkastelemaan Kuntalaistilijärjestelmän ja Xcityn välistä tietoliikennettä TCP Monitor -ohjelmaa käyttäen. Analyysin pohjalta tietoliikenteen nähtiin pysähtyvän WSDL-tiedostojen haun ym. alkumuodollisuuksien jälkeen itse palvelupyyntöön, joka

esitetään alla olevassa listauksessa Listaus 22. Xcity ei palauttanut listaukseen minkäänlaista vastausta.

```
1  POST /TeklaXcityPersonWebServiceTest/Person.svc HTTP/1.2
2  Content-Type: application/soap+xml;
   action="http://tekla.com/webservices/Person/GetPersonAdditionalIn
   foByIdentityCode"
3  X-MULE_CORRELATION_ID: d5d0d661-191e-11de-8013-0d22843a67d8
4  X-MULE_ENCODING: UTF-8
5  X-MULE_ENDPOINT:
   http://localhost:9876/TeklaXcityPersonWebServiceTest/Person.svc
6  X-MULE_ORIGINATING_ENDPOINT:
   endpoint.vm.TeklaXcityPersonService.GetPersonAdditionalInfoByIden
   tityCode
7  X-MULE_REMOTE_SYNC: true
8  X-MULE_SESSION:
   SUQ9ZDVkMWMwYzItMTkxZS0xMWRlLTgwMTMtMGQyMjg0M2E2N2Q4O0lEPWQ1ZDFjM
   GMyLTE5MWUtMTFkZS04MDEzLTBkMjI4NDNhNjdkOA==
9  User-Agent: Jakarta Commons-HttpClient/3.1
10 Host: teky01:9876
11 Transfer-Encoding: chunked
12
13 2ee
14 <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-
   envelope"><soap:Header><mule:header
   xmlns:mule="http://www.muleumo.org/providers/soap/1.0"><mule:MULE
   _CORRELATION_ID>d5d0d661-191e-11de-8013-
   0d22843a67d8</mule:MULE_CORRELATION_ID><mule:MULE_CORRELATION_GRO
   UP_SIZE>-
   1</mule:MULE_CORRELATION_GROUP_SIZE><mule:MULE_CORRELATION_SEQUEN
   CE>-
   1</mule:MULE_CORRELATION_SEQUENCE></mule:header></soap:Header><so
   ap:Body><GetPersonAdditionalInfoByIdentityCode
   xmlns="http://tekla.com/webservices/"
   xmlns:ns2="http://schemas.datacontract.org/2004/07/TeklaXcityWebS
   ervice"
   xmlns:ns3="http://schemas.microsoft.com/2003/10/Serialization/"><
```

```
   Key>Avain</Key><IdentityCode>ID-
   Koodi</IdentityCode></GetPersonAdditionalInfoByIdentityCode></soa
   p:Body></soap:Envelope>
15 0
```

Listaus 22: Mulen Xcityn henkilöpalvelulle lähettämä viesti, johon ei palauteta minkäänlaista vastausta

Luultavasti ensimmäinen asia, johon listauksessa Listaus 22 tulee kiinnittäneeksi huomiota, on Mulen lisäämien otsikkokenttien määrä niin HTTP-pyynnössä kuin SOAP-viestissäkin. Vaikka niiden ei valinnaisina kenttinä pitäisi häiritä viestin tulkitsemista palvelinpäässä, poistettiin ne viestistä ja yritettiin sen uudelleenlähettämistä suoraan TCP Monitor -ohjelmasta. Tällainenkaan kysely ei palauttanut mitään vastausta, vaikka on täysin HTTP 1.2 -RFC:n (Request For Comments) [W3C2003] mukainen. Yrityksen ja erehdyksen kautta lopulta saatiin vastaus muuttamalla HTTP-otsikon rivi 2 muodosta `Content-Type: application/soap+xml;action="http://tekla.com/webservices/Person/GetPersonAdditionalInfoByIdentityCode"` muotoon `Content-Type: application/soap+xml; charset="utf-8"`, eli poistamalla valinnaisena parametrina käsiteltävä viittaus kutsuttavaan metodiin ja lisäämällä maininta käytettävästä merkistöenkoodauksesta. Vain jommankumman muutoksen toteutus ei riittänyt!

Koska CXF on viestintäkehys, sen käyttämisen myötä ohjelmistokehityksessä tapahtuu ns. vastuunkääntö eli IOC (Inversion of Control). IOC:n myötä ohjelmistokehittäjän mahdollisuudet vaikuttaa joihinkin ohjelman toiminnallisiin yksityiskohtiin pienenevät, sillä koko ajatus kehysten takana on, että nämä useimmiten epäoleelliset seikat voidaan jättää kehysten hoidettaviksi. CXF:n tapauksessa IOC vaikuttaa mm. HTTP-otsikoiden rakentamiseen. Viestintäkehys muodostaa automaattisesti HTTP-otsikon, ja ohjelmistokehittäjälle annetaan vain rajalliset mahdollisuudet vaikuttaa siihen. Tämä vaikeuttaa edellä kuvatun kaltaisten ongelmien ratkomista. Periaatteessa CXF tarjoaa HTTP-otsikoiden muokkausmahdollisuuden Mulen `<message-properties-transformer>`-tyyppisen ns. muuntajan kautta, mutta tuki sille on CXF:ssa käytännössä

olematon. Ongelma saatiin kuitenkin kierrettyä laatimalla integraatiopalveluväylään HTTP-muotoinen päätepiste, johon CXF:stä otetaan yhteyttä ennen viestin välittämistä Xcity-järjestelmään.

HTTP-päätepisteissä HTTP-otsikoiden muokkaus toimii kuten pitääkin. Alla olevasta listauksesta (Listaus 23) nähdään, kuinka yksinkertaisesti tällainen HTTP-otsikkoa muokkaava päätepiste voidaan konfiguroida Mule-sovelluskehyksessä. Listauksen rivillä yksi määritellään varsinainen päätepiste ja viittaus `httpHeaderTransformer`-nimiseen muuntajaan. Muuntaja esitellään riveillä 2-4. Rivillä kolme annetaan HTTP-otsikkoon lisättäväksi `Content-Type`-avain-arvopari. Rivillä kaksi määritetään muuntaja ylikirjoittamaan otsikossa valmiiksi tätä avainta vastaava avain-arvopari. Reititys CXF:sta HTTP-päätepisteeseen kuvataan listauksessa Listaus 24. Tämän lisäksi listauksen Listaus 21 rivit 4 ja 6 muokattiin osoittamaan tähän uuteen reititykseen, virtuaalikoneosoitteeseen `vm://PersonService-Out`. Reitittämällä kaikki CXF:n päätepisteestä Xcity-järjestelmään lähtevät viestit yo. HTTP-päätepisteen kautta, saatiin aikaiseksi automaattinen HTTP-otsikoiden muunto oikeaan muotoon.

```
1  <endpoint name="Http-Out" synchronous="true" encoding="utf-8"
   transformer-refs="httpHeaderTransformer" />
2  <message-properties-transformer name="httpHeaderTransformer"
   overwrite="true">
3      <add-message-property key="Content-Type"
    value="application/soap+xml; charset=utf-8" />
4  </message-properties-transformer>
```

Listaus 23: HTTP-päätepisteen ”Http-Out” ja muuntajan esittely

```
1  <service name="PersonService-Out">
2   <inbound>
3      <vm:inbound-endpoint address="vm://PersonService-Out"/>
4    </inbound>
5    <outbound>
6      <multicasting-router>
```

```
7           <outbound-endpoint ref="Http-Out"
    address="http://teky01/TeklaXcityPersonWebServiceTest/Person.svc"
    />
8       </multicasting-router>
9    </outbound>
10 </service>
```

Listaus 24: Reititys päätepisteeseen ”Http-Out”

7.6 Kuljetuskerroksen salaus

Laatimalla HTTP-päätepiste ja muuntaja, saatiin Kuntalaistilijärjestelmän integraatioväylältä viimein luotua `teky01`-palvelimen Xcity-järjestelmään yhteys, joka toimii samoin, kuin SoapUI:n kautta testatessa. Koska Xcityn henkilöpalvelussa kuljetetaan henkilötietoja, viestiliikenne on lopulta kuitenkin suojattava. Xcity tarjoaa kaksinkertaista suojausta: Ensin tieto salataan web-palvelutasolla eli OSI-mallin sovelluskerroksella WS-Security-protokollan avulla. Kuten aiemmin SoapUI-yhteyskokeilun paluuviestistä nähdään, palvelu palauttaa SOAP-muotoisen virheviestin, mikäli palvelupyyntöä ei ole muodostettu WS-Securityn kannalta oikein. Sovelluskerroksen suojaukset kuitenkin suojaavat vain viestin hyötykuorman. Jotta myös viestien reititystiedot saadaan salattua, tarvitaan kuljetuskerroksen salausta, mikä käytännössä tarkoittaa SSL-suojatun (Secure Sockets Layer) tai TLS-suojatun (Transport Layer Security) HTTPS-protokollan (HTTP over SSL/TLS) käyttöä.

