PORTAALIEN JA SISÄLLÖNHALLINTAJÄRJESTELMIEN INTEGROINTI PALVELUKESKEISEN ARKKITEHTUURIN AVULLA

Jukka Hirvonen

4.3.2008

Joensuun yliopisto
Tietojenkäsittelytiede

Pro gradu -tutkielma

TIIVISTELMÄ

Palvelukeskeisen arkkitehtuurin avulla voidaan kehittää organisaation toimintaa, integroimalla prosesseja ja hyödyntämällä olemassa olevien tietojärjestelmien tietosisältöä. Tässä tutkimuksessa tutustutaan sisällönhallintajärjestelmiin, portaaleihin ja niiden integroimiseen palvelukeskeisen arkkitehtuurin avulla. Eräänä palvelukeskeisen arkkitehtuurin tavoitteena on standardinmukaisuus, jonka takia tutkimuksessa käsitellään keskeisiä sisällönhallintaan, portaaleihin ja palveluväylään liittyviä standardeja, sekä konkretisoidaan palvelupohjaisen arkkitehtuurin peruselementit yksinkertaisen esimerkkisovelluksen avulla.

ACM-luokat: (ACM Computing Classification System, 1998 version): D.2.11, D.2.12, H.3.5
Avainsanat: Portaalit, sisällönhallintajärjestelmät, SOA

SISÄLLYSLUETTELO:

LYHENTEET:

API	Application Programming Interface
B2B	Business to Business
BC	Binding Component
BPM	Business Process Management
BPEL	Business Process Execution Language
CMS	Content Management System
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
DMS	Document Management Systems
DTD	Document Type Definition
DTO	Data Transfer Object
EAI	Enterprise Application Information
ECM	Enterprise Content Management
EIP	Enterprise Information portal
EJB	Enterprise Java Beans
ETML	Extract, Transform, Move and Load
ESB	Enterprise Service Bus
FTP	File Transfer Protocol
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDL	Interface Description Language
J2EE	Java 2 Platform, Enterprise Edition
JAX-RPC	Java API for XML-Based RPC
JBI	Java Business Integration
JCA	J2EE Connector Architecture
JCR	Java Content Repository
JMS	Java Message Service
JDK	Java Development Kit
JMX	Java Management Extensions
JRE	Java Runtime Environment

JSP	Java Server Pages
JSR	Java Specification Request
MD5	Message Digest 5 Algorithm
ME	Message Exchange
MOM	Message Oriented Middleware
NMR	Normalized Message Router
ORB	Object Request Broker
POJO	Plain Old Java Object
RDF	Resource Description Framework
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RSA	Rivest, Shamir & Adleman
RSS	Really Simple Syndication
SAML	Security Assertion Markup Language
SCA	Service Component Architecture
SE	Service Engine
SES	Smart Enterprise Suite
SHA1	Secure Hash Algorithm Version 1.0
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UCC	User Created Content
UDDI	Universal Description Discovery and Integration
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WML	Wireless Markup Language
WS-I	Web Services Interoperability Organization
WSD	Web Service Definition
WSDL	Web Service Definition Language
WSDL-S	Web Service Semantics

WSS	Web Services Security
WSRP	Web Services for Remote Portlets
WS-TX	Web Services Transaction
XML	Extensible Markup Language
XSD	XML Schema Definition
XSL	Extensible Stylesheet Language
XHTML	Extensible Hypertext Markup Language

1 JOHDANTO

Yrityksen menestykseen, ainakin pitkällä aikavälillä tarkasteltuna, vaikuttaa osaltaan keskeisesti organisaation muutosherkkyys ja kyky reagoida oikein jatkuvasti muuttuvaan ympäristöön (Kalakota & Robinson, 2001). Esimerkiksi tuotteiden markkinoille tuloaikaa on pystyttävä pienentämään, jotta ne vastaavat nopeammin muuttuvaan ympäristöön ja vastaavat asiakkaiden alati muuttuviin tarpeisiin. Ympäristön nopea muuttuminen aihettaa myös sen, että tiedon tuottajien on voitava pienentää sisällön tuottamiseen kuluvaa aikaa, parannettava tiedon uudelleenkäytettävyyttä ja hyödynnettävyyttä sekä pyrittävä joustavuuteen ja etukäteisaktiivisuuteen organisaation arvoketjuun ja sidosryhmiin kohdistuvassa viestinnässä (Jokela, 2001). Tämä siitäkin huolimatta, että jo sisällön tuottaminen uusilla välineillä eri medioihin, puhumattakaan sisällön hallinnoinnista ja uudelleenkäyttöstä, voi olla hankalaa (Clark, 2008).

Toimintatapojen kehittäminen prosesseja automatisoimalla ja tietotekniikkaa hyödyntämällä mahdollistaa yrityksen nopean reagointikyvyn myös tulevaisuudessa (Kalakota & Robinson, 2001). Tämä edellyttää organisaatiolta joustavaa ja skaalautuvaa, liiketoimintalähtöistä it-arkkitehtuuria, joka mahdollistaa laadukkaiden uusien palveluiden tarjoamisen asiakkaalle, sekä uusien sovellusten kehittämisen ja käyttöönottamisen mahdollisimman kustannustehokkaasti, luotettavasti ja helposti, vastaten kuitenkin organisaation laatu-, luotettavuus- ja tietoturvavaatimuksiin. Muutosherkkyyttä edesauttaa myös olemassa olevien tietojärjestelmäinvestointien tehokas hyödyntäminen, uudelleen järjestämällä niissä oleva tietosisältö uusiksi palveluiksi (Hau & al., 2008).

Palvelukeskeinen arkkitehtuuri on viime aikoina herättänyt huomiota, pyrkien omalta osaltaan vastaamaan edeltä mainittuihin liiketoiminnan haasteisiin. Vaikka suurin osa palvelukeskeisen arkkitehtuurin hyödyistä saadaan todennettua vasta pitkällä aikavälillä (Hau & al., 2008), on hyvä tiedostaa, mitkä ovat arkkitehtuurin peruspiirteet ja mitä se edellyttää ohjelmistokehitykseltä.

Tässä tutkimuksessa olen lähtenyt liikkeelle ajatuksesta, jonka mukaan organisaation liiketoimintaan liittyvä tietosisältö säilötään organisaation sisällönhallintaprosessin mukaisessa sisältövarastossa, josta tämä tietosisältö integroidaan loppukäyttäjälle helposti käytettävään ja tavoitettavaan muotoon (tässä tapauksessa selainpohjaiseen portaalinäkymään) hyödyntämällä

uuden teknologian mukanaan tuomia mahdollisuuksia. Siksipä tässä tutkimuksessa käsitellään portaalien ja sisällönhallintajärjestelmien integrointia hyödyntäen palvelukeskeisen arkkitehtuurin mukaisia teknologioita, kuten web-palveluita ja palveluväylää. Tutkimuksen tarkoituksena on tutustuttaa lukija peruskäsitteisiin, kuten portaaleihin, sisällönhallintajärjestelmiin ja niihin liittyviin standardeihin. Lisäksi tutkimuksessa käsitellään yritysjärjestelmien integrointiin ja palvelukeskeiseen arkkitehtuuriin liittyviä menetelmiä, haasteita ja teknologioita. Asiaa konkretisoidaan toteuttamalla esimerkkisovellus, joka sisältää keskeisimmät elementit aiemmin käsitellyistä asioista.

Tämän johdannon jälkeen, luvussa 2 määritellään termit portaali ja sisällönhallintajärjestelmä sekä kerrotaan yritysjärjestelmien integroinnista. Luvussa tarkastellaan kyseisten osatekniikoiden kehitystä, kerrotaan eri teknologioiden keskeisimmät roolit ja tehtävät, sekä perusteet osajärjestelmien integroimiseksi hyödyntäen palvelukeskeistä arkkitehtuuria. Luvussa 3 tutustutaan palvelukeskeiseen arkkitehtuuriin. Kyseisessä luvussa käydään ensin läpi web-palveluarkkitehtuuri, ja sen keskeiset osa-alueet, kuten web-palvelu ja web-palveluarkkitehtuurin eri mallit. Tämän jälkeen tarkastellaan tarkemmin palvelukeskeisen arkkitehtuurin erityispiirteitä, kuten löyhää sidontaa ja palvelukeskeisen arkkitehtuurin keskeisiä komponenteja. Luvun 3 lopussa tutustutaan palveluväylään ja sen tarjoamiin toiminnallisuuksiin.

Luvussa 4 esitellään keskeisimpiä XML-pohjaisia integraatiotekniikoita. Ensiksi käydään läpi XML:n perusteita, kuten XML-skeemoja, sisällön integrointistandardeja ja web-palveluiden perusasioita, kuten WSDL-kuvaukset, SOAP-viestit ja UDDI-hakemiston toiminta. Lisäksi tutustutaan web-palveluiden tietoturvaan ja yhteensopivuuteen, sekä sisällönhallintajärjestelmien tukemaan JSR 170 -rajapintaan. Luvussa 5 käydään läpi Java-teknisiä toteutusmenetelmiä ja toteutusta tukevia kirjastoja, kuten Apache Axis tai JAX-RPC. Luvussa 6 käydään läpi esimerkkitoteutus, jossa luodaan web-palvelu, jota voidaan käyttää sisällön integroimiseen sisällönhallintajärjestelmästä portaaliin. Esimerkkisovellus käyttää palveluväylää palvelupyyntöjen reititykseen. Lopuksi, luvussa 7, on tämän tutkimuksen yhteenveto.

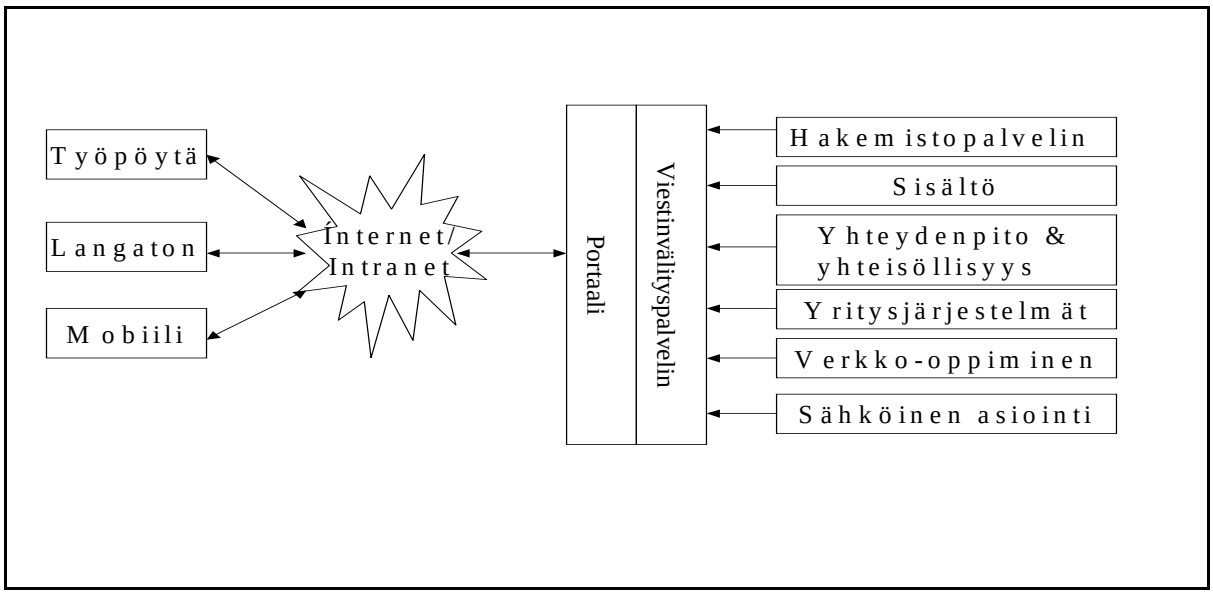
2 PORTAALEISTA, SISÄLLÖNHALLINTAJÄRJESTELMISTÄ JA YRITYSJÄRJESTELMIEN INTEGROINNISTA

Portaaleista on tullut viime vuosina erittäin keskeisiä palvelukanavia asiakkaiden ja organisaatioiden välillä. Samalla kun portaalien lukumäärä on kasvanut, on myös niiden avulla esitettävän tietosisällön tarve ja määrä kasvanut, joka usein heijastuu suoraan myös ylläpidon työmäärään. Jotta portaalit aidosti tehostavat organisaation toimintaa, on niihin liittyvä ylläpitotyö pystyttävä minimoimaan mahdollisimman tehokkaasti. Lisäksi portaaliin liittyvän tietosisällön ylläpitäminen on usein syytä tehdä niiden henkilöresurssien toimesta, jotka muutenkin vastaavat kyseisten tietojen tuottamisesta. Organisaatioiden on pystyttävä myös hyödyntämään jo tehdyt tietojärjestelmäinvestoinnit mahdollisimman hyvin, joka usein onnistuu parhaiten integroimalla olemassa olevia tietojärjestelmiä uusiin hankittaviin järjestelmiin. Tällöin uusiin hankittaviin järjestelmiin ei tarvitse hankkia erikseen vanhoissa järjestelmissä jo olevaa toiminnallisuutta.