Kuljetuskerroksen salaus HTTPS-protokollaa käyttäen suojaa tietoliikenteen salakuuntelijoita vastaan, mutta ongelmaksi jää vastapuolen luotettava tunnistaminen. Tunnistamista varten tunnistettava pää esittää ITU-T X.509 -suosituksessa [ITU-T2005] (International Telecommunication Union – Telecommunication Standardization Section) määritellyn sertifikaatin vastapäälle. Tunnistaminen voi olla joko yksisuuntaista (asiakas tunnistaa isännän), tai kaksisuuntaista (asiakas tunnistaa isännän ja isäntä asiakkaan). Tuotantoympäristössä sertifikaatit ovat yleensä jonkin kolmannen, luotettavan tahon, kuten VeriSign [VeriSign2009] tai Thawte [Thawte2009], allekirjoittamia, mutta joissakin rajatuissa tilanteissa, kuten testikäytössä, niiden sijaan voidaan

käyttää itse allekirjoitettuja sertifikaatteja. Xcity-yhteyden kehityksen yhteydessä käytettiin sertifikaatteja vain yksisuuntaiseen tunnistamiseen ja isännän sertifikaatti oli itse allekirjoitettu. Asiakkaan tunnistaminen, tai tarkkaan ottaen valtuuttaminen, toteutettiin käyttäjätunnuksen perusteella sovelluskerroksella.

Kuntalaistilijärjestelmän integraatioväylän sovelluskehyksenä käytetty Mule edellyttää, että isännän itse allekirjoittamat sertifikaatit tallennetaan asiakkaan omaan avainvarastoon (keystore). Menetelmällä varmistutaan siitä, ettei vahingossa oteta yhteyttä isäntään, jonka identiteetistä ei ole luotettavan kolmannen tahon antamia takeita. `Teky01`-palvelimen sertifikaatti haettiin palvelimelta tavallista www-selainta käyttäen ja tallennettiin omaan avainvarastoonsa käyttäen Javan mukana tulevaa Keytool-komentorivityökalua. Mulen konfiguraatiotiedostoon lisättiin jälleen uusi päätepiste, tällä kertaa HTTPS-yhteyttä varten. Päätepisteen määrittely esitetään listauksessa Listaus 25.

Listauksen Listaus 25 rivillä yksi määritellään päätepisteen nimi, kutsuttavan isännän nimi ja viittaus HTTPS-yhdistimeen, joka määritellään riviltä kaksi alkaen. Rivillä kolme määritellään avainvarasto, josta asiakkaan oma sertifikaatti haettaisiin, jos sellaista kysyttäisiin. Rivillä neljä määritellään avainvarasto, josta haetaan socket-yhteyden luontiin tarvittava salasana. Rivillä viisi määritellään varasto, jossa säilytetään luotetun isännän sertifikaattia. Koska asiakaspäätä ei tunnisteta sertifikaatilla, rivin kolme voisi jättää pois listauksesta, mutta sen mukanaolo lisää palvelun joustavuutta myöhemmin, jos kaksisuuntainen tunnistaminen otetaan käyttöön. Päätepisteen reitityskonfiguraatio on vastaava kuin HTTP-päätepisteellä (Listaus 24). Lopuksi, ennen HTTPS-yhteyden käyttöönottoa piti vielä muokata HTTP-päätepisteen reitityskonfiguraatiot ohjaamaan HTTP-päätepisteeltä lähtevät viestit HTTPS-päätepisteelle, mutta muuten kuljetuskerroksen suojaus saatiin toimimaan näillä toimenpiteillä.

```
1  <https:endpoint name="Https-Out" synchronous="true" host="teky01
   " port="443" connector-ref="PersonService-HttpsConnector" />
2  <https:connector name="PersonService-HttpsConnector">
```

```
3   <https:tls-client path="\teky01\keys\keystore"
    storePassword="salasana" />
4   <https:tls-key-store path="\teky01\keys\keystore"
    keyPassword="salasana" storePassword="salasana" />
5   <https:tls-server path="\teky01\keys\keystore"
    storePassword="salasana" />
6   </https:connector>
```

Listaus 25: HTTPS-päätepisteen ”Https-Out” määritys avainvarastoineen

HTTPS-yhteyden käyttöönoton jälkeen Kuntalaistilijärjestelmän ja Xcityn välisen tietoliikenteen seuraaminen TCP Monitor -ohjelmalla muuttui tietenkin mahdottomaksi. Koska itse HTTPS-yhteys kuitenkin saatiin toimimaan, ei HTTPS-välin kuunteleminen enää tässä vaiheessa silti välttämätöntä ollutkaan. TCP Monitor asetettiin kuuntelemaan tiettyä porttia paikallisessa päässä (`localhost`) ja ohjaamaan siihen tulleet viestit toiseen paikallisessa päässä olevaan porttiin. Mule-reititykset HTTP-päätepisteeltä HTTPS-päätepisteelle ohjattiin kulkemaan näiden porttien kautta, ja näin yhteyttä voitiin jälleen seurata palvelun jatkokehityksen kannalta riittävässä määrin, eli juuri ennen kuljetuskerroksen salausta.

7.7 Asiakkaan auktorisointi sovelluskerroksella

Sertifikaatteja käyttäen saatiin toteutettua yhteyden salaus ja palvelimen tunnistaminen, mutta tämä jätti jäljelle vielä asiakkaan auktorisoinnin (valtuuttamisen) yhteyden muodostusvaiheessa. Asiakkaan valtuuttaminen toteutettiin käyttäen niin sanottua HTTP Basic -auktorisointia [RFC2617], joka perustuu käyttäjätunnukseen ja salasanaan. HTTP sijoittuu loogisesti TCP/IP-pinon ylimmälle eli sovellustasolle, joten HTTP:n avulla tapahtuva tunnistaminen ja valtuuttaminen ovat näin ollen sovelluskerroksen toimenpiteitä. Basic-auktorisoinnissa käyttäjätunnus ja salasana liitetään osaksi asiakaspäästä lähtevän kutsun HTTP-otsikkoa siten, että auktorisointitapa määritellään selkokielisenä, kaksoispisteellä erotetun käyttäjätunnus-salasanaparin seuratessa perässä Base64-enkoodattuna [RFC1521] merkkijonona. Esimerkiksi tunnuksilla `käyttäjänimi` ja `salasana` saatava otsikon avain-arvopari esitetään muodossa `Authorization: Basic a+R5dHTkauRuaW1pOnNhbGFzYW5h` . Väärillä tunnuksilla palvelinpää

palauttaa viestin, jossa ilmoitetaan auktorisointivirheestä HTTP-otsikossa ja sitä seuraavassa html-muotoisessa (Hypertext Markup Language) virheviestissä alla olevan listauksen (Listaus 26) mukaisesti.

```
1  X-Powered-By       ASP.NET
2  Content-Length     1539
3  #status#           HTTP/1.1 401 Unauthorized
4  Date               Wed, 29 Apr 2009 08:22:17 GMT
5  Server             Microsoft-IIS/6.0
6  Content-Type       text/html
7  WWW-Authenticate   Basic realm="teky01"
8  ...
9  <h1>You are not authorized to view this page</h1>
10 You do not have permission to view this directory or page using
   the credentials that you supplied.
11 ...
```

Listaus 26: Palvelimen palauttama ilmoitus valtuutusvirheestä

Palvelun testaaminen väärillä tunnuksillakin on hyödyllistä, sillä näin voidaan helposti nähdä ensinnäkin, että palvelu ymmärtää sille syötetyn auktorisointiavaimen, ja toisekseen, että palvelu myös tarkastaa sille syötetyn käyttäjätunnuksen oikeellisuuden. Näin voidaan varmistua siitä, ettei SSL-putken sisällä tapahdu aina false positive -hyväksyntää. Negatiivisen, eli väärillä tunnuksilla suoritetun testin jälkeen palvelua voidaan ruveta jatkotestaamaan positiivisin testein, eli oikeilla tunnuksilla.

Basic-autentikoinnissa on eräs ilmeinen huono puoli: Base64-enkoodaus ei salaa syötettyä käyttäjätunnusta, vaan toimii pelkästään merkistöongelmien torjumiseen. Näin ollen basic-autentikointia voidaan turvallisesti käyttää vain SSL-, TLS-, tai vastaavan kuljetustason suojauksen kanssa. Myös Mule-CXF-yhdistelmässä käyttäjänimi ja salasana esitetään tavalla, jota voidaan hyvinkin pitää huonona käytäntönä. Ongelma ilmenee listauksesta Listaus 27:

```
1  <service name="PersonService-HttpsOut">
2   <inbound>
```

```
3       <inbound-endpoint address="vm://PersonService-HttpsOut"
    synchronous="true" />
4    </inbound>
5    <outbound>
6       <pass-through-router>
7          <outbound-endpoint ref="Https-Out"
    address="https://kayttajanimi:salasana@teky01/TeklaXcityPersonWeb
    ServiceTest/Person.svc" />
8       </pass-through-router>
9    </outbound>
10 </service>
```

Listaus 27: Mule 2.2 -palvelualustalla HTTPS-yhteyden Basic-auktorisointi toteutetaan kirjoittamalla käyttäjänimi ja salasana kutsuttavaan osoitteeseen

Mule-palveluväylässä HTTP Basic -auktorisointi konfiguroidaan yllä olevan listauksen mukaisesti. Kuten riviltä 7 huomataan, palvelukutsun yhteydessä käytettävä käyttäjänimi ja salasana ilmoitetaan osoiterivin alussa. Tällaisessa käytännössä on ainakin kaksi ongelmaa. Toteutetun integraation kannalta epäoleellinen ongelma on arvojen kovakoodaus. Jossakin toisessa yhteydessä voisi olla tärkeää vaihtaa käyttäjänimeä ja salasanaa kutsutilanteen mukaan. Oleellisempi ongelma on sen sijaan se, että arvot esitetään selkokielisenä tekstinä XML-tiedoston sisällä. Toisin sanoen, kuka tahansa, joka pääsee käsiksi XML-muotoiseen konfiguraatiotiedostoon, näkee myös käytetyt tunnukset.

Vaikka Mule ja sen konfiguraatiotiedostot sijaitsevatkin suojatulla palvelimella, jolle vain järjestelmän ylläpitäjillä tulisi olla pääsymahdollisuus, selkokielinen salasanojen säilytys XML-tiedostossa on konfigurointitapana epäilyttävä. Konfiguroinnin kannalta edelleen melko helppo, mutta tietoturvan kannalta parempi ratkaisu olisi vähintäänkin kutsua jotakin Java-luokkaa, johon tunnukset on kirjattu. Luokan kääntämisessä tavukielelle voitaisiin tämän jälkeen käyttää monimutkaistamisasetusta (obfuscation), jolloin tunnusten selvittämiseen tulisi edes hiukan vaikeusastetta, vaikkei menetelmä taitavampia tietokoneenkäyttäjiä pysäyttäisikään. Mule hyödyntääkin tällaista menetelmää joissakin muissa

auktorisointitilanteissa, joten toivottavasti ominaisuus esitellään tulevaisuudessa myös Basic-auktorisointia varten.

7.8 Web-palvelutason reititys

SOA-paradigman mukaan sovellukset tulee jakaa mahdollisimman pieniksi komponenteiksi, jotka huolehtivat aina vain yhdestä tehtävästä kerrallaan. Tämän vuoksi monimutkaiset palvelupyynnöt ja palveluvastaukset voivat joutua kulkemaan usean eri komponentin kautta, ennen kuin kaikki palvelupyynnön suorittamiseksi vaaditut osatehtävät saadaan suoritettua. Poikkeuksellisissa virhetilanteissa kaikki viestit voidaan haluta ohjata tietylle, virheenkäsittelyyn erikoistuneelle komponentille. WS-Addressing on protokolla, joka vastaa palvelupyyntöjen ja vastauksien oikeanlaisesta reitityksestä eri päätepisteiden kesken [Erl2005]. Se ei ole pakollinen ominaisuus toimivia web-palveluita luotaessa, mutta Xcity-palveluissa sitä käytetään. Tämän vuoksi pyynnöt, joissa ei ole määritelty WS-Addressing-ohjausdataa, palauttavat HTTPS-protokollan yli Xcityltä listauksen Listaus 12 mukaisen virheviestin, joka esitetään uudelleen myös alla.

```
11 HTTP/1.1 500 Internal Server Error
12 ...
13 <s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
   xmlns:a="http://www.w3.org/2005/08/addressing">
14 …
15 <s:Fault>
16  …
17      <s:Value>a:DestinationUnreachable</s:Value>
18  …
19      <s:Reason><s:Text xml:lang="fi-FI">The message with To ''
   cannot be processed at the receiver, due to an AddressFilter
        mismatch at the EndpointDispatcher.  Check that the sender
   and receiver's EndpointAddresses agree.</s:Text></s:Reason>
20  </s:Fault></s:Body>
21 </s:Envelope>
```

Listaus 28: HTTPS-yhteyden käyttöönoton jälkeen palvelu ilmoittaa WS-Addressing-protokollan käyttötarpeesta

Jotta WS-Addressing saatiin käyttöön asiakaspäässä, palvelua kutsuvan Mule-palveluväylän CXF-viestintäkehykseen tehtiin oma, eri WS-protokollista huolehtiva protokollayhdistimensä. Protokollayhdistimen kautta WS-Addressingin käyttöönotto onnistuu varsin suoraviivaisesti. Alla näytetään, kuinka varsinainen protokollayhdistin määritellään (Listaus 29) ja kuinka se liitetään osaksi aiemmin, listauksen Listaus 21 riveillä 1-10 määriteltyä päätepistettä (Listaus 30).

Kuten jo listauksessa Listaus 21 määriteltiin, Xcityn henkilöpalveluita haetaan WSDL-portista ”WSHttpBinding_Person”. Tämä nähdään myös listauksen Listaus 30 rivillä 4. Rivit 5 ja 7 sisältävät viittauksen päätepisteessä käytettävään protokollayhdistimeen. CXF-protokollayhdistimessä esitellyt määrittelyt ovat porttikohtaisia, mikä onkin loogista, sillä näin eri porteista löytyviä palveluja voidaan käyttää eri parametrein, aivan kuten niitä voidaan tarjotakin. Porttikohtaisuudesta johtuen tiettyyn nimiavaruuteen liittyvä portti täytyy tietysti näin ollen esitellä myös protokollayhdistimen määrittelyssä, kuten listauksen Listaus 29 rivillä 1 tehdäänkin. WS-Addressing-protokollan käyttöönotto kyseiselle portille onnistuu protokollayhdistimessä yhdellä XML-rivillä. CXF-kehystä käytettäessä muuta ei tarvita web-palvelutason reitityksen käyttöönottoon.

```
1  <jaxws:client
   name="{http://tekla.com/webservices/}WSHttpBinding_Person"
   createdFromAPI="true">
2   <jaxws:features>
3       <wsa:addressing usingAddressingAdvisory="true"/>
4   </jaxws:features>
5  </jaxws:client>
```

Listaus 29: CXF-protokollayhdistimen määrittely ja WS-Addressing-protokollan käyttöönotto

```
1  <cxf:endpoint
2    name="PersonService-CxfOut"
3    …
4    wsdlPort="WSHttpBinding_Person"
```

```
5  protocolConnector="PersonService-CxfConnector"
6    …  />
7  <cxf:connector name="PersonService-CxfConnector"
   configurationLocation="conf/personservice-cxfconnector.xml"/>
```

Listaus 30: CXF-protokollayhdistimen liittäminen päätepistekonfiguraatioon

7.9 Web-palvelutason salaus

Aiempana on esitetty, kuinka Xcity-integraation kuljetuskerroksen salaus ja päätepisteiden autentikointi tai auktorisointi toteutetaan. TCP/IP-mallin kannalta sekä HTTP että sen päälle rakentuvat WS-protokollat kuuluvat kaikki sovelluskerrokselle. Niitä ei kuitenkaan voida pitää täysin samantasoisina protokollina, vaan sovelluskerros itsessään on käsitteiden selkiyttämiseksi syytä jakaa (tapauskohtaisesti) esimerkiksi HTTP-tasoon ja erilaisiin WS-protokollatasoihin. Kuljetuskerroksen salauksen ja HTTP-kerroksen auktorisoinnin lisäksi Xcityn henkilöpalvelut voivat käyttää myös WS-protokollatason salausta. Tällä tasolla Kuntalaistilijärjestelmän ja Xcityn väliset viestit salataan käyttäen WS-Security-protokollaa [OASIS2004].