Portaalien sisältö voidaan jakaa karkeasti kolmeen osa-alueeseen: *Loppukäyttäjän tuottama sisältö* (User Created Content, UCC) on nopeasti kasvava tapa tuottaa sisältöä kustannustehokkaasti (OECD, 2007). Tässä mallissa sisällön tuotanto ulkoistetaan palvelun loppukäyttäjälle. Loppukäyttäjä käyttää tietyn määrän luovaa panosta normaaleiden ammattirutiiniensa ulkopuolella, ja tuottaa sisältöä, jonka hän usein julkaisee verkossa muiden hyödynnettäväksi. Esimerkkejä loppukäyttäjän tuottamasta sisällöstä ovat erilaiset wikipalvelut, blogit tai keskusteluryhmät. *Sisällöntuottajien tuottama sisältö* sisältää ammattimaisen tiedon tuottamisen, organisaation jonkin liiketoimintaprosessin osana. Sisällöntuottajat voivat olla joko organisaation sisäisiä tai ulkoisia sidosryhmiä. Esimerkkeinä sisällöntuottajien tuottamasta sisällöstä on tv-, ja radioyrityksien verkkosivut, intranet tai erilaiset uutispalvelut. *Automaattisesti tuotettu sisältö* syntyy sivutuotteena käyttäjien ja organisaation muista aktiviteeteista, kuten liiketoimintaprosessien aiheuttama metriikka, joka raportoidaan loppukäyttäjälle. Tässä tutkimuksessa keskitytään lähinnä sisällöntuottajien tuottaman sisällön integrointiin portaaleihin, joskin sen tuloksia voidaan hyödyntää myös automaattisesti tuotetun sisällön esittämiseen.

Vuosituhannen alusta lähtien keskeinen osa organisaatioiden it-investoinneista on kohdistunut sisällönhallintajärjestelmiin ja portaaleihin, jotka kuitenkin ovat joidenkin mielestä itsessään

palveluina liian kapeita pärjätäkseen massamarkkinoilla (Gilbert & al., 2002). On esitetty arvauksia, että tämä johtaa siihen, että portaalit ja sisällönhallintajärjestelmät tulevat integroitumaan yhdeksi kokonaisuudeksi. Gilbert & al. (2002) kertovat *organisaation älykkäästä kokonaisratkaisusta* (Smart Enterprise Suite, SES), joka pyrkii täyttämään yritysten sisäiset sekä niiden väliset tiedonhallinnan, sisällönhallinnan ja työryhmäsovelluksiin liittyvät tarpeet. Kyseisen artikkelin mukaan organisaation älykäs kokonaisratkaisu perustuu toisiinsa eri tasoilla integroiduista osasovelluksista. Vaikka kyseinen artikkeli käynnistikin epäilemättä monta integroimishanketta ohjelmistotoimittajien taholla, on itsenäisiä portaalituotteita ja sisällönhallintajärjestelmiä vielä markkinoilla.



Kuva 1 – Tyypillinen portaaliarkkitehtuuri (Dovey, 2001)

Gilbertin & al. (2002) mukaan osasovellusten tulee koostua sisällönhallinnasta, työryhmäsovelluksista ja portaalikehyksestä (portal framework). Sisällönhallinta käsittää tässä yhteydessä dokumenttien ja verkkosisällön hallinnan sekä mahdollisesti digitaalisen materiaalin hallinnan. Työryhmäsovellukset pitävät sisällään viestintäjärjestelmän, hälytykset, reaaliaikaisen sovellusten jakamisen, paikallaolotiedot ja ketjumuotoiset keskusteluryhmät. Muita mahdollisia SES-järjestelmän osia ovat muun muassa tiedon esittäminen, sisältäen kategorisoinnin ja luokittelun sekä profiloinnin ja osaamisen hallinnan, sekä tekniikat online-yhteisöjen perustamiseen ja prosessien hallintaan. Portaali tarjoaa organisaation tai sen kumppaniorganisaation loppukäyttäjille yhtenäisen, monikanavaisen käyttöliittymän. Kyseisen artikkelin mukainen esimerkinomainen portaaliarkkitehtuuri löytyy kuvasta 1.

Gilbertin & al. (2002) artikkeli on ollut osasyynä nykyiselle kehitykselle, jossa portaaleja integroidaan voimakkaasti erilaisiin tietojärjestelmiin. Tällaista portaalinäkymää kutsutaan usein sähköiseksi työpöydäksi, ja sen tarkoituksena on tuottaa roolin tarjoamat palvelut yhden portaalinäkymän kautta. Tässä tutkielmassa keskitytään portaalin ja sisällönhallintajärjestelmän väliseen integraatioon, joka luo perustan roolikohtaisen työpöydän kehittämiselle.

Jäljempänä tässä luvussa käydään lyhyesti läpi tutkielman kannalta keskeisimpien käsitteiden sisältöä ja taustaa. Ensin käydään läpi portaalien historiaa ja määrityksiä, sekä keskeisimpiä toiminnallisuuksia. Tämän jälkeen tutustutaan sisällönhallintajärjestelmiin, sisällönhallintaprosessiin ja sisällönhallintajärjestelmien keskeisimpiin ominaisuuksiin. Lopuksi tutustutaan yritysjärjestelmien integrointiin ja erilaisiin integraatiotapoihin.

2.1 Portaalit

Seuraavaksi käydään läpi erilaisia portaalitermien määrityksiä. Ensin tarkastellaan portaalikäsitteen syntymistä ja määrittelemme portaaleja niiden toiminnallisuuden kautta. Kohdan lopussa tutustutaan portaalin määritykseen teknisestä näkökulmasta.

Yleisesti uskotaan (Aiken & Finkelstein, 1999; Firestone, 2002), että termi *yritysportaali* (corporate portal) pohjautuu Merill Lynchin raporttiin vuodelta 1998 (Shilakes & Tylman, 1998), jossa määriteltiin käsite *yritysinformaatioportaali* (Enterprise Information Portal, EIP) ensimmäistä kertaa. Yritysinformaatioportaalit ovat sovelluksia, jotka vapauttavat yrityksen sisäisesti ja ulkoisesti tallennetun informaation ja tarjoavat yhden näkymän personoituun tietoon, jota tarvitaan tietoisten liiketoimintapäätösten (business decisions) tekemiseen. Samassa raportissa yritysinformaatioportaalit nähtiin suurena mahdollisuutena uusille ohjelmistoille, jotka yhdistävät, hallitsevat, analysoivat ja jakavat tietoa yrityksen sisällä sekä sieltä ulospäin.

Diasin (2001) ja Reynoldsin & Koulopouloksen (1999) mukaan yritysportaalit kehittyivät internet-hakukoneiden pohjalta. Alun perin hakukoneet mahdollistivat dokumenttien etsimisen käyttäen loogisia operaattoreita ja sivujen välisiä linkkejä. Joihinkin hakukoneisiin liitettiin hakemistoja, joihin ryhmitellään dokumentteja ja sivustoja niiden aihealueiden mukaan, kuten esimerkiksi urheilu, uutiset, kulttuuri. Myöhemmin käyttäjille tarjottiin mahdollisuus perustaa virtuaalisia yhteisöjä ja käydä reaaliaikaisia keskusteluja sekä mahdollistettiin loppukäyttäjälle käyttöliittymän personointi (esimerkiksi My Lycos) ja pääsy erikoistuneeseen ja kau-

palliseen sisältöön.

Reynoldsin & Koulopouloksen (1999) mukaan portaalien kehityksessä on havaittavissa kolme vaihetta: looginen haku, navigointi (kategorisointi), personointi sekä laajennettu toiminnallisuus muiden tietolähteiden integroimiseksi palveluun. Yritysportaalit eroavat muista julkisista portaaleista ainoastaan portaalin tarkoitusperän takia, julkisten portaalien tavoitteena on houkutella paljon kävijöitä, jotta portaali toimisi kiinnostavana mainospaikkana muille yrityksille. Yritysportaalin tarkoituksena on taasen parantaa työntekijän toimintamahdollisuuksia luomalla interaktiivinen, kaksisuuntainen, kanava yrityksen liiketoiminnassa tarvittavaan tietoon (Reynolds & Koulopoulos, 1999).

Murray (1999) laajentaa portaalin tarkastelunäkökulmaa työpöytäympäristön suuntaan. Hänen mukaansa portaali tarjoaa käyttäjälleen erilaisia palveluita, kuten sähköpostin, työnkulut, työpöytäsovellukset ja jopa kriittisten operatiivisten järjestelmien käyttömahdollisuuden, päätarkoituksena helpottaa ihmisten välistä työskentelyä ja kanssakäymistä. Murray (1999) luokittelee erilaisia portaaleja käyttötarkoituksen mukaan:

- yritysinformaatioportaalit (enterprise information portal), jotka yhdistävät ihmisen ja tiedon
- yritysyhteisöportaalit (enterprise collaborative portal), jotka tarjoavat erilaisia yhteisöllisiä palveluita
- yritysasiantuntijaportaalit (enterprise expertise portal), jotka yhdistävän ihmiset toisiin ihmisiin kykyjen, osaamisen tai kiinnostusten kautta
- yritystietämysportaalit (enterprise knowledge portals), jotka yhdistävät yllämainitut tarjoten loppukäyttäjälle personoitua informaatiota.

Dias (2001) tulkitsee, että Murray on kiinnostunut enemmän portaaleista, jotka toimivat loppukäyttäjän varsinaisen työn tukena, eivätkä pelkästään porttina tietoon tai päätöksenteon tukijärjestelmänä.

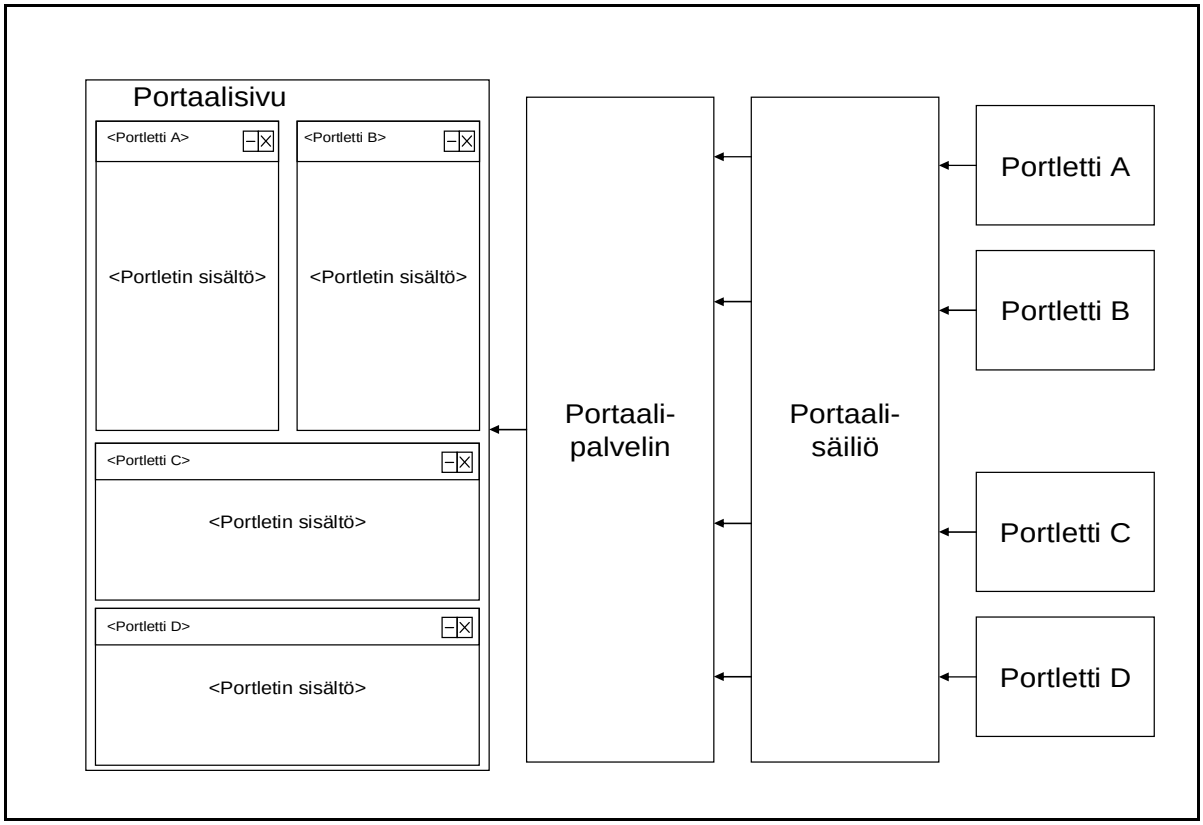
Portaaleja voidaan kategorisoida myös niiden julkaisijaorganisaation mukaan (Sidorof, 2005; O'Murchu & al., 2004). Yritysportaalit ovat yrityksen sisäiseen ja sen sidosryhmien (asiakkaat ja kumppanit) käyttöön tarkoitettuja portaaleja. *Julkishallinnon portaalit* (government portals) tarjoavat valtion ja kunnallishallinnon palveluita kansalaisille.

Yhteisöportaalit (community portals) toimivat yhteisön kommunikointikanavana. Tässä tutkimuksessa keskitytään pääasiassa yritysportaaleihin.

Reynolds ja Koulopoulos (1999) näkevät portaalit käyttäjäkeskeisenä informaatiopalveluna, jonka tarkoituksena on integroida ja jakaa työntekijöiden ja työryhmien tarvitsema tietämys ja kokemus, jotta he saavuttaisivat informaatiokeskeisen työelämän asettamat vaatimukset. Heidän mukaansa yritysportaali yhdistää tietämyksen, joka sisältyy tiedostoihin, tietokantoihin, sähköposteihin, www-sivuihin sekä yritysohjelmistoihin, yhdeksi graafisesti rikkaaksi ja sovellusriippumattomaksi käyttöliittymäksi. Reynolds ja Koupoulos (1999) ovatkin näkemykseltään hyvin lähellä työpöytäpohjaista portaalia.

On kuitenkin syytä muistaa, kuten Firestone (2002) huomauttaa, että eri portaalitermien määrittämisen takana on politiikkaa. Firestone perustelee kantaansa siten, että termi pyritään määrittämään siten, että se suosii jonkun tietyn valmistajan tai analyytikon etuja. Jos valmistajat saavat oman määrityksensä hyväksyttyä, saavat he etua verrattuna heidän kilpailijoihinsa, joiden tuotteista puuttuu joitakin termin määrityksessä olevia piirteitä.