Tuki WS-Securitylle CXF-viestintäkehyksessä on tätä kirjoitettaessa vielä melko alustavalla tasolla, mutta sille on kuitenkin jonkinlainen tuki. Ennen kuin CXF-viestintäkehyksessä (versio 2.1.2) voidaan ottaa WS-Security käyttöön, on Mule 2.2 -JCA-yhdistimen kirjastoon lisättävä paketit `cxf-rt-ws-security-2.1.2.jar` ja `cxf-rt-ws-policy-2.1.2.jar`, jotka löytyvät esimerkiksi MVN (Apache Maven) Browser –palvelun CXF-osiosta [MVNBrowser2009]. Parempi tuki WS-Securitylle ja WS-Policylle on luvattu CXF:n versioon 2.2 [CXF2009], mutta kannattaa ottaa huomioon, että protokollat itsessäänkin olivat vielä muutama vuosi sitten kehitysasteella (mm. [Newcomer2005]), eikä niitä välttämättä kannata täysin kypsinä pitää vieläkään. Kun paketit on asennettu, saadaan CXF:n policy- ja security-nimiavaruudet käyttöön.

Toinen tarvittava toimenpide ennen WS-Securityn käyttöä on lisätä integraatiossa käytetyn projektin `META-INF`-kansioon tiedostot nimeltä `javax.xml.soap.MessageFactory`, `javax.xml.soap.MetaFactory`,

`javax.xml.soap.SOAPFactory` ja `javax.xml.ws.spi.Provider`. Tiedostot ovat tekstitiedostoja, jotka sisältävät viittauksia luokkiin, jotka toteuttavat tiedostojen nimiä vastaavien luokkien toimintoja. Toimenpide on melko omituinen, eikä siitä varoiteta CXF:n ohjeistuksessa, vaan se pitää selvittää askel kerrallaan seuraamalla CXF:n konsolin virhelokia käynnistysten yhteydessä. Ongelma vaikuttaa juontavan juurensa CXF:n ja JBoss-palvelimen yhteiseloon [Shaw2008]. Yllä listattujen toimenpiteiden avulla voidaan ruveta muokkaamaan web-palvelutason tietoliikenteen salausasetuksia.

WS-Security toimii vuorovaikutuksessa kahden muun erikseen päälle kytkettävän protokollan kanssa. Nämä kaksi protokollaa ovat WS-Policy [W3C2006] ja WS-Addressing [W3C2004b]. WS-Addressing huolehtii web-palvelutason suojauksessa mm. virheviestien reitittämisestä oikeaan paikkaan. Sen käyttöönotto esiteltiin kappaleessa 7.8. WS-Policy puolestaan määrittelee käytettävän tietoturvapolitiikan [Erl2005], siinä missä WS-Security keskittyy varsinaiseen salaukseen. Xcity-palveluiden WS-Policya ei käytetä muihin tehtäviin. WS-Policy otetaan käyttöön samalla tavalla kuin WS-Addressing, eli protokollayhdistimen kautta. Kun palvelun keskeneräisyydestä johtuen tarvittavat jar-paketit on asennettu ja muut vastaavanlaiset esityöt tehty, voidaan WS-Policy ottaa käyttöön yksinkertaisesti lisäämällä protokollayhdistimen määrityksiin yksi XML-rivi (Listaus 31):

```
1  <jaxws:features>
2   <wsa:addressing usingAddressingAdvisory="true"/>
3   <p:policies ignoreUnknownAssertions="true"/>
4  </jaxws:features>
```

Listaus 31: WS-Policy-protokollan käyttöönotto tapahtuu samoin kuin WS-Addressing-protokollan käyttöönotto

Kun kaikki alemmat protokollat WS-Addressing-protokollaan asti on saatu toimimaan, ennen web-palvelutason salauksen käyttöönottoa Xcity palauttaa palvelupyyntöihin virheviestin, jossa ilmoitetaan selvällä englannin kielellä salauksen puutteesta. Tämä virheviesti esitetään listauksessa Listaus 32. Viesti on

samanlainen riippumatta siitä, onko WS-Policy käytössä vai ei, mutta WS-Policyn käyttöönotto on tarpeellinen seuraavan askeleen ottamiseksi, eli WS-Securityn määrittelyä varten.

```
1  <s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
   xmlns:a="http://www.w3.org/2005/08/addressing">
2   …
3  <s:Fault>
4  …
5                 <s:Value
   xmlns:a="http://schemas.xmlsoap.org/ws/2005/02/sc">a:BadContextTo
   ken</s:Value>
6  …
7           <s:Reason><s:Text xml:lang="fi-FI">The security context
   token is expired or is not valid. The message was not
   processed.</s:Text><s:Reason>
8      </s:Fault></s:Body>
9  </s:Envelope>
```

Listaus 32: Palvelutason salauksen puutteesta ilmoittava virheviesti

WS-Security on Xcityn käyttämässä WS-protokollapinossa ylimmällä tasolla. Sen päälle ei rakenneta enää uusia protokollatasoja, vaan sovellukset keskustelevat WS-Security-protokollaa käyttäen keskenään. WS-Security voidaan CXF-kehyksessä määritellä joko yksisuuntaiseksi tai kaksisuuntaiseksi. Yksisuuntaisessa tapauksessa vain joko lähtevät tai saapuvat viestit salataan. Niin haluttaessa molemmat suunnat voidaan salata eri tavoin. Listaus 33 (alla) esittää, kuinka Kuntalaistilijärjestelmästä ulospäin suuntautuva salaus voidaan toteuttaa.

Listauksessa Listaus 33 määritellään ”kaappaaja” (interceptor), joka nimensä mukaisesti havaitsee kaikki protokollayhdistintä hyödyntävät, ulospäin lähtevät viestit, ja kaappaa ne käsiteltävikseen. Myös saapuvien viestien kaappaaminen onnistuu vastaavanlaisella kaappaajalla. Oleellisimmat rivit listauksessa ovat rivit 6-9, joilla määritellään käytetty salaustapa ja siihen liittyvät yksityiskohdat. Salaustavaksi valitaan rivillä 6 käyttäjänimi-salasanapari. Salaustavaksi voidaan valita myös NTLM, joka Microsoftin omana tekniikkana voisi hyvinkin olla

käytössä Teklan .NET-pohjaisissa Xcity-palveluissa. Itse käyttäjänimi esitetään rivillä 7. Rivillä 8 määritellään tapa, jolla salasana tulkitaan. Tässä tapauksessa salasanan tyyppi on ”`PasswordDigest`”, eli tunnusta ei kuljeteta selkokielisenä, vaan siitä lasketaan yksisuuntaisesti SHA-1-tyyppinen (Secure Hash Algorithm) hash-arvo, joka liitetään viestiin salasanan sijaan. Koska Xcityn tapauksessa voidaan olettaa, että WS-Security toimii SSL:n päällä, tyypiksi voitaisiin valita turvallisesti myös ”`PasswordText`”, joka lähettää salasanan selkokielisenä, mutta varma on aina varmaa.