Teknisestä näkökulmasta katsottuna Java Portlet Specification (JSR 168, 2003) määrittää portaalin verkkopohjaiseksi sovellukseksi, joka tyypillisesti mahdollistaa personoinnin, kertakirjautumisen ja se kokoaa sisällön eri sovelluksista yhdeksi www-sivuksi. Portaalisivut koostuvat joukosta *portletteja*, jolloin eri käyttäjät voivat nähdä eri tietosisältöä eri portlettikokoelmien avulla. Portletti on Java-pohjainen verkkokomponentti, joka käsittelee pyyntöjä ja luo dynaamista sisältöä. Luotu dynaaminen sisältö (fragmentti) yhdistetään *portlettisäiliössä* (portlet container) muiden portlettien sisältöön. Portlettisäiliön tärkein tehtävä on tarjota ajonaikainen ympäristö ja elinkaaren hallinta portleteille. Portlettisäiliö sisältää myös pysyvän tallennustilan portlettien ominaisuuksien ja asetusten tallentamiseen.



Kuva 2 – Portaalisivun muodostuminen (JSR 168, 2003)

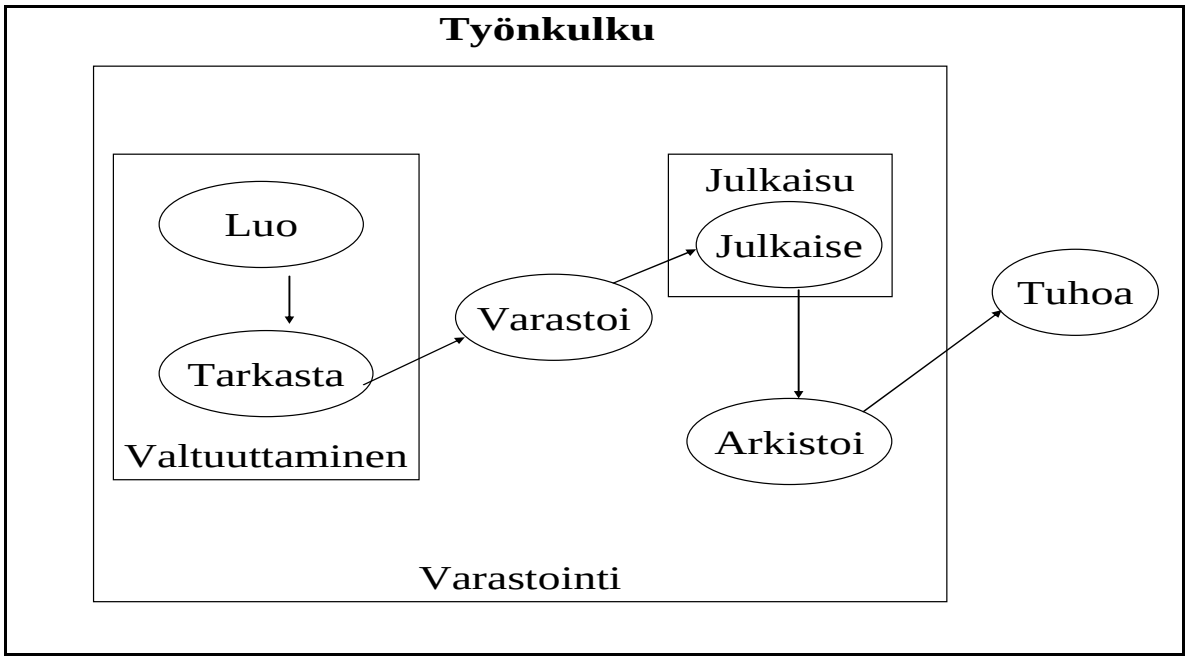
Portletin sisältö, *fragmentti*, koostuu rakenteisella kielellä, kuten HTML, XHTML tai WML, esitetystä sisällöstä. Portaalisäiliö ajaa portletteja, ja se välittää portleteilta saamansa sisällön portaalille. Portaali liittää fragmentin sisältöön muun muassa hallinnointiin tarvittavia painikkeita sekä koristeita ja tätä kokonaisuutta kutsutaan *portletti-ikkunaksi*. Portaali kokoaa eri portaali-ikkunat yhdeksi portaali-dokumentiksi, jota kutsutaan *portaalisivuksi* ja lähettää sen asiakkaalle (esimerkiksi selain) (kuva 2).

Javan portlettimäärityksestä on tulossa uusi versio, 2.0, joka tunnetaan myös nimellä JSR 286 (2007). Uuden version tarkoituksena on parantaa ja lisätä aiemman version (JSR 168, 2003) toiminnallisuutta, ja se tulee olemaan näillä näkymin alaspäin yhteensopiva. Uudet piirteet liittyvät pitkälti WSRP (2006) standardin versio 2.0:ssa kuvattuihin ominaisuuksiin, kuten portlettien väliseen kommunikointiin ja portlettivälimuistiin.

2.2 Sisällönhallintajärjestelmät

Sisällönhallinta (content management) tarkoittaa ”kokonaiskonseptia, joka kattaa kaiken si-

sällön julkaisemisen digitaalisilla työkaluilla" (Addey & al., 2002). Boiko (2002) tarkentaa kokonaiskonseptia sisällön keräämisen, hallinnan ja julkaisemisen prosessiksi. Sisällönhallinnassa keskeisiä toimintoja ovat tietovarantojen hallinta ja organisointi, tiedon muuntaminen ja esittäminen sekä julkaisu ja jakelu eri sidosryhmille (Addey & al., 2002). Clark (2008) luokittelee, että sisällönhallinta tarkoittaa nimenomaan sisällönhallintaan liittyvää prosessia, ja *sisällönhallintajärjestelmä* (Content Management System, CMS) tarkoittaa digitaalista teknologiaa (joka on joskus ohjelmisto, joskus laitteisto), joka auttaa sisällönhallinnassa yksinkertaistamalla ja automatisoimalla sisältöobjektien käsittelyä. Sisältöobjektien käsittelyyn kuuluvat sisällön luominen, hyväksyminen, varastointi, vastaanottaminen, versiointi, uudelleenkäyttö ja jakelu. Browning & Lowndes (2001) jakavatkin sisällönhallintajärjestelmän tärkeimmät toiminnot neljään kategoriaan: valtuuttamiseen, työnkulkuun, varastointiin ja julkaisuun (kuva 3).



Kuva 3 - Sisällönhallinnan prosessi

Valtuuttaminen mahdollistaa hajautetun sisällöntuotannon hallitussa ja valvotussa ympäristössä. *Työnkulku* on prosessi, joka mahdollistaa hyväksymisen ja kommentoinnin sisällöntuotannon ja julkaisun välissä. *Varastointi* tarkoittaa valtuutetun sisällön tallentamista tietovarastoon, käsittäen myös versionhallinnan. Versionhallinnan avulla vältytään useiden ylläpitäjien aiheuttamista sisällön virhetilanteista ja saadaan tarvittaessa palautettua vanhat versiot. Varastoinnin yhteydessä sisältö voidaan pilkkoa pienempiin, järjestettyihin osakokonaisuuk-

siin jotka voidaan tallentaa esimerkiksi tietokantariveiksi tai Extensible Markup Language (XML) -tiedostoiksi. *Julkaisu* tarkoittaa sisällön saattamista loppukäyttäjän ulottuville, halutussa muodossa.

Browning & Lowndes (2001) käyttävät sisällönhallintajärjestelmän määrittävinä tekijöinä kolmea perusominaisuutta: versionhallinta, työnkulku ja integraatio. Jos järjestelmässä on nämä ominaisuudet, voidaan sitä kutsua sisällönhallintajärjestelmäksi. Muut ominaisuudet voidaan jakaa pääosin viiteen kategoriaan:

- Käyttäjien hallinta, eli käyttäjäroolien ylläpito ja käyttöoikeuksien hallinta.
- Ulkoasu ja käyttöliittymä, mielellään selainpohjainen käyttöliittymä sisällöntuottoon ja sisällönhallintaan ja/tai www-palvelun hallintaan.
- Tietolähteet, pitää sisällään hallitun, järjestelmällä luodun tiedon varastoinnin, sekä toimisto-ohjelmilla luotujen tiedostojen sisällön hallinnan. Tiedon varastointimenetelmät voivat olla tiedostojärjestelmiä, relaatiotietokantoja, oliotietokantoja tai nyttemmin XML-tiedostoja. Avainasemassa on järjestelmän tarjoama mahdollisuus hyödyntää tietoa mahdollisimman joustavasti. Tietovarasto vaatii myös, että tiedon tulee olla kuvattua. Tätä tiedon kuvaustietoa kutsutaan *metatiedoksi*.
- Sovellukset, jotka integroivat sisällön olemassa olevaan tietoon ja tunnistusmenetelmiin, sekä suorittavat tiettyjä ohjelmallisia muokkauksia sisältöön hallittavuuden, eheyden ja yksinkertaisuuden lisäämiseksi. Kantavana ajatuksena on mahdollistaa verkkopalvelun ulkoasun säilyminen yhdenmukaisena ja keskitetysti hallinnassa, jolloin se on helposti mukautettavissa. Sovellukset voivat myös sisältää toiminnallisuuden tiedon dynaamiseen esittämiseen suoraan tietokannasta www-sivuille.
- Jakelu, verkkopalvelun julkaisu tuotantoon. Joissakin sovelluksissa sisällönhallintajärjestelmä ei eroa sisällöntuotantopalvelun ja varsinaisen julkaistavan sisältöpalvelimen osalta, vaan sisällönhallintajärjestelmä on verkkopalvelu ja se generoi siltä pyydetyn tietosisällön dynaamisesti. Toisissa järjestelmissä sisällönhallinta- ja julkinen puoli ovat tiukasti eroteltuja, ja ne saattavat sijaita eri koneilla. Tällöin koko sisältörakenne saatetaan joutua replikoimaan, mikäli kaikki sivut ovat dynaamisesti tuotettuja. Joissakin tapauksissa osa sisältöelementeistä on esimuotoiltuja ja julkaistuja staattiseen muotoon, jolloin ainoastaan tietty dynaaminen sisältö käydään hakemassa sisällönhallintajärjestelmästä julkisen palvelun toimesta.

Sisällönhallintajärjestelmän halutuimmat ominaisuudet Doylen (2000) mukaan ovat roolipohjainen oikeusmalli, omatoiminen sisällön luominen ja saumaton julkaisu (ilman toimittajan apua), joustava tiedon tuottaminen ja muotoilu (kirjoita kerran, julkaise useasti), työnkulun hallinta, metatietojen hallinta ja integroitavuus olemassa olevaan tietoon.

Clark (2008) kategorisoi sisällönhallintajärjestelmät dokumenttienhallintajärjestelmiin (Document Management System, DMS), web-sisällönhallintajärjestelmiin (Web CMS), sisällönhallintajärjestelmiin (CMS) ja organisaation sisällönhallintaan (Enterpise Content Management, ECM). *Dokumenttienhallintajärjestelmät* toimivat dokumenttitasolla, eli ne varastoivat dokumentit sellaisinaan, kytkevät niihin metatietoa ja tarjovat toimintoja, kuten versiointia ja hakua. Esimerkkinä dokumenttienhallintajärjestelmästä toimii vaikkapa Googlen Picasa-verkkoalbumit (Google Picasaweb, 2008), joihin voi tallentaa kokonaisia kuvadokumentteja ja kytkeä niihin metatietoa. *Web-sisällönhallintajärjestelmät* sisältävät toiminnallisuuden www-sisältöjen luomiseen, esittämiseen ja ylläpitoon. Web-sisällönhallintajärjestelmiä on useita erilaisia, alkaen yksinkertaisista blogieditoreista aina monimutkaisempiin www-sivustojen hallintaohjelmistoihin. Sisällönhallintajärjestelmällä Clark (2008) tarkoittaa järjestelmää, joka lähestyy sisällönhallinnan problematiikkaa käyttämällä rakenteista tiedonkuvausta (markup), metatietoja, ja työkaluja jakaakseen dokumentin organisaation määrittämän tietomallin mukaisiin osiin, ja kiinnittämällä kuhunkin osaan metatietoa, joka kertoo osan tarkoituksen ja suhteen muuhun sisältöön. *Organisaation sisällönhallinnalla* tarkoitetaan järjestelmää, joka kokoaa edellä mainittujen järjestelmien toiminnot yhteen ja lisää siihen systemaattisesti vielä muita julkaisutoimintoja, kuten esimerkiksi koko organisaation sähköpostit, sekä talous- ja henkilöstöhallinnon raportit.

Vaikka täydellinen eriyttäminen ulkoasun ja sisällön välillä onkin mahdotonta (Clark, 2008), CMS-järjestelmään tallennettu sisältö on syytä pyrkiä erottamaan sen ulkoasuun ja esittämiseen liittyvistä asioista, jotta sen hyödyntäminen on mahdollisimman tehokasta (Rockley & al., 2003). Clark (2008) perustelee kantaansa sillä, että kaikella sisällöllä on aina jokin ulkoasu, koska jopa kaikkein yksinkertaisimmilla Notepad-teksteillä on ulkoasu, johon vaikuttaa esimerkiksi tekstin asettelu, fontti, kappalejaot ja otsikot. Clark (2008) kuitenkin myöntää, etteivät yritykset eriyttää ulkoasu sisällöstä ole turhia. Rockleyn & al. (2003) mukaan tehokas tiedon hyödyntäminen edellyttää, että sisältöä osataan tulkita, kun se siirtyy järjestelmästä toiseen. Sisällön tulkinta edellyttää yhteistä tietomallia (rakennetta), yhteistä tapaa kuvata tieto-

sisältöä ja yhteistä metatietoa. Tässä tutkielmassa keskitytään sisällön integraatioon lähinnä teknisestä näkökulmasta, johon ei sisälly tiedon tulkintaan tai sen esittämiseen liittyvä problematiikka kuin ainoastaan välttämättömin osin.

2.3 Yritysjärjestelmien integroinnista

Yritysjärjestelmien integrointi (Enterprise Application Integration, EAI) mahdollistaa yritysjärjestelmien välisen yhteisen tiedon vaihtamisen. Rodsendahlin & Runen (2000) mukaan ei ole olemassa yleispätevää ratkaisua, joka soveltuisi jokaiseen integroimistarpeeseen, vaan jokainen integroimistapaus vaatii oman yksilöllisen ratkaisunsa. Rosendahl & Rune (2000) listaavat joitakin integraation haasteita:

- Tietoturvahaasteet: tiedon siirron luotettavuus ja käyttäjien tunnistaminen.
- Nopeus: integraatioratkaisun nopeus/hitaus.
- Joustavuus: integraation mukauttaminen uuteen ympäristöön.
- Tulevaisuus: mitä kyseiselle ratkaisulle tapahtuu jatkossa?
- Vikasietoisuus: mitä tapahtuu, jos jotakin menee vikaan?
- Käytettävyys: onko ratkaisu helposti ymmärrettävä, jäsennetty kokonaisuus?
- Kehitykseen käytetty aika: onko kehittäminen kyllin nopeaa?

Integrointiprojektissa joudutaan yleensä ottamaan kantaa edellä mainittuihin haasteisiin, ja ollakseen integrointiratkaisu, on sen koostuttava vähintään yhdestä seuraavista palveluista (Rosendahl & Rune, 2000):

- Yhteyspalvelut: viestien välittäminen eri tietolähteiden välillä, kuten TCP/IP.
- Verkkopalvelut: tietoturvan, jonotuksen, hajautuksen ja kuorman tasauksen lisääminen edelliseen.
- Vuorovaikutuspalvelut: useiden erilaisten viestinvälitystapojen tukeminen. Viestinvälitystavat voivat toimia pyyntö/vastaus, julkaise/tilaa, julkaise/vastaa tai keskusteluperiaatteella.
- Liittymäpalvelut: useat tavat resurssien hyödyntämiseen, kuten erilaiset adapterit.
- Rakennepalvelut: tiedon rakenteen ja merkityksen hallinta.

- Tietovirtapalvelut: integroitujen resurssien välinen tietovirtojen hallinta.
- Muunnospalvelut: tiedon muunnokset muodosta toiseen.

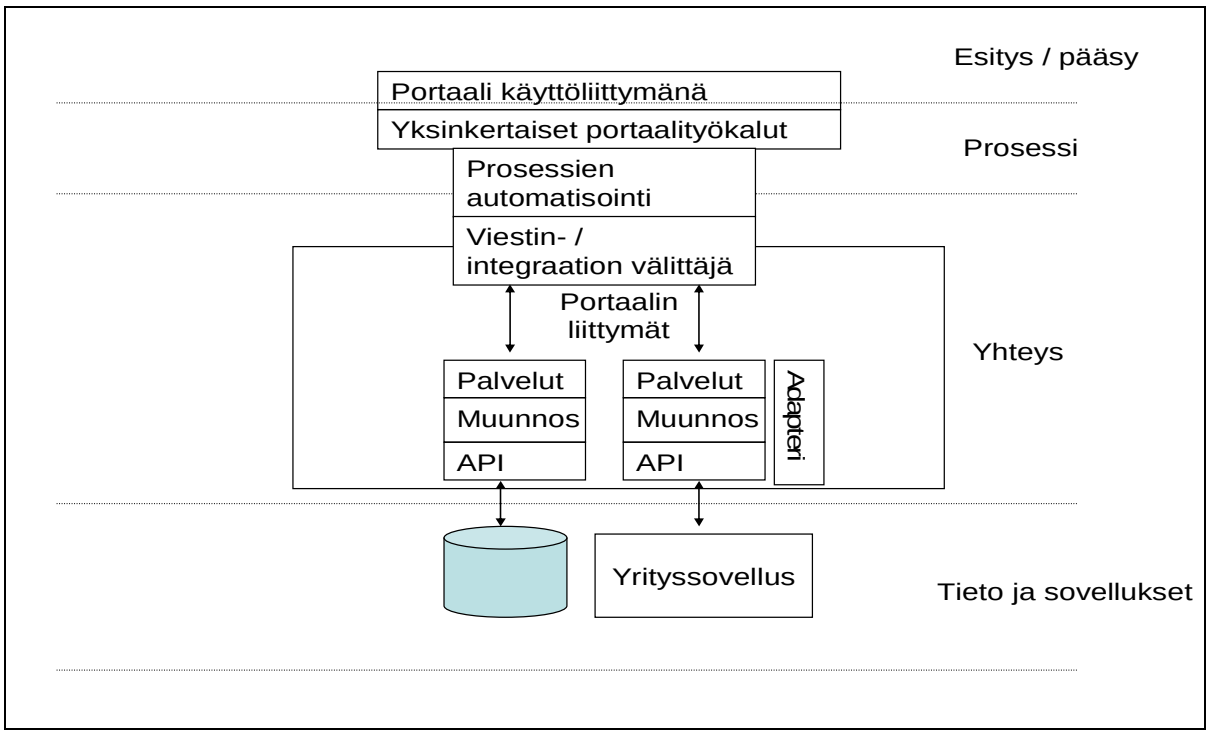
Hurwitz Group on luonut erilaisille integraatioille kategorisoinnin integraation kypsyyden perusteella. Niitä kuvailevat mm. Rosendahl & Rune (2000) sekä Gold-Bernstein (1999): *Alustaintegraatio* tarjoaa liityntäpalveluita erilaisten laitteistojen, käyttöjärjestelmien ja sovellusalustojen välillä. Alustaintegraation tekniikoihin kuuluu oliopyyntöjen välittäjät (Object Request Brokers, ORB), ja etäproseduurikutsut (Remote Procedure Calls, RPC), sekä viestinvälitys. Tekniikoista RPC:t ovat synkronisia (pyytäjä odottaa vastausta ennen kuin suoritus jatkuu), viestinvälitys on asynkroninen (suorittava järjestelmä tekee pyynnön, eikä jää odottamaan vastausta, vaan jatkaa suoritustaan) ja ORB:t voivat olla joko asynkronisia tai synkronisia. Kaikissa mittavissa integroimisprojekteissa tarvitaan molempia yhteystapoja, sekä yleensä myös julkaise/tilaa periaatteella toimivaa yhdeltä lähettäjältä monelle vastaanottajalle kohdistuvaa viestinvälitystä (Gold-Bernstein, 1999).

Tietointegraatio tarkoittaa tietolähteiden integroimista suoraan tietokantatasolla, joko tietokantayhdyskäytävän kautta tai poiminta, muunto, siirto ja lataustyökalujen (Extract, Transform, Move and Load, ETML) avulla. Tietokantayhdyskäytävä tarjoaa synkronisia integraatiopalveluita, ETML-työkalut yleensä eräajopohjaisia. Tietointegraatiossa yleensä ohitetaan sovelluksen logiikka, ja tietoa siirretään suoraan sovellusten välillä. Tietointegraatio soveltuu esimerkiksi tietovaraston pohjatietojen lataamiseen tai suuriin eräajoihin (Gold-Bernstein, 1999).

Komponentti-integraatio on tietointegraation kehittyneempi muoto. Komponentti-integraation perusta on tuotepohjainen palvelinratkaisu, joka tarjoaa aiemmin esitetyt palvelut sekä mahdollistaa niiden lisäksi uuden logiikan rakentamisen. Komponentti-integraatioon tarkoitettuun tuotteeseen on lisätty yleensä ominaisuuksia, kuten kuorman tasaaminen, istuntojen hallinta, tietoturva ja vikasietoisuus (Rosendal & Rune, 2000). Tunnetuimpia komponentti-integraatiotekniikoita ovat EJB, DCOM sekä CORBA.

Sovellusintegraatio tarjoaa erilaisten teknologioiden muodostaman kehikon integraatioiden luomiseen ja muuttamiseen. Kehys mahdollistaa lähes reaaliaikaiset integraatiot ja se sisältää yllä mainittujen ominaisuuksien lisäksi adaptereita yleisimpiin liitettäviin järjestelmiin. Tämä nopeuttaa uusien sovellusten liittämistä huomattavasti.

Prosessi-integraatio mahdollistaa graafisen mallinnustyökalun avulla integraation pidemmälle viedyn abstrahoinnin ja mukautettavuuden. Prosessien mallinnustyökalulla liiketoiminnasta vastaavat voivat määrittää, muuttaa ja valvoa liiketoimintaprosesseja. Asiakkaiden, toimittajien, ja kumppaneiden sovellukset voidaan liittää yrityksen integraatioratkaisuihin liiketoimintaintegraation (Business to Business -integraatio, B2B) avulla. Liiketoimintaintegraatio on tärkeä väline e-liiketoiminnan kehittämiseen, koska yritysten on voitava automatisoida yhä laajempia prosesseja ja yhdistää entistä useammista tietolähteistä koostuvaa informaatiota (Kalakota & Robinson, 2001).



Kuva 4 - Yhdistetty EAI- ja portaaliarkkitehtuuri (Ferreira & Harris-Jones, 2004)

Ferreira & Harris-Jones (2004) yhdistivät portaali- ja EAI-arkkitehtuurit kuvan 4 mukaisesti. Ratkaisumallissa saatiin yhdistettyä molempien arkkitehtuurien edut, kuten sisällön integrointi ja integrointien uudelleenkäytettävyys. Tiedon esittäminen ja pääsynvalvonta ovat portaalin vastuulla, sisältäen aktiivisen, tilauspohjaisen sekä passiivisen, ylläpitäjien toimesta tapahtuvan tiedon jakelun. Prosessikerros muodostuu portaalien ja EAI-työkalujen yhteisistä piirteistä. Portaali huolehtii muun muassa sisällön esittämiseen liittyvistä prosesseista, kun taas EAI-työkalujen keskeisintä ominaisuutta, eli työnkulkuja voidaan käyttää porttlettien yhdistämiseen, jolloin eri sovelluksiin liitetyt portletit saatiin koottua liiketoimintaprosessien

mukaisiksi ketjuiksi. Yhteyskerros mahdollistaa portlettien tehokkaamman uudelleenkäytön. EAI-työkalut sisältävät modulaarisia ominaisuuksia, jotka helpottavat portlettien liittämistä eri taustajärjestelmiin, sekä erityyppisten sisältöjen siirtämistä. Kuvan 4 ratkaisumalli on edelleenkin pääperiaatteiltaan validi, tosin nykyään integraatiot pyritään toteuttamaan usein vakioituja rajapintoja noudattaen. Tätä kehitystä edesauttaa palvelukeskeinen arkkitehtuuri, jota selvitetään luvussa 3.

3 PALVELUKESKEISESTÄ ARKKITEHTUURISTA

Aiemmin kävimme läpi portaaleja, sisällönhallintajärjestelmiä ja yritysjärjestelmien integrointia. Tässä luvussa käydään läpi palvelukeskeistä arkkitehtuuria, joka pohjautuu pitkälti web-palveluihin ja siihen liittyvään web-palveluarkkitehtuuriin. Ensin käydään läpi web-palvelukeskeisen arkkitehtuurin keskeisimmät käsitteet, toimijat ja mallit. Tämän jälkeen tutustutaan palvelukeskeisen arkkitehtuurin pääperiaatteisiin, jonka jälkeen tarkastellaan palveluväylän roolia osana palvelukeskeistä arkkitehtuuria.

3.1 Web-palveluarkkitehtuuri

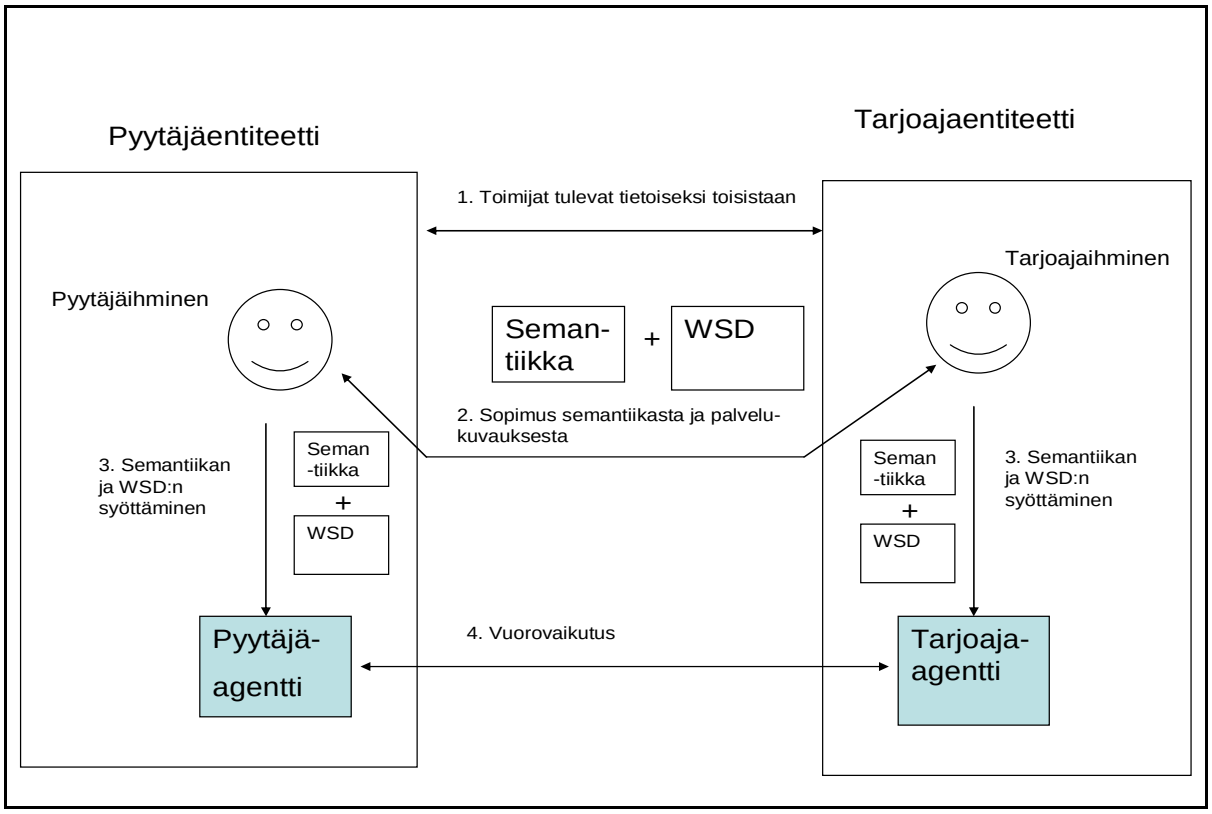
Web-palveluarkkitehtuuri (Web Service Architecture) (W3C, 2004b) kuvaa käsitteellisen mallin ja kontekstin web-palveluiden ja mallin komponenttien välisten suhteiden ymmärtämiseksi. Se määrittelee web-palveluiden yleiset piirteet. Web-palveluarkkitehtuuri perustuu palveluiden yhteentoimivuuteen: se identifioi ne globaalit web-palveluverkoston elementit, jotka vaaditaan palveluiden yhteentoimivuuden varmistamiseksi.