Kuten listauksesta ilmenee, konfiguraatiossa ei missään kohtaa kuitenkaan selkokielisesti esitetä käytettävää salasanaa. Tämäkin olisi toki mahdollista, mutta CXF-viestikehys mahdollistaa salasanan määrittelyn Java-tiedostossa. Juuri näin allaolevassa listauksessakin tehdään: Riviltä 9 viitataan rivillä 15 määriteltyyn Java-luokkaan, jossa salasana kerrotaan. Järjestelyn ansiosta kuka tahansa XML-konfiguraatiota tuotantoympäristössä lukeva henkilö ei näe yhteyden käyttöön vaadittua salasanaa, vaan sen lukeminen vaatii pääsyä ohjelman käännettyyn tavukoodiin, jota huomattavasti harvempi osaa tai jaksaa ilman syytä lukea. Salasanaluokka esitellään listauksessa Listaus 34.

```
1  <jaxws:outInterceptors>
2  <bean class="org.apache.cxf.binding.soap.saaj.SAAJOutInterceptor"
   />
3  <bean
   class="org.apache.cxf.ws.security.wss4j.WSS4JOutInterceptor">
4  <constructor-arg>
5   <map>
6       <entry key="action" value="UsernameToken" />
7       <entry key="user" value="Kayttajanimi" />
8       <entry key="passwordType" value="PasswordDigest" />
9       <entry key="passwordCallbackRef" value-ref="clientCallback"
   />
10  </map>
11 </constructor-arg>
12 </bean>
13 </jaxws:outInterceptors>
```

```
14 </jaxws:client>
15 <bean id="clientCallback"
   class="com.propentus.psc.integration.xcity.util.ClientPasswordCal
   lback"/>
```

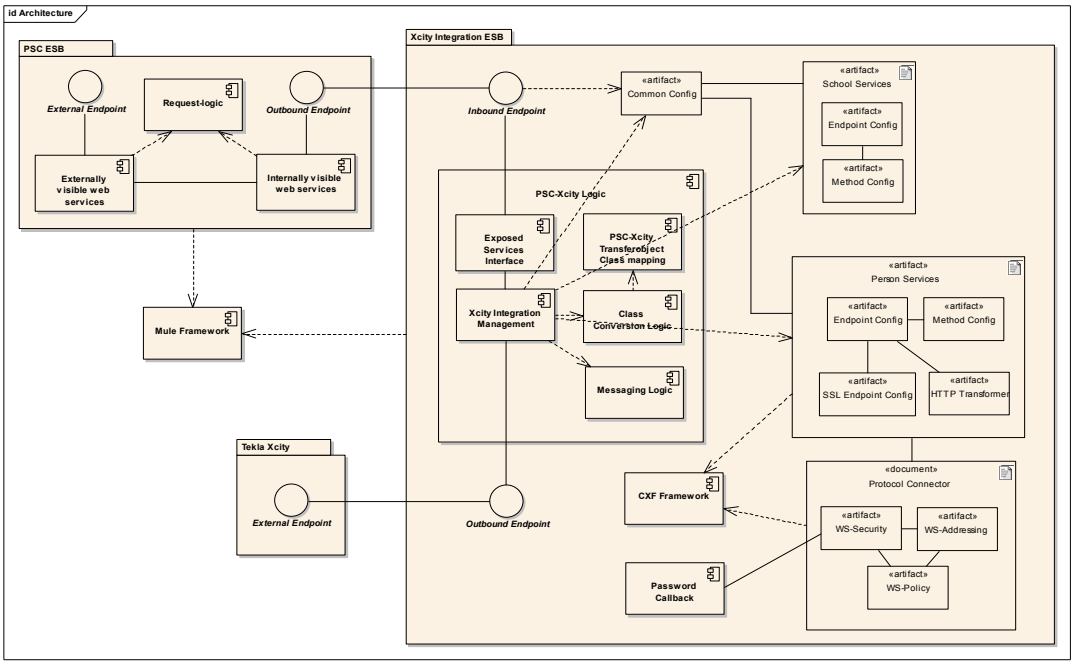
Listaus 33: Protokollayhdistimen määrittäminen käyttämään WS-Security-salausta ulospäin suuntautuvissa viesteissä

```
1  public class ClientPasswordCallback implements CallbackHandler {
2   public void handle(Callback[] callbacks) throws IOException,
3       UnsupportedCallbackException {
4       WSPasswordCallback pc = (WSPasswordCallback) callbacks[0];
5       pc.setPassword("Salasana");
6   }
7  }
```

Listaus 34: WS-Security-salausprotokollassa käytettävän salasanan sisältävä Java-luokka

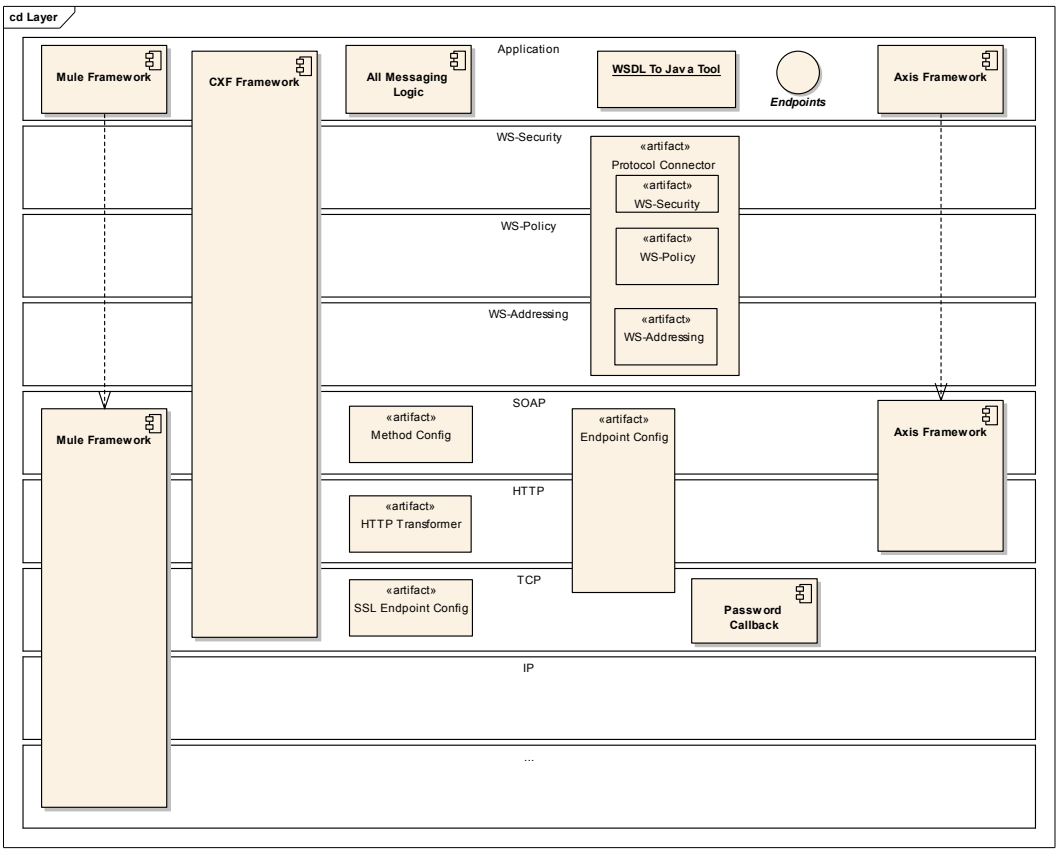
7.10 Kokonaiskuva integraatioarkkitehtuurista

Kuten edellä on kuvattu, integraatio Xcity-järjestelmään toteutettiin omana moduulinaan. Moduuli sisältää oman integraatiopalveluväylänsä, joka tarjoaa palveluita Kuntalaistilijärjestelmän pääpalveluväylälle federointiperiaatteella. Pääpalveluväylällä voitiin palveluiden kutsumiseen käyttää työssä [Mustonen2009a] toteutettua request-palvelupyyntömallia, mutta integraatiopalveluväylälle laadittiin oma ratkaisunsa. Kehitystyö tapahtui lähes kokonaan palveluväylätasolla. Palveluväylätason arkkitehtuurin oleelliset osat esitetään kuvan Kuva 8 arkkitehtuurikaaviossa.