Web-palvelu tarkoittaa ohjelmistojärjestelmää, joka on suunniteltu tukemaan koneiden välistä kommunikointia verkon välityksellä. Sillä on rajapinta, joka on kuvattu koneellisesti prosessoitavassa muodossa. Muut järjestelmät kommunikoivat web-palvelun kanssa sen kuvauksessa määrätyllä tavalla käyttäen SOAP-viestejä, jotka yleensä siirretään HTTP:n yli XML-serialisointia ja muita internet-pohjaisia standardeja hyödyntäen (W3C, 2004b).

Web-palvelu on abstrakti käsite, joka tulee toteuttaa konkreettisen *agentin* avulla (W3C, 2004b). Agentti on ohjelmisto tai laitteisto, joka lähettää ja vastaanottaa viestejä, kun taas palvelu on resurssi, joka määräytyy tarjotun (abstraktin) toiminnallisuuden perusteella. Palvelu voidaan siis esittää yhden tai useamman agentin avulla, jos samaa palvelua hyödynnetään esimerkiksi eri ohjelmointikielillä toteutettujen agenttien avulla.

Web-palvelun tarkoituksena on tarjota toiminnallisuutta sen omistajansa puolesta (W3C, 2004b). Omistaja voi olla esimerkiksi yritys, yhteisö tai yksilö. *Tarjoajaentiteetti* on omistaja, joka tarjoaa soveltuvan agentin tietyn palvelun toteuttamista varten. *Pyytäjäentiteetti* on henkilö tai organisaatio, joka haluaa hyödyntää tarjoajan palvelua. Se käyttää *pyytäjäagenttia*

viestinvaihtoon tarjoajaentiteetin *tarjoaja-agentin* kanssa. Jotta viestinvaihto onnistuu, on tarjoajaentiteetin ja pyytäjäentiteetin ensin sovittava viestin semantiikasta ja viestinvälitystavasta. Kuva 5 havainnollistaa web-palvelun käyttöönottoa.



Kuva 5 - Web-palvelun yleinen kytkeytymisprosessi (W3C, 2004b)

Mekanismi viestinvälitykseen määritetään Web-palvelukuvauksessa (Web Service Description, WSD). Web-palvelukuvaus on koneellisesti prosessoitavissa muodossa oleva määritys palvelun rajapinnasta, ja se kuvataan käyttämällä siihen tarkoitettua kieltä (Web Services Definition Language, WSDL). Palvelukuvaus kuvaa viestimuodot, tietotyypit, siirtoprotokollat ja siirron serialisointimuodot, joita pyytäjä- ja tarjoaja-agentti tarvitsevat keskinäiseen kommunikaatioon. Lisäksi kuvataan tarjoaja-agentin verkko-osoite. Yksinkertaistettuna palvelukuvaus edustaa sopimusta tavasta, jolla palvelun kanssa kommunikoidaan. Palvelukuvauksesta kerrotaan lisää jäljempänä luvussa 4.

Web-palveluiden semantiikalla (Web Service Semantics, WSDL-S) (W3C, 2005) tarkoitetaan yhteistä näkemystä siitä, miten palvelu toimii, kun se vastaanottaa viestin ja vastaa siihen. Tämä tarkoittaa tarjoajaentiteetin ja vastaanottajaentiteetin välistä sopimusta tietyn toiminnon syistä ja seurauksista. Käytännössä tämä sopimus voi olla virallinen tai epävirallinen, lisäksi

se voi olla esimerkiksi suullinen tai kirjallinen, sillä se edustaa yhteisymmärrystä siitä miksi palveluiden väliset transaktiot ylipäätään tapahtuvat. Palvelun kuvaus kertoo miten palvelun kanssa kommunikoidaan, kun taas semantiikka kertoo tarkoituksen tälle kanssakäymiselle. Raja palvelun kuvauksen ja semantiikan välillä on häilyvä. Mikäli palvelu kuvataan semanttisesti riittävän rikkaalla kielellä, saadaan olennainen tarkoitus sisällytettyä palvelun kuvaukseen. Tämän avulla palveluiden käyttöönoton automatisointia voidaan kehittää entisestään.

Web-palveluarkkitehtuuriin (W3C, 2004b) kuuluu neljä mallia, jotka ovat: palvelukeskeinen malli (Service Oriented Model), resurssikeskeinen malli (Resource Oriented Model), sääntökeskeinen malli (Policy Oriented Model) ja viestikeskeinen malli (Message Oriented Model). Mallit ovat osittain päällekkäisiä, eli osassa malleista hyödynnetään muiden mallien piirteitä. *Palvelukeskeinen malli* keskittyy nimensä mukaisesti palveluihin, toiminnallisuuksiin ja muihin samantyyppisiin asioihin. Palvelukeskeinen malli on arkkitehtuurimalleista monipuolisin ja samalla myös monimutkaisin. Palvelukeskeinen malli perustuu siihen, että palvelu realisoidaan yhden agentin toimesta ja käytetään toisen agentin toimesta. Palvelut välitetään tarjoajan ja pyytäjän välillä vaihdettavien viestien avulla. Palvelukeskeisen mallin ajatusmalliin kuuluu myös, että palvelut yleensä toteutetaan lisäarvon ja toiminnallisuuden tarjoamiseksi tosimaailmassa. Mallissa palvelun omistajalle kuuluu vastuu palvelusta tosimaailmassa. Palvelukeskeinen malli hyödyntää myös metatietoa, jota käytetään esimerkiksi palvelun rajapintojen kuvauksiin, palvelun semantiikan kuvaamiseen tai palvelua koskevien sääntöjen kuvaamiseen.

Resurssikeskeisessä mallissa keskitytään resursseihin, jotka ovat olemassa, ja joilla on omistaja. Agentti löytää resurssin, jolla on URI ja mahdollisesti myös resurssin tilaa kuvaava esitys, joka tyypillisesti kuvataan URI:iin kohdistuvalla HTTP GET:illä. *Sääntökeskeisessä mallissa* keskitytään rajoitteisiin, jotka koskevat agentteja ja palveluita. Agentti on riippuvainen ihmisten asettamista säännöistä, jotka liittyvät sen hakemiin resursseihin ja suorittamiin toimenpiteisiin. *Viestikeskeisessä mallissa* keskitytään viesteihin, niiden rakenteeseen ja siirtotiehen, ei liiemmälti viestien lähettämisen syihin tai niiden merkitykseen. Viestikeskeinen malli perustuu siihen, että agentti ottaa vastaan ja lähettää viestejä, jotka jakautuvat viestin otsikkoon ja varsinaiseen viestirunkoon, ja varsinaiseen viestin välitystapaan.

Web-palveluarkkitehtuurin haasteet ovat käytännössä samoja kuin myös muilla tavoilla toteutettujen hajautettujen järjestelmien vastaavat haasteet. Kendall & al. (1994) listaavat kes-

keisimmiksi haasteiksi suorituskyvyn, muistinkäytön, järjestelmän osittaisen toimivuuden/toimimattomuuden ja yhtäaikaisuuden. Jos web-palveluiden käyttämiä tekniikoita peilataan edellä mainittuihin haasteisiin, voidaan arvioida, että suorituskyky on keskeisessä asemassa, kun arvioidaan web-palveluiden mahdollisia käyttökohteita. XML-pohjainen viestiliikenne kasvattaa viestikokoa, jolloin verkon yli siirrettävä tietomäärä lisääntyy entisestään. Myös XML:n muodostaminen ja muunto tietorakenteeksi kuluttavat sekä aikaa että prosessoritehoa.

Muita web-palveluihin liittyviä ongelmia ovat standardien jatkuva kehittyminen (tai kehittymättömyys) ja niiden versiointiin liittyvät haasteet. Tietoturva tuottaa tänä päivänä hienoisia haasteita, mutta tilanne on paranemaan päin, esimerkiksi kohdassa 4.4 tarkasteltavan standardin WS-Security (2006) myötä. Myös luotettavuuden parantamiseen tähtäävä standardi WS-Reliability (2004) laajentaa web-palveluiden perustoiminnallisuutta. Lisäksi SOAP-protokollasta itsestään puuttuvaan transaktioiden hallintaan on luotu standardi WS-Transaction, WS-TX (2007).

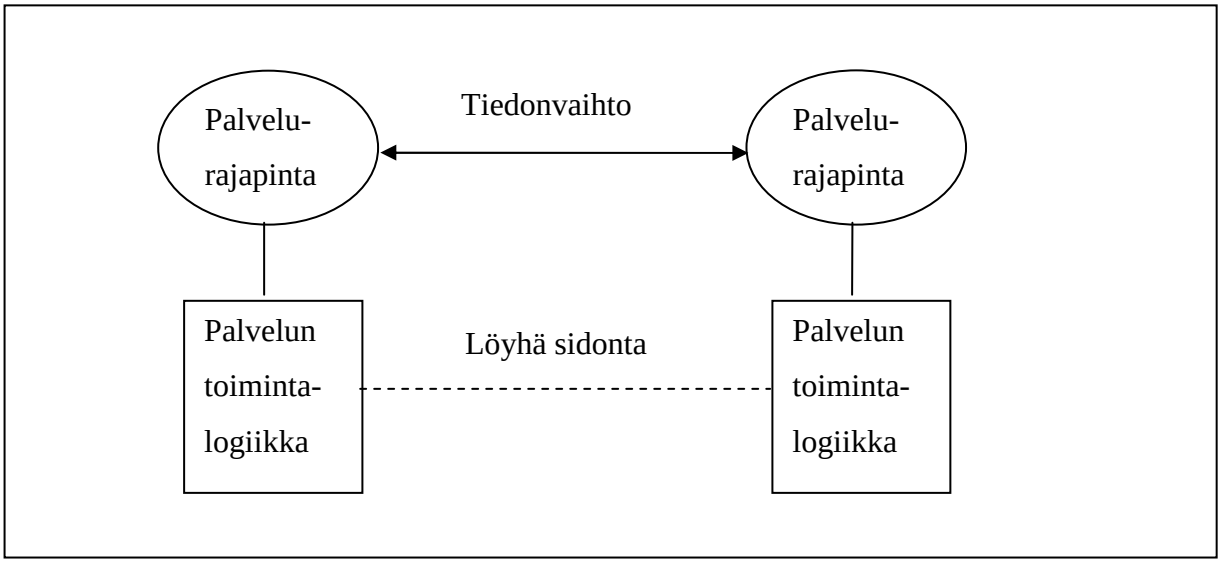
Web-palvelut ovat erinomaisia tilanteissa, joissa hajautetun järjestelmän eri komponentit on toteutettu toisistaan poikkeavilla teknologioilla tai ne ovat eri toimittajien tekemiä. Myös perintöjärjestelmien osien tarjoaminen palveluna on hyvä toteuttaa käyttäen web-palveluita (W3C, 2004b).

3.2 Web-palveluista palvelukeskeiseen arkkitehtuuriin

Palvelukeskeisen arkkitehtuurin (Service Oriented Architecture, SOA) taustalla on ajatus siitä, että logiikka, joka tarvitaan suuren ongelman ratkaisemiseen, on helpommin toteutettavissa, käyttöönotettavissa ja hallittavissa, jos se puretaan pienempiin osiin (Erl, 2005). Tietojärjestelmien kannalta tämä tarkoittaa sitä, että toteutus tulisi jakaa palveluihin, jotka koostuvat palveluista. Kukin palvelu voi sisältää yhden prosessin vaiheen, tai kokonaisen aliprosessin. Jotta palveluista saadaan koottua liiketoimintaprosessi, on palveluiden muodostettava suhteita niihin toimijoihin, jotka haluavat hyödyntää niitä.

Kuten aiemmin kohdassa 3.1 on mainittu, jotta pyytäjä- ja tarjoajaentiteetit voivat hyötyä toistaan, on niiden oltava tietoisia toisistaan, eli tarvitaan yhteinen palvelukuvaus (Erl, 2005). Palvelut hyödyntävät palvelukuvauksia *löyhän sidonnan* periaatteen mukaisesti. Löyhä

sidonta tarkoittaa tilaa, jossa palvelu on tietoinen toisesta palvelusta, olematta siitä kuitenkaan riippuvainen. Löyhä sidonta saavutetaan käyttämällä sellaisia palvelukuvauksia, joissa on ennalta määrätyt parametrit. Käytännössä tämä tarkoittaa sitä, ettei palvelun kuluttajan tarvitse tietää palvelun tuottajan palvelutoteutuksesta mitään muuta kuin mitä palvelukuvauksessa on sovittu (ja päinvastoin). Tällöin palvelun (ja sen kutsujan) toteutuslogiikka ei näy vastapuolelle. Asiaa havainnollistaa kuva 6.



Kuva 6 – Palveluiden välinen löyhä sidonta (Erl, 2005)

Jotta palvelut voivat säilyttää löyhän sidonnan toistensa välillä, tulee niiden keskinäinen kommunikointi perustua sellaiseen malliin, joka mahdollistaa sen. Esimerkkinä tällaisesta kommunikaatiomallista on viestipohjainen kommunikaatio (Erl, 2005). Viestipohjaisen kommunikaation viestit eivät ole lähetyksen jälkeen riippuvaisia enää lähettäjästä, joten niiden tulee olla autonomisia, kuten niitä hyödyntävien palveluidenkin tulee olla.

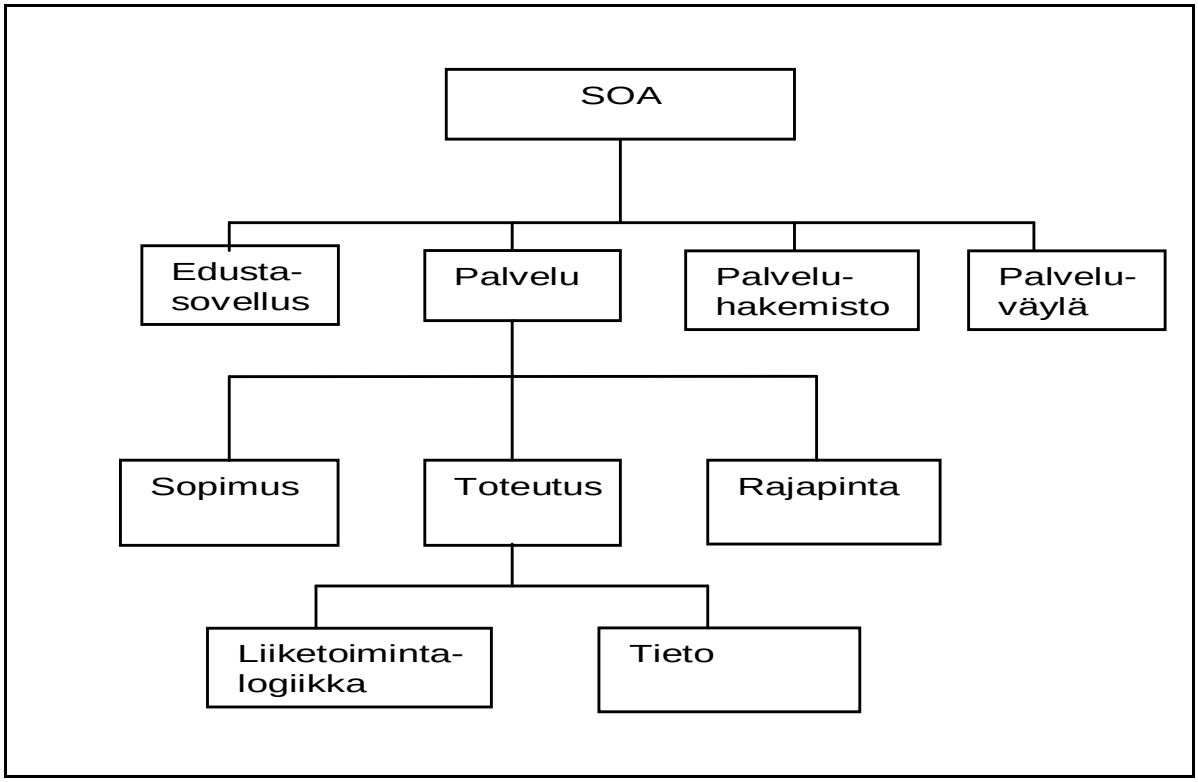
Käytännössä palvelukeskeisen arkkitehtuurin keskeisin ajatus on noudattaa palveluiden suunnittelussa ja toteutuksessa tiettyjä periaatteita. Erl (2005) nimeää keskeisimmiksi periaatteiksi seuraavat:

- *Löyhä sidonta* – palvelut säilyttävät toistensa välillä suhteita, jotka minimoivat niiden keskinäiset riippuvuudet ja vaativat ainoastaan toistensa tiedostamista.
- *Palvelusopimus* – palvelut noudattavat kommunikointisopimusta, joka on kuvattu yhteisesti määritetyssä palvelukuvauksessa ja siihen liittyvissä dokumenteissa.

- *Autonomia* – palvelut hallitsevat sisältämänsä logiikan.
- *Abstrahointi* – palvelut piilottavat ulkomaailmalta kaiken sellaisen logiikan, joka ei ole kuvattuna palvelukuvauksessa.
- *Uudelleenkäytettävyys* – logiikka on jaettu palveluihin, joiden tarkoituksena on olla uudelleenkäytettäviä.
- *Yhdistettävyys* – valmiista palveluista voidaan yhdistelemällä ja koordinoimalla koostaa uusia yhdistelmäpalveluita.
- *Tilattomuus* – palvelut minimoivat aktiviteettiä koskevan tiedon.
- *Löydettävyys* – palvelut on suunniteltu ulospäin suuntautuneiksi, jolloin niiden löytäminen ja hyödyntäminen on mahdollista.

Hau & al. (2008) tiivistävätkin palvelukeskeisen arkkitehtuurin erityispiirteiksi rajapintasuuntautuneisuuden, standardoinnin, itsenäisyyden ja modulaarisuuden sekä liiketoimintalähtöisyyden. *Rajapintasuuntautuneisuus* vastaa pitkälti olio-ohjelmoinnin kapselointia, ja se piilottaa palveluiden toteutukset niiden hyödyntäjiltä, mutta edellyttää tarkkaa rajapintamäärittelyä. *Standardointi* mahdollistaa palvelukomponenttien integroimisen uudeksi toiminnallisuudeksi ja sen avulla useampi komponentti voi hyödyntää samaa rajapintaa. *Itsenäisyys ja modulaarisuus* mahdollistavat sen, että komponenttia voidaan kutsua viestipohjaisia menetelmiä hyödyntäen, jolloin komponentti voi käsitellä viestin ilman riippuvuuksia muihin komponentteihin. Palvelukeskeisen arkkitehtuurin tavoittelema *liiketoimintalähtöisyys* tarkoittaa sitä, että palvelukeskeisessä arkkitehtuurissa liiketoimintaan liittyvät reaalimaailman liiketoimintaobjektit pyritään esittämään palveluna, esimerkkinä laskunmaksupalvelu.

Banke & al. (2005) lähestyvät palvelukeskeisen arkkitehtuurin määrittämistä määrittämällä ensin mitä tarkoitetaan ohjelmistoarkkitehtuureilla. *Ohjelmistoarkkitehtuuri* kuvaa ohjelmistokomponentit ja jakaa ohjelmiston toiminnallisuuden näihin komponentteihin. Se kuvaa komponenttien teknisen rakenteen, rajoitteet ja niiden väliset rajapinnat. *Arkkitehtuuri* on hahmotelma järjestelmästä, ja sen takia se toimii korkean tason suunnitelmana järjestelmän rakentamista varten. Banke & al. (2005) jatkavat palvelukeskeisen arkkitehtuurin määrityksellä: palvelukeskeinen arkkitehtuuri on ohjelmistoarkkitehtuuri, joka koostuu seuraavista avaintekijöistä: edustasovellus, palvelu, palveluhakemisto ja palveluväylä. Palvelu koostuu sopimuksesta, yhdestä tai useammasta rajapinnasta ja toteutuksesta. Asiaa havainnollistaa kuva 7.



Kuva 7 – Palvelukeskeisen arkkitehtuurin keskeisiä komponentteja (Banke & al., 2005)

Edustasovellukset (kuva 7) toimivat palveluprosessien käynnistäjänä ja kommunikaatiokanavana. Niiden rooli palvelukeskeisessä arkkitehtuurissa on tämän takia keskeinen. Edustasovelluksia on hyvin paljon erilaisia, esimerkkeinä selainpohjaiset käyttöliittymäsovellukset tai monipuolisemmat asiakasohjelmistot. On kuitenkin syytä huomata, ettei edustasovelluksella ole välttämättä ollenkaan käyttöliittymää, vaan se voi olla myös eräajoprosessi tai muu pitkäkestoinen prosessi.

Banken & al. (2005) mukaan palvelut (kuva 7) ovat komponentteja, joilla on yksilöivä tarkoitus ja ne yleensä koteloivat korkeamman tason liiketoimintakonseptin. Palvelut koostuvat *sopimuksesta*, joka määrittää palvelun sisällön. Sopimus kertoo yleensä palvelun tarkoituksen, toiminnallisuuden, rajoitteet ja miten palvelua käytetään. Palvelusopimuksen esitystapa riippuu pitkälti tarjottavasta palvelusta. Palvelulla voi olla myös formaali rajapintakuvaus, joka on kuvattu standardinmukaisella rajapintojenkuvauskielellä, kuten IDL (Interface Description Language) tai WSDL. Vaikka formaali rajapintakuvaus ei ole pakollinen, on sen käytöstä huomattavia etuja, koska se tuo mukanaan teknologiariippumattomuutta liittyen ajoympäristöön, väliohjelmistoihin, protokolliin ja ohjelmointikieliin. On kuitenkin syytä huomata, että varsinainen sopimus palvelun käytöstä sisältää yleensä muutakin kuin pelkän formaalin

palvelukuvauksen, esimerkiksi palvelun semantiikan kuvaaminen tai mahdollisista kustannuksista sopiminen on usein formaalin palvelukuvauksen ulkopuolella.

Palvelu tarjotaan palvelun asiakkaalle (edustasovellukselle) *rajapinnan* kautta (kuva 7). Vaikka palvelun sopimus kuvaa käytettävän rajapinnan, pitävät Banke & al. (2005) sitä keskeisenä palvelun osana, koska osa rajapinnan toteutuksesta kuuluu palvelun lisäksi myös palvelun asiakkaalle, ja mahdollisille väliohjelmistoille. Palveluun kuuluu myös sen *toteutus*, joka tarkoittaa palvelusopimuksen mukaista rajapinnan teknistä tuotosta. Toteutus tarjoaa palvelun liiketoimintalogiikan ja sen sisältämän tiedon, ja se voi koostua esimerkiksi yhdestä tai useammasta sovelluksesta, tietokannasta ja ohjaustiedostosta. *Liiketoimintalogiikka* on toteutuksen kapseloitu osa, joka tarjotaan normaalisti rajapintojen kautta. Palvelu voi sisältää myös *tietoa*, jota palvelun asiakas tarvitsee. Tieto voi tulla esimerkiksi tiedostosta tai tietokannasta.

Palveluhakemiston (kuva 7) tehtävänä on turvata palveluiden löydettävyys ja mahdollistaa niihin liittyvä tiedon jakaminen. Tiedon jakaminen kohdistuu usein ulkoisille sidosryhmille, kuten palvelun toteutusprojektin ulkopuoliset tahot. Palveluhakemistossa kuvataan usein palvelusopimuksen ulkopuolisia asioita, kuten palvelun sijaintipaikka, vastuuhenkilö, käytön hinnat, tietoturvaan liittyvät rajoitteet ja palvelutaso. Sinänsä palveluhakemisto ei ole välttämätön osa palvelukeskeistä arkkitehtuuria, mutta sen käyttö on erittäin hyödyllistä, koska se helpottaa palveluiden uudelleenkäyttöä. Palveluhakemisto ei myöskään välttämättä tarvitse ollenkaan tietoteknistä toteutusta, palveluhakemisto voi olla myös esimerkiksi ilmoitustaulu organisaation fyysisissä tiloissa. Banken & al. (2005) mukaan palveluhakemiston tulisi sisältää tiedot palvelusta, sen metodeista ja parametreista formaalissa kuvausmuodossa, joka tarkoittaa esimerkiksi WSDL ja XSD-muotoista kuvausta. Lisäksi palvelusta olisi syytä kuvata palvelun liiketoiminnallinen omistaja, tekninen omistaja ja tukihenkilö ongelmatilanteisiin liittyen. Palveluhakemistoon on myös syytä viedä käyttöoikeuksiin liittyvät asiat ainakin palvelun käyttöön ottamisen näkökulmasta, sekä kuvaus palvelun palvelutasosta, eli suunnitellut vasteajat ja palveluajat. Tämän lisäksi palveluhakemistoon kannattaa viedä tiedot liittyen palvelutapahtumien hallintaan, eli tiedot siitä mikä on tapahtuman luonne (luku/kirjoitus/muutos) ja mitä toimenpiteitä tapahtumaan liittyy (esimerkiksi tapahtuman keskeyttäminen tai peruuttaminen).

Palveluväylän (Enterprise Service Bus, ESB) tehtävänä on yhdistää palvelut ja edustasovellukset. Palveluväylä itsessään ei välttämättä koostu yhdestä teknologiasta, vaan se kokoaa

useita eri teknologioita yhdeksi kokonaisuudeksi. Palveluväylän keskeisimpinä ominaisuuksina Banke & al. (2005) pitävät liitettävyyttä, teknologian ja yhteystapojen monimuotoisuutta, sekä teknisiä palveluita, joita se tarjoaa. Liitettävyyden avulla palveluväylään on mahdollista tuoda uusia palveluita ja yhdistää niitä edustasovelluksiin. Teknologian monimuotoisuus mahdollistaa sen, että palvelut ja niiden kuluttajat voidaan toteuttaa eri ohjelmointikielillä, niillä voi olla esimerkiksi eri ajoympäristöt ja käyttöjärjestelmät. Yhteystapojen monimuotoisuus mahdollistaa protokollariippumattomuuden lisäksi myös erilaiset viestintämallit, kuten pyyntö-vastaus (request-response) ja julkaise-tilaa (publish & subscribe), sisältäen myös mahdollisuuden synkronisiin ja asynkronisiin viestinvaihtoihin. Tekniset palvelut sisältävät esimerkiksi lokituksen, tietoturvan, auditoinnin, transformoinnin ja tapahtumien hallinnan. Palveluväylää tarkastellaan tarkemmin kohdassa 3.3.