Kuva 8: Xcity-integraatioon toteutettu palveluväylätason arkkitehtuuri

Kuvassa Kuva 8 kuvataan tarkimmin integraatiopalveluväylän osuus, joka on jaettu lisäksi komponentteihin ja artefakteihin. Komponenteilla kuvataan tässä yhteydessä erilaisia palvelulogiikkaan liittyviä Java-kielellä ohjelmoituja osuuksia, joita ovat esimerkiksi dynaamisesti toimiva kuljetusolioiden muuntaminen sisäisen ja ulkoisen esitystavan välillä. Artefakteilla kuvataan erilaisia XML-muotoisia osuuksia, jotka liittyvät palvelukonfiguraatioihin ja vaikuttavat pääsääntöisesti viestiliikenteessä käytettyihin protokolliin ja koodaukseen. Tällaisia osuuksia ovat esimerkiksi SSL-suojaus ja HTTP-muuntaja. Koska erityisesti artefaktit vaikuttavat useisiin eri protokolliin, niiden välisten suhteiden selvittämiseksi on ne jaoteltu eri protokollatasoille kuvassa Kuva 9.

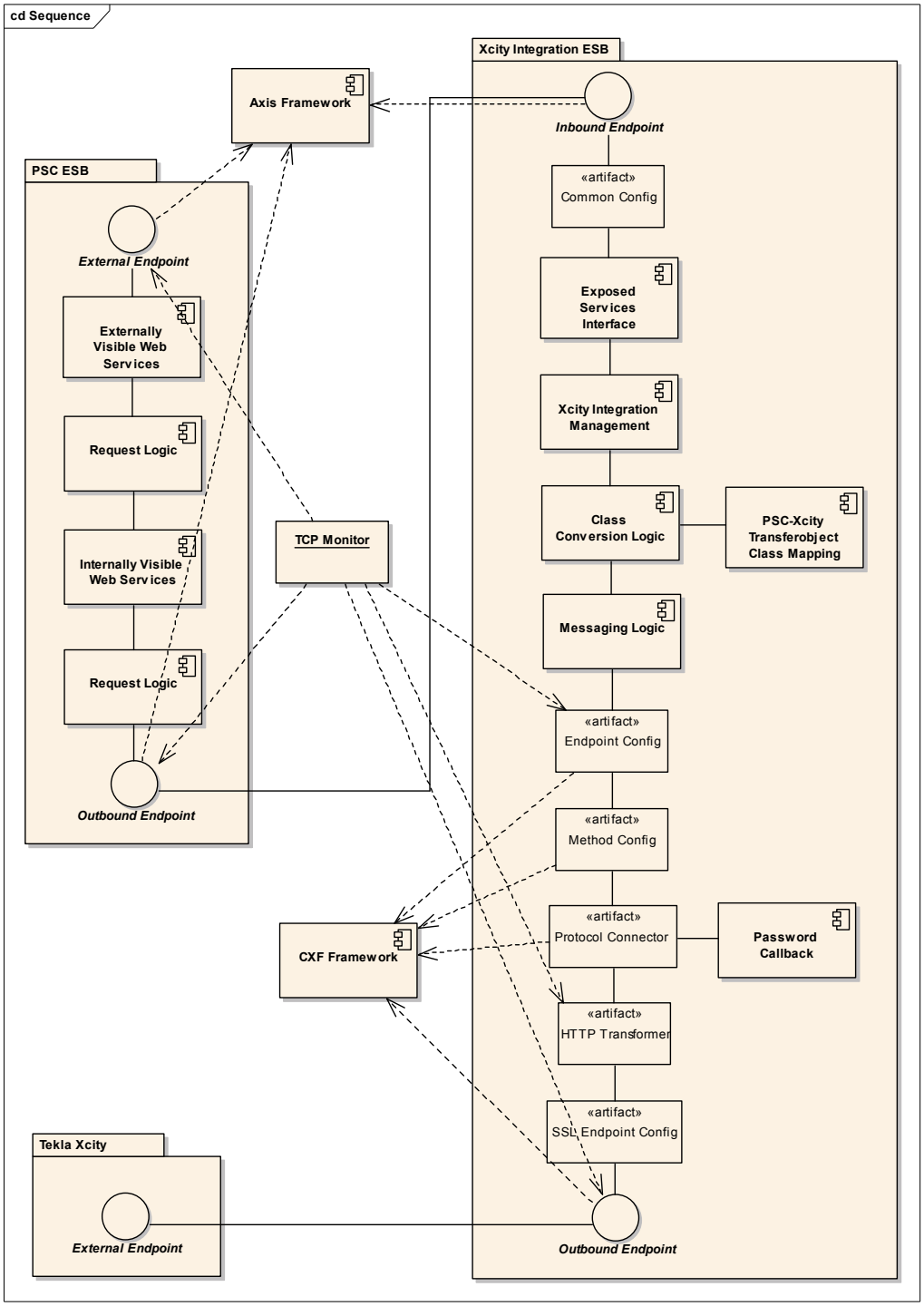


Kuva 9: Integraatiossa käytettyjen komponenttien vastuualueita protokollatasoittain eroteltuna

Kuvan Kuva 9 esitys protokollista on mukautettu varta vasten tätä työtä varten. TCP/IP-protokollapinon sovelluskerros on korvattu pinolla, joka sisältää työssä käytetyt WS-protokollat. Näiden suhteenkin on tosin huomioitava, että koska uusia WS-protokollia laaditaan jatkuvasti, eivätkä niiden vastuualuiden rajat ole selvästi määritettävissä, ei WS-protokollista ole olemassa vakiintunutta pinoesitystä. Esimerkiksi [Singh2005] ja [CBDI2009] tarjoavat keskenään varsin erilaiset tulkinnat WS-protokollapinosta. Kuvan Kuva 9 esitys onkin yksinkertaistus, joka toimii vain Xcity-integraation kontekstissa.

Koska kuvan Kuva 8 arkkitehtuurikaavio ei sellaisenaan anna selkeää kuvaa siitä, missä järjestyksessä mikäkin komponentti tai artefakti osallistuu viestinvälitykseen Kuntalaistilijärjestelmän pääpalveluväylältä Xcity-järjestelmälle, laadittiin arkkitehtuurikaaviosta vielä kuvan Kuva 10 mukainen vuokaavio. Vuokaaviosta nähdään järjestys, jossa eri osien välinen

viestinvälitysketju rakentuu. Lisäksi kuvasta ilmenevät kohdat, joista liikennettä TCP Monitorilla pääsääntöisesti analysoitiin, sekä viestintäkehys, jota kukin päätepiste tai artefakti hyödyntää.



Kuva 10: Vuokaavio Xcityn henkilöpalvelun kutsumiseen liittyvistä komponenteista ja pisteet, joista liikennettä seurattiin

8 JOHTOPÄÄTÖKSET

Viestiliikenne sekä uusia järjestelmiä toteutettaessa että järjestelmäintegraatioissa on mahdollista toteuttaa monella eri tavalla, mutta viime aikoina on tehty paljon tutkimusta ja tuotekehitystä viestiliikenteen yhdenmukaistamiseksi. Erilaiset XML-pohjaisia SOAP-viestejä välittävät web-palvelut ovat yhä suositumpia, ja niiden käyttö on mahdollista hyvin laaja-alaisesti. Teoriassa jopa 92 prosenttia kaikista sovelluksista tarjoaa jonkinlaisen valmiuden XML-pohjaisen viestiliikenteen käyttöönottoon. Kaikissa tilanteissa XML ei kuitenkaan mm. sen overheadin vuoksi ole järkevä vaihtoehto viestiliikenteeseen, sillä overhead pudottaa järjestelmien suorituskykyä ja aiheuttaa samalla kuormaa tietoliikenneverkkoihin. Tämän voi havaita myös kehitteillä olevassa Lahti Fenix Kuntalaistilijärjestelmässä, joka käyttää web-palveluita sekä sisäisissä että ulkoisissa järjestelmäintegraatioissa, kuten integraatiossa Teklan Xcity-järjestelmään. Palvelukeskeisen arkkitehtuurin ja palveluväyläreitityksen vuoksi web-palveluiden käyttö Kuntalaistilijärjestelmässä on kuitenkin perusteltua.