Palvelukeskeisen arkkitehtuurin hyötyinä pidetään (Hau & al., 2008) ketteryyttä, järjestelmien vähentynyttä kompleksisuutta, uudelleenkäytettävyyttä ja ohjelmistojen parempaa keskinäistä yhteensopivuutta. Ketteryys perustuu ajatukseen, että palvelukeskeisen arkkitehtuurin avulla voidaan olemassa olevista palveluista kerätä valmista toiminnallisuutta ja näitä palveluita yhdistelemällä muuttaa organisaation prosessia nopeammin, mikäli se altistuu sisäiselle tai ulkopuoliselle muutostarpeelle (esimerkiksi lakimuutos). Kompleksisuuden vähenemisen takana on ajatus siitä, että palvelun sisäisiä toteutusmuutoksia voidaan löyhän sidonnan ansiosta toteuttaa ilman, että palveluiden kuluttajat niitä edes huomaavat. Lisäksi palvelukeskeisen arkkitehtuurin mukaisissa järjestelmissä riippuvuudet muihin järjestelmiin ovat hallinnassa, koska ne on selkeästi kuvattu. Uudelleenkäytettävyys on mahdollista saavuttaa juuri palvelukuvausten ansiosta, koska niiden avulla on helpompi tunnistaa kohteet, joissa tarvitaan jo olemassa olevaa toiminnallisuutta. Ohjelmistojen parempi yhteensopivuus liittyy pitkälti siihen, että ohjelmistot voivat kommunikoida keskenään helpommin, koska niitä integroitaessa tarvitsee ottaa huomioon ainoastaan liittymät ja välikerrosohjelmistot.

3.3 Palveluväylä osana palvelukeskeistä arkkitehtuuria

Vaikka *palveluväylää* pidetään olennaisena osana palvelukeskeisen arkkitehtuurin toteutusta (Banke & al., 2005; Chappell, 2004), on sen määrittämistä pidetty haastavana (Richards, 2006). Termi juontaa juurensa Gartner Groupin raporttiin vuodelta 2002 (Schulte, 2002), jossa ESB:n kerrotaan yhdistävän Web-palvelut ja viestinvälitysohjelmistot lisäämällä niihin

toiminnallisuutta transformointiin ja reititykseen liittyen, ja sitä käytetään palvelukeskeisen arkkitehtuurin perustana. Palveluväylä määritelläänkin usein sen ominaisuuksien perusteella.

Chappellin (2004) mukaan ESB:ssä toimintaperiaatteena on pyrkimys yhdistää sovellukset ja tapahtumapohjaiset palvelut toisiinsa löyhän sidonnan periaatteella. ESB eroaa perinteisistä EAI-työkaluista olemalla täysin web-palvelupohjainen, sekä sisältämällä tuen erilaisille kommunikaatiomalleille. Perinteiset EAI-työkalut tukeutuvat keskeisesti tähtimallin mukaiseen (hub-and-spoke) arkkitehtuuriin, jossa törmätään usein ongelmiin organisaatiorajojen kohdalla. Tämä johtuu lähinnä siitä, että tähtimallin mukainen, keskitetty arkkitehtuuri ei salli paikallisten integraatiotoimialueiden alueellista hallinnointia. Tämän lisäksi monoliittisen EAI-järjestelmän sisäänrakennettu MOM (Message Oriented Middleware) voi rajoittaa ratkaisun käyttöönottoa eri verkkosegmenteissä. ESB:ssä molemmat, sekä viestiväylä että integrointiominaisuudet voivat olla hajautettuna. Lisäksi perinteisissä EAI-hankkeissa törmätään usein korkeisiin aloituskustannuksiin ja korkeaan oppimiskynnykseen. Chappell (2004) listaa ESB:n keskeisimmät ominaisuudet: läpinäkyvyys, standardeihin pohjautuva integraatio, laajasti hajautettu integraatio, hajautettu tiedon muunnettavuus, laajentaminen kerrosteisten palveluiden avulla, tapahtumakeskeinen SOA, prosessipohjaisuus, tietoturva ja luotettavuus, itseohjautuva ympäristö, etäkonfigurointi ja -hallinta, reaaliaikaisuus sekä mahdollisuus asteittaiseen käyttöönottoon.

Läpinäkyvyys tarkoittaa kykyä hyödyntää ja tarjota palveluita myös organisaatiorajoista riippumatta. Sovellukset liittyvät palveluväylään, jolloin ne voivat hyödyntää toistensa palveluita ja tietosisältöä. Vaikka palveluväylä on web-palvelupohjainen, ei kaikkia sovelluksia ole välttämättä pakko muuttaa hyödyntämään web-palveluita, vaan liittäminen voidaan suorittaa esimerkiksi suoraan tiedonsiirtoprotokollilla ja -palveluilla, kuten esimerkiksi File Transfer Protocol (FTP) sekä Java Message Service (JMS). *Standardeihin pohjautuva integraatio* mahdollistaa erilaisten teknologioiden, kuten tiedonsiirtoprotokollien ja ohjelmointikielien tukemisen. Standardien rajapintojen kautta voidaan tukea myös valmistajakohtaisia sovelluksia ja hyödyntää niissä olevaa tietosisältöä. Esimerkkinä tästä J2EE Connector Architecture (JCA)-määrityksen (JSR 016, 2001) mukaiset sovellusadapterit. Perinteiset EAI-työkalut olivat yleensä valmistajasidonnaisia, ja tuki standardeille oli hyvin puutteellinen.

Toisena keskeisenä erona perinteisiin EAI-välineisiin ESB:ssä on mahdollisuus *laajasti hajautettuun integraatioon.* Perinteiset EAI-välineet ovat tuottaneet samantyyppisiä palveluita kuin ESB, kuten reititys, prosessien hallinta, tiedon muunnokset ja sovellusadapterit. ESB:n avulla näiden komponenttien tarjoamat palvelut on mahdollista hajauttaa, toisin kuin EAI-työkaluissa, jotka ovat perinteisesti olleet hyvin vahvasti keskitettyjä, monoliittisia järjestelmiä. Lisäksi ESB:n kautta on mahdollista tehdä palveluiden valikoivaa jakelua, eli palveluita voidaan ottaa helposti käyttöön yksi kerrallaan ESB-ratkaisujen liitettävyyden avulla.

Keskeistä ESB-ratkaisussa on myös pyrkimys *hajautettuun tiedon muunnettavuuteen*, jossa erilaisten sovellusten tietotarpeet saadaan täytettyä muunnospalveluiden avulla. Muunnospalvelut mahdollistavat viestien ja tietosisällön muokkaamisen sovelluskohtaisiin tarpeisiin. Käyttämällä palveluväylää voidaan toiminnallisuutta asteittain *laajentaa kerrosteisten palveluiden* kautta. Esimerkiksi perusintegraatiopalveluita voidaan laajentaa myöhemmin ottamalla käyttöön liiketoimintaprosessien hallintaa (Business Process Management, BPM), vaikka työnkulkuihin, yhtenä palveluväylän palveluna.

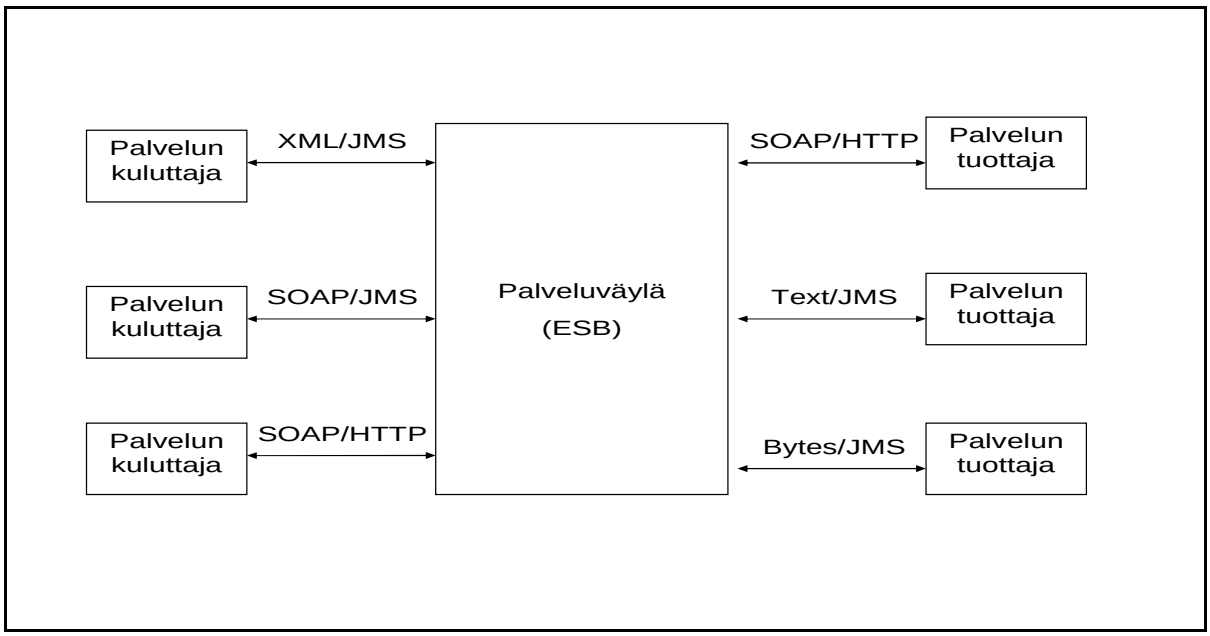
Tapahtumakeskeisessä SOA:ssa palvelut jaetaan abstrakteihin päätepisteisiin, jotka voivat käsitellä asynkronisia tapahtumia. Palvelun toteutuksessa ei tarvitse huomioida viestin reititykseen tai viestin vastaanottajaan liittyviä asioita, sillä näistä palveluista huolehtii ESB. Palvelut vain vastaanottavat ja käsittelevät viestejä, ja lähettävät niitä ESB:lle. Keskeinen ominaisuus palveluväylässä on *prosessipohjaisuus*, joka voi vaihdella toteutuksesta riippuen yksinkertaisesta säännöstöstä monimutkaiseen prosessien orkestrointiin jollakin korkean tason kielellä, kuten Business Process Execution Language (BPEL). Ajatuksena on kuitenkin kuvata liiketoimintaprosessi siten, että samaan prosessiin voidaan liittyä organisaatiorajoista tai fyysisestä sijainnista riippumatta.

ESB tarjoaa palveluita myös *tietoturvaan ja luotettavuuteen* liittyen. ESB:t tarjoavat usein välineitä viestiliikenteen salaamiseen, esimerkiksi WS-Securityn (2006) avulla. Lisäksi ESB:hen kohdistuva liikenne voidaan ohjata tarvittaessa palomuurin läpi, ja yleensä ESB:n ja siihen liitettyjen sovellusten kytkemisen yhteydessä on mahdollista hyödyntää tunnusten hallinta- ja tunnistamispalveluita sekä yhteistä pääsynhallintaa. Viestiliikenteen luotettavuus toteutetaan hyödyntämällä jotakin viestinvälitysohjelmistoa (Message Oriented Middleware, MOM), kuten esimerkiksi JMS. MOM tarjoaa asynkroniset kommunikointitavat, luotettavan liiketoimintatiedon perille toimittamisen ja transaktioiden yhtenäisyyden.

ESB:n avulla on mahdollista toteuttaa *itseohjautuva, mutta hallittu ympäristö*, jossa organisaation yksiköt itsenäisesti tuottavat ja hyödyntävät hajautetusti palveluita omaan tahtiinsa. Tällöin organisaatioyksiköt eivät ole riippuvaisia yhteisestä integraatiokeskittimestä (integration hub), vaan ne huolehtivat omista tarpeistaan ja IT-resursseistaan, kuten sovelluksista. ESB:n avulla kukin yksikkö voi asentaa, hallita, turvata, konfiguroida tarvitsemansa viestiliikenteen ja integraatiokomponentit paikallisesti ja silti liittää palvelunsa osaksi suurempaa yhteistä jaettua integraatioverkkoa. *Etäkonfigurointi ja -hallinta* mahdollistaa puolestaan hallinnan keskittämisen, mikäli se nähdään tarpeelliseksi.

XML-pohjaisuus mahdollistaa tiedon hyödyntämisen sovellusten välillä, sekä liiketoiminta-objektien formaalin kuvaamisen. ESB:ssä on mahdollista käyttää palveluita, jotka kokoavat tietosisältöä useista sovelluksista uusiksi näkymiksi, ja tarvittaessa viestien tietosisältöä voidaan rikastaa ESB:ssä tiedonjaon edistämiseksi. ESB:n avulla on myös *toteutettavissa reaaliaikainen* näkymä liiketoimintaan ja sen tapahtumiin, joka tukee päätöksentekoa tarjoamalla faktoihin perustuvaa tietoa. ESB voidaan myös *käyttöönottaa asteittain*, ja se soveltuu hyvin myös pieniin projekteihin.

Palveluväylä tukee yleensä erilaisia tietoliikenne- ja viestiprotokollia (kuten JMS tai HTTP) palveluiden liittämiseksi palveluväylään (Tost, 2006). Sen tehtävänä on toimia välittäjänä palvelun tuottajan ja kuluttajan välillä, tiedonsiirtotavasta riippumatta. Asiaa havainnollistaa kuva 8.



Kuva 8 – Esimerkki palveluväylän erilaisista protokollasidoksista (Tost, 2006)

ESB:n sisäiseen komponenttimalliin on kaksi osittain päällekkäistä ratkaisua, Java Business Integration (JBI) ja Service Component Architecture (SCA). SCA (2007) on suunniteltu kehittäjän näkökulmasta, kun taas JBI standardoi komponenttien toteutustavan, jolloin se on mielenkiintoinen erityisesti välikerrosohjelmistojen valmistajien näkökulmasta katsottuna (Edwards, 2007). Tässä tutkimuksessa keskitytään JBI-standardiin, jota tarkastellaan kohdassa 5.3.

4 XML-POHJAISISTA INTEGROINTITEKNIIKOISTA

XML-pohjaisten standardien kehittyminen on heijastunut myös integraatiomenetelmien kehitykseen. XML mahdollistaa siirrettävän tiedon kuvaamisen XML-skeeman tai DTD (Document Type Definition)- tiedoston avulla. XML-pohjainen tieto voidaan muuttaa myös helposti toiseen muotoon XSL:n (Extensible Stylesheet Language) avulla. Prosessipohjainen integraatio perustuu käytännössä XML-pohjaisiin menetelmiin (Rosendahl & Rune, 2000).

Koska XML alkaa kuitenkin olla yleisesti tiedossa oleva standardi (esim. Tahvanainen, 2006), ei tässä tutkimuksessa käydä sitä läpi kuin ainoastaan välttämättömin osin (kohta 4.1). Tämän jälkeen (kohdassa 4.2) tutustutaan lyhyesti RSS-sisältömuotoon, joka toimii yksinkertaisena sisältörakenteena luvussa 6 esitettävän esimerkkisovelluksen tarpeisiin. Web-palveluiden perustekniikat käydään läpi kohdassa 4.3, jonka jälkeen kohdassa 4.4 tutustutaan web-palveluiden tietoturvaan ja yhteensopivuuteen. Sen jälkeen tarkastellaan sisällönhallintajärjestelmien integroinnissa keskeistä JSR 170 -rajapintaa (kohta 4.5).

Web-palveluihin liittyy myös muita standardeja, jotka laajentavat tässä tutkimuksessa esitettyä web-palveluiden perustoiminnallisuutta. Oheiset standardit ovat enenevissä määrin vaikuttamassa web-palveluiden kehitykseen, mutta kunkin osa-alueen laajuuden takia niiden syvällisempi käsittely rajataan ulos tästä tutkimuksesta.

4.1 XML-skeemat

XML-skeemat ovat W3C:n XML-standardiin (W3C, 2006) kuuluva osa, jonka avulla dokumentissa voidaan kuvata sen käyttämät tietotyypit. Käytännössä tämä mahdollistaa dokumenttien tarkistamisen sisällön suhteen laajemmin kuin on mahdollista käytettäessä XML-skeemoja edeltänyttä DTD-dokumenttien kuvauskieltä (W3C, 2004c). XML-skeemakielen tulee (W3C, 2004c):

a) tarjota primitiiviset tietotyypit, kuten esimerkiksi kokonaisluvut tai merkkijonot
b) kuvata riittävä tyyppijärjestelmä tiedon siirtämiseksi relaatiotietokantaan ja -kannasta
c) eriyttää tiedon kuvaaminen varsinaisesta tietojoukosta
d) mahdollistaa käyttäjän itse määrittelemät ja kuvaamat tietotyypit, jotka voivat mahdollisesti koostua aikaisemmin määrätyistä tietotyypeistä ja niiden arvoalueesta.

Kuvassa 9 on esitelty yksinkertainen XML-tiedosto, joka käyttää kuvassa 10 olevaa XML-skeemaa tietotyyppien kuvaamiseen. Kyseiset XML-esimerkit on toteutettu lähdettä W3Schools (2006) mukaillen.

```
<?xml version="1.0" encoding="UTF-8"?>
<viesti xmlns="http://cs.joensuu.fi/~jhirvone/esimerkki"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance":schemaLocation="http://cs.joensuu.fi/~jhirvone/esimerkki
esimerkkiSkeema.xsd">
<lahettaja>Jukka</lahettaja>
      <kenelle>Pekka</kenelle>
      <otsikko>Terve!</otsikko>
      <sisalto>Terveiset Espoosta!</sisalto>
</viesti>
```

Kuva 9 - XML esimerkkitiedosto, joka hyödyntää kuvan 10 XML-skeemaa

Kuvan 10 skeemaa voidaan käyttää kuvan 9 sisällön validointiin. Kyseinen skeema kertoo viesti-tietotyypin koostuvan neljästä kentästä, jotka ovat tekstimuotoisia primitiivityyppejä.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://cs.joensuu.fi/~jhirvone/esimerkki"
elementFormDefault="qualified">
      <xs:element name="viesti">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element name="lahettaja" type="xs:string"/>
                        <xs:element name="kenelle" type="xs:string"/>
                        <xs:element name="otsikko" type="xs:string"/>
                        <xs:element name="sisalto" type="xs:string"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

Kuva 10 - Esimerkki XML-skeemasta

4.2 RSS

XML-pohjaisia sisällöntuotantojärjestelmiin liittyviä tekniikoita ovat muun muassa uutisten välityksiin käytettävät uutis-/sisältövirrat, kuten Really Simple Syndication (RSS) ja Resource Description Framework (RDF). RSS oli aiemmin Netscapen kehittämä kuvausformaatti (täl-

löin se tunnettiin nimellä Rich Site Summary), jonka avulla voitiin tuoda portaaliin ulkoisen sisällöntuottajan toimittamia uutisia ja muita artikkeleja. RSS-formaatista on useita versioita, jotka poikkeavat rakenteeltaan jonkin verran toisistaan, joka hankaloittaa käyttöä integrointityössä (Pilgrim, 2002). RSS:n etuja ovat kuitenkin tunnettavuus ja yksinkertaisuus, joka helpottaa käyttöönottoa. Kuvassa 11 on esimerkki RSS versiosta 2.0.

```
<?xml version="1.0" encoding="UTF-8"?>
<rss:rss xmlns:rss="http://blogs.law.harvard.edu/RSS20.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://blogs.law.harvard.edu/RSS20.xsd RSS20.xsd ">
      <channel version="2.0">
            <title>Minun kanava</title>
            <link>http://cs.joensuu.fi/~jhirvone</link>
            <description>Esimerkki RSS-tiedostosta</description>
            <copyright>Jukka Hirvonen</copyright>
            <item>
                  <title>Google</title>
                  <link>http://www.google.com</link>
                  <description>Hakukone</description>
            </item>
            <item>
                  <title>Yahoo</title>
                  <link>http://www.yahoo.com</link>
                  <description>Toinen hakukone</description>
            </item>
      </channel>
</rss:rss>
```

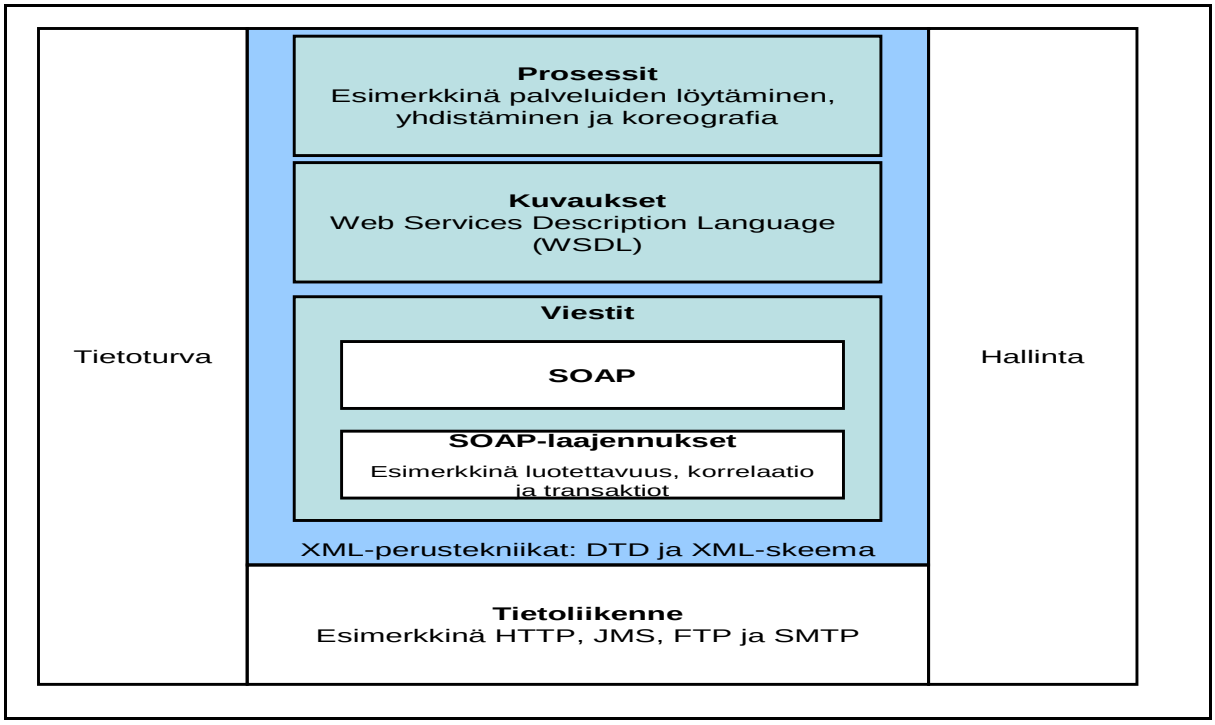
Kuva 11 - Esimerkki RSS-tiedostosta (versio 2.0)

RSS-formaatti jakautuu kahteen merkittävään päähaaraan (Wikipedia, 2008). Alkuperäinen, Netscapen kehittämä versio 0.90, ja RSS-DEV-työryhmän julkaisemat versiot 1.0 ja 1.1 muodostavat ensimmäisen 1.x-haaran. Toisen, 2.x-haaran muodostavat versio 0.91, sen laajentavat versiot 0.92 - 0.94 sekä versio 2.0.1, jonka mukana RSS-lyhenne muuttui tarkoittamaan ”Really Simple Syndication”. Haarat eivät ole keskenään yhteensopivia, esimerkiksi versio 1.0 (Swartz, 2000) pohjautuu W3C:n (2004a) RDF-suosituksen versioon 1.0, kun taas RSS versio 2.0 käyttää suoraan XML-nimiavaruuksia. Tästä huolimatta suurin osa esimerkiksi RSS-lukijaohjelmistoista tarjoaa tuen molemmille versiohaaroille. Kuten nimestäkin voidaan jo päätellä, ei RSS sellaisenaan sovellu kovinkaan syvälliseen sisällön integrointiin, koska se on tarkoitettu lähinnä linkkityyppisen informaation esittämiseen

4.3 Web-palveluiden tekniikat

Web-palvelut (web services) ovat kokoelma XML-pohjaisia tekniikoita, jotka edesauttavat sovellusten suunnittelua ja toteutusta. Web-palveluiden etuina verrattuna perinteisiin EAI-tekniikoihin on niiden suunnittelun, kehityksen, ylläpidon ja käytön yksinkertaisuus ja avoimien standardien tuoma levinneisyys. Muita etuja ovat joustavuus, edullisuus, dynaamisuus sekä mahdollisuus jakaa integraatio pienempiin, asteittaisesti toteutettaviin osakokonaisuuksiin (Sadhwani & Samtani, 2002).

Web-palvelut tarjoavat työkalut sovellusten kutsumiseen, kuvaamiseen, löytämiseen ja julkaisuun internet-verkossa. Yleiskuva web-palveluihin liittyvistä tekniikoista löytyy kuvasta 12.



Kuva 12 – Yleiskatsaus web-palveluiden tekniikoihin (W3C, 2004b)

4.3.1 WSDL

Web Services Definition Language (WSDL) on web-palveluiden kuvaamiseen käytetty rakenteinen XML-formaatti (W3C, 2001). WSDL:n avulla voidaan kuvata palveluiden sisältämät toiminnot kootusti ja samalla voidaan piilottaa niiden sisäiset toteutustavat. WSDL ei ota kantaa varsinaiseen viestien siirtotiehen tai sisältöön, jolloin sitä voidaan käyttää tehokkaasti SOAP:in tai HTTP -protokollan kanssa. WSDL-dokumentti määrittää *palvelu*n koostuvan

verkon päätepisteiden kokoelmista, eli *porteista*. Koska WSDL:ssä päätepisteiden ja viestien abstraktit kuvaukset on eriytetty niiden konkreettisesta siirtotiestä ja tietotyyppisidoksista, mahdollistaa tämä viestikuvausten uudelleenkäytön. *Viesti* on kuvaus vaihdettavasta tiedosta ja *porttityypit* ovat *operaatioiden* kokoelmia. Operaatiot koostuvat palvelun toimintojen abstrakteista kuvauksista. *Sidos* koostuu tietyn porttityypin protokolla- ja tietosisällön kuvauksesta. Portti kuvataan kiinnittämällä sidokseen verkko-osoite, ja palvelu koostuu kokoelmasta portteja. *Tyypit* sisältävät tietotyyppien kuvaukset jollakin tyyppijärjestelmällä kuten esimerkiksi XML-skeemoilla (W3C, 2004c).

Kuvassa 13 on yksinkertainen WSDL-kuvaus, josta havaitaan edellä mainitut palvelukuvauksen olennaiset osat. Itse palvelukuvauksessa kiinnitetään ensin nimiavaruudet, luodaan kaksi viestityyppiä ja näille porttityypit ja sidokset. Lopuksi kuvataan itse palvelu ja määrätään mistä osoitteesta palvelu löytyy. Käytännössä parempi tapa Haajasen (2004) mukaan on jakaa WSDL-kuvaus omiin tiedostoihin, joissa yhdessä kuvataan tietotyypit, toisessa abstraktit palvelukuvaukset ja kolmannessa konkreettiset palveluiden sidokset. Tämä parantaa olemassa olevien palveluiden uudelleenkäytettävyyttä ja helpottaa kuvausten