Tässä raportissa kuvattiin vaiheittain, kuinka Kuntalaistilijärjestelmään liitettiin ulkoinen Xcity-resurssi web-palveluita käyttäen. Järjestelmien välisessä integraatiossa keskeisenä osana oli Mule-palveluväylä. Mule-palveluväylästä käytössä oli kaksi eri versiota, mutta ne eivät aiheuttaneet keskinäisiä ristiriitoja. Vanhemman pääpalveluväylän palvelut ja viestiliikenne on toteutettu käyttäen Apache Axis 1.4 -viestiliikennekehystä, joka on ensimmäisiä laajalti käytettyjä Java-sovellusten yhteyteen suunniteltuja viestintäkehyksiä. Axis muodostaa oletusarvoisesti RPC/encoded-tyyppisiä SOAP-viestejä, joiden käyttö on melko vähäistä muissa viestiliikennekehyksissä. Axis ei myöskään tue WS-Security-, WS-Policy- eikä WS-Addressing-protokollia. Tämä on potentiaalinen haitta useissa tulevaisuudessa tapahtuvissa järjestelmäintegraatioissa, joten uusissa järjestelmissä kannattaa Axis-kehyksen sijaan harkita jonkin muun kehyksen käyttöä.

Kuntalaistilijärjestelmään liitetty Xcity on toteutettu .NET-tekniikalla, joten se tarjoaa WCF- ja ASMX-tyyppisiä web-palveluja. Näissä viestit serialisoidaan tyypillisesti document/literal wrapped -muotoisina, mikä onkin hyvä käytäntö. Axis tukee myös document/literal wrapped -muotoa SOAP-viesteissä, mutta käytännön testaus osoitti Axis-.NET-välin olevan hankalasti toteutettavissa. Tästä syystä integraatiota lähdettiin toteuttamaan uudella integraatiopalveluväylällä ja CXF-viestintäkehyksellä, joka tarjoaa Axista paremman yhteensopivuuden .NET-palveluiden kanssa ja tukee myös useita erilaisia WS-protokollia. CXF:n avulla ASMX-tyyppiseen Xcity-koulupalveluun olikin erittäin helppoa luoda yhteys, mutta toisaalta WCF-tyyppinen Xcity-henkilöpalvelu aiheutti ongelmia senkin edestä.

Mule-alustalla toimiakseen CXF vaatii vähintään Mule-version 2. Mule 2 on saatavilla sekä itsenäisesti pyörivänä sovelluksena, että muihin palvelimiin upotettavana ns. JCA-yhdistimenä. Näistä vaihtoehdoista tässä työssä otettiin käyttöön JCA-versio. Palveluväylän ja CXF-kehyksen asennus osoittautui muuten helpoksi, mutta CXF:n JBoss-riippuvuudet aiheuttivat suuria ongelmia. Mule-palvelualustasta kannattaa siis asentaa standalone-versio, ellei JBoss-riippuvaisen version asennukselle ole erityisen hyvää syytä.

Integraatiopalveluväylältä pääpalveluväylälle päin palvelut tarjottiin Axis 1.4 -muotoisina. Näin saavutettiin maksimaalinen yhteensopivuus Kuntalaistilijärjestelmän valmiiden komponenttien kanssa, mutta samalla voitiin käyttää uusimpia Mule-palveluväylän ja sille saatavissa olevien viestintäkehysten tukemia web-palveluprotokollia integraatioväylältä Xcityyn päin. Protokollakonversio onnistui palveluväylän avulla ilman pienintäkään ongelmaa.

Valitun arkkitehtuurin ansiosta integraatioihin liittyvä logiikka saatiin Kuntalaistilijärjestelmän sisäisestä toimintalogiikasta täysin eristettyyn moduuliin. Tiedon piilotus ja eristäminen helpottaa jatkossa järjestelmän rakenteen hahmottamista, auttaa rajapintamuutosten tekemisessä ja mahdollistaa mm. integraatiomoduulin poisjättämisen tai korvaamisen toisella moduulilla, mikäli

asiakas niin toivoo. Palveluväylän avulla tällaiset muutokset voidaan toteuttaa pelkillä konfiguraatiomuutoksilla, koskematta ohjelmakoodiin.

Palveluväylän ja viestintäkehysten käyttö vaatii oman opettelunsa, mikä edellyttää paitsi käytännön harjoittelua, myös riittävän hyvää referenssidokumentaatiota. Toteutushetkellä CXF-viestintäkehyksen ja Mule-palveluväylän version 2.2 dokumentointi oli vielä melko puutteellista ja epäjohdonmukaista. Dokumentaatiosta löytyneet harvat XML-konfiguraatiosta kertovat esimerkkilistaukset osoittautuivat usein toimimattomiksi, joten toimivien ja keskenään yhteensopivien parametrien selvittäminen oli hidasta, työlästä ja vaati useiden eri kombinaatioiden järjestelmällistä testaamista. Testaamisen lisäksi jokaisen muutoksen jälkeen oli tarpeen analysoida päätepisteiden välillä kulkevaa viestiliikennettä, sillä kehysten virheilmoitukset eivät yleensä antaneet riittävästi informaatiota ilmenneiden ongelmien ratkaisemiseksi. Analysoinnissa ja testauksessa käytetyt TCP Monitor ja SoapUI ovat ilmaisia ja helppokäyttöisiä ohjelmia, joita voi varauksetta suositella jokaiselle ohjelmistosuunnittelijalle, joka on tekemisissä web-palveluiden kanssa.

Web-palvelut, palveluväylä ja viestintäkehykset ovat kaikki viestiliikenteen parantamiseen tarkoitettuja keinoja. Valitettavasti yksikään esitetyistä tekniikoista tai ratkaisuista ei ole täysin yksiselitteisesti toimiva, vaan erilaisten protokollien tulkinnassa on toteutuskohtaisia eroja, jotka jossakin määrin vesittävät niiden alkuperäisen tarkoituksen. Järjestelmäintegraatiot ovat olleet työläitä aikaisemmin, ja ne ovat melko työläitä edelleen, markkinointihenkilöstön lupauksista huolimatta. Toisaalta tulee kuitenkin muistaa, että useat käytetyistä sovelluksista ja protokollista ovat edelleen kehitysasteella, ja suunta esimerkiksi Mule-palveluväylän eri versioissa on selvästi parempaan päin. Uusien tekniikoiden käyttöönotto on vaikeaa, mutta toimivat ratkaisut ovat usein toteutukseltaan lyhyitä ja siirrettävissä vastaisuudessa kehitettäviin palveluihin lähes sellaisenaan. Palveluväylää siis kannattaa hyödyntää järjestelmäintegraatioissa.

LÄHDELUETTELO

[Anderson2003] Anderson, James. Common Lisp Support for the Extensible Markup Language (CL-XML), 2003. [Ohjelmakirjasto] [Viitattu 2008-12-04] [Saatavana: http://pws.prserv.net/James.Anderson/XML/documentation/index.html]

[Apache2007] Apache Software Foundation. Xerces-C++ XML Parser, 2007 [Ohjelmakirjasto] [Viitattu 2008-12-04] [Saatavana: http://xerces.apache.org/xerces-c/]

[CBDI2009] CBDI Forum. Web Services Protocol Summary, 2009. [Verkkodokumentti] [Viitattu 2009-03-18] [Saatavana: http://roadmap.cbdiforum.com/reports/protocols/summary.php]

[CXF2009] Apache CXF, 2009. [Viestintäkehys] [Viitattu 2009-03-19] [Saatavana: http://cxf.apache.org/]

[DSource2008] Tuntematon julkaisija. SimpleXMLD, 2008. [Ohjelmakirjasto] [Viitattu 2008-12-05] [Saatavana: http://www.dsource.org/projects/simplexmld]

[ECMA2005] ECMA-357. ECMAScript for XML (E4X) Specification. ECMA International, 2005.

[Erl2005] Erl, Thomas. Service-Oriented Architecture – Concepts, Technology, and Design. Prentice Hall 2005. ISBN 0-13-185858-0

[Eviware2009] Eviware SoapUI, 2009. [Sovellus] [Viitattu 2009-03-20] [Saatavana: http://www.soapui.org]

[Expat2007] Tuntematon julkaisija. Expat, 2007. [Ohjelmakirjasto] [Viitattu 2008-12-05] [Saatavana: http://expat.sourceforge.net/]

[Heiss2007] Heiss, Janice J. Meet Peter von der Ahé, Tech Lead for Javac at Sun Microsystems [Haastattelu] [Viitattu 2009-03-20] [Saatavana: http://java.sun.com/developer/Meet-Eng/vonderahe/]

[IBM2003] Butek, Russell. Which style of WSDL should I use, 2003. [Verkkodokumentti] [Viitattu 2009-02-03] [Saatavana: http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/]

[Infosys] Infosys. SOA Plug-and-Play Services. Julkaisuajankohta tuntematon. [Verkkodokumentti] [Viitattu 2009-03-04] [Saatavana: http://www.infosys.com/finacle/pdf/thoughtpapers/SOA-Plug-and-Play.pdf]

[ITU-T2005] International Telecommunication Union – Telecommunication Standardization Section. Recommendation X.509 (08/05), 2005

[Juric2005] Juric et al. Comparison of performance of Web services, WS-Security, RMI, and RMI-SSL, 2005. ACM Journal of Systems and Software, Volume 79, Issue 5.

[Keller2004] Keller, Horst. The Official ABAP Reference. SAP Press, 2004.

[LuaExpat2007] Kepler Project. LuaExpat, 2007. [Ohjelmakirjasto] [Viitattu 2008-12-05] [Saatavana: http://www.keplerproject.org/luaexpat/]

[Markus2008] Markus Arjen. XML-Fortran, 2008. [Ohjelmakirjasto] [Viitattu 2008-12-04] [Saatavana: http://xml-fortran.sourceforge.net/]

[McLean2008] McLean, Grant. Perl XML, 2008. [ohjelmakirjasto] [Viitattu 2008-12-05] [Saatavana: http://perl-xml.sourceforge.net/]

[Microsoft2003] Microsoft. Microsoft .NET Framework 1.1 Class Library Reference Volume 6: System.Xml and System.Data. Microsoft Press, 2003. ISBN 9780735618176.

[MSDN2009] Microsoft Developer Network. Microsoft NTLM, 2009. [Verkkodokumentti] [Viitattu 2009-03-18] [Saatavana: http://msdn.microsoft.com/en-us/library/aa378749.aspx]

[MuleSource2008] Mason, Ross. Axis SOAP Styles – Mule 1.x User Guide, 2008. [Käyttöohje] [Saatavana: http://www.mulesource.org/display/MULEUSER/Axis+SOAP+Styles]

[Mustonen2009a] Mustonen, Timo Aleksi. Matalakompleksisen palvelupyyntömallin kehittäminen palveluväylää hyödyntäviin järjestelmäintegraatioihin, 2009.

[MVNBrowser2009] MVN Browser, CXF repository, 2009 [Verkkopalvelu] [Viitattu 2009-04-09] [Saatavana: http://www.mvnbrowser.com/artifacts-browse.html?groupId=org.apache.cxf]

[Møller2006] Møller, Anders & Schwartzbach, Michael. An Introduction to XML and Web Technologies. Addison-Wesley, 2006. ISBN 0-321-26966-7

[Newcomer2005] Newcomer, Eric & Lomow, Greg. Understanding SOA with Web Services. Addison-Wesley 2005. ISBN 0-321-18086-0

[OASIS2004] Web Services Security: SOAP Message Security 1.0 (WS-Security 2004). OASIS Standard 200401, 2004.

[Oracle2008] Oracle. XML parser for PL/SQL, 2008. [Ohjelmakirjasto] [Viitattu 2008-12-05] [Saatavana: http://www.oracle.com/technology/tech/xml/index.html]

[Pascal2003] Tuntematon julkaisija. SAX for Pascal, 2003. [Ohjelmakirjasto] [Viitattu 2008-12-04] [Saatavana: http://saxforpascal.sourceforge.net/]

[PHPGroup2008] The PHP Group. SimpleXML, 2008. [Ohjelmakirjasto] [Viitattu 2008-12-05] [Saatavana: http://us2.php.net/simplexml]

[Propentus2008] Propentus Oy, Interest Group Management –järjestelmä. [Saatavana: https://impera.propentus.fi/igm/login.jsp] [Vaatii käyttäjätunnukset]

[PyXML2008] The Python XML Special Interest Group. PyXML, 2008. [Ohjelmakirjasto] [Viitattu 2008-12-05] [Saatavana: http://pyxml.sourceforge.net/topics/]

[Remec2008] Remec, Miha & Gabrijelčič Primož. OmniXML for Delphi, 2008. [Ohjelmakirjasto] [Viitattu 2008-12-05] [Saatavana: http://www.omnixml.com/]

[RFC1521] Borenstein, N. & Freed, N. MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies, 1993. The Internet Engineering Task Force (IETF). [Saatavana: http://www.ietf.org/rfc/rfc1521.txt]

[RFC2617] Luotonen et al. Http Authentication: Basic and Digest Access Authentication, 1999. The Internet Engineering Task Force (IETF). [Saatavana: http://www.ietf.org/rfc/rfc2617.txt]

[Robertson2007] Robertson, Ian. Reflecting generics, 2007. [Verkkodokumentti] [Viitattu 2009-03-23] [Saatavana: http://www.artima.com/weblogs/viewpost.jsp?thread=208860]

[SAS2008] SAS. Base SAS: XML LIBNAME Engine [Verkkodokumentti] [Viitattu 2008-12-05] [Saatavana: http://support.sas.com/rnd/base/xmlengine/index.html]

[Schmidt] Schmidt, Maik. XmlSimple. Julkaisuajankohta tuntematon. [Ohjelmakirjasto] [Viitattu 2008-12-05] [Saatavana: http://xml-simple.rubyforge.org/]

[ServiceMix2009] Apache ServiceMix, 2009. [Sovellus] [Viitattu 2009-03-18] [Saatavana: http://servicemix.apache.org/home.html]

[Shaw2008] Shaw, Steve. JBoss and CXF, 2008. Intelliware.ca [Verkkodokumentti] [Viitattu 2009-04-29] [Saatavana: http://i-proving.ca/space/Technologies/Apache+CXF/JBoss+and+CXF]

[Singh2005] Singh, M. & Huhns, M. Service-Oriented Computing – Semantics, Processes, Agents. Wiley, 2005. ISBN 978-0-470-09148-7.

[Sun1998] McCluskey, Ken. Using Java Reflection, 1998. [Verkkodokumentti] [Viitattu 2009-02-04] [Saatavana: http://java.sun.com/developer/technicalArticles/ALT/Reflection/index.html]

[Sun2004] Mahmoud, Qusay. Using and Programming Generics in J2SE 5.0, 2004. [Verkkodokumentti] [Viitattu 2009-02-04] [Saatavana: http://java.sun.com/developer/technicalArticles/J2SE/generics/]

[Sun2006] JSR222. JAXB 2.1 Runtime Library Final Release specification, 2006. Sun Microsystems.

[Tekla2009] Tekla Oy, 2009. [Verkkodokumentti] [Viitattu 2009-03-04] [Saatavana: http://www.tekla.com/fi/products/tekla-xcity/Pages/Default.aspx]

[Thawte2009] Thawte, Inc. [Verkkodokumentti] [Viitattu 2009-04-08] [Saatavana: www.thawte.com]

[TIOBE2008] TIOBE Software. Programming community index for 2008, 2008. [Verkkodokumentti] [Viitattu 2008-12-04] [Saatavana: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html]

[TR247162007] ISO/IEC TR 24716:2007. Native COBOL Syntax for XML Support, 2007. International Standardization Organization.

[VeriSign2009] VeriSign, Inc. [Verkkodokumentti] [Viitattu 2009-04-08] [Saatavana: www.verisign.com]

[W3C2003] World Wide Web Consortium (W3C). SOAP Version 1.2 Part 2: Adjuncts, 2003.

[W3C2004] World Wide Web Consortium (W3C). Web Services Glossary, 2004. [Verkkodokumentti] [Viitattu 2008-12-09] [Saatavana: http://www.w3.org/TR/ws-gloss/]

[W3C2004b] World Wide Web Consortium (W3C). Web Services Addressing (WS-Addressing), 2004.

[W3C2006] World Wide Web Consortium (W3C). Web Services Policy 1.2 Framework (WS-Policy), 2006.

[XFire2008] Apache XFire, 2008. [Viestintäkehys] [Viitattu 2009-03-19] [Saatavana: http://xfire.codehaus.org/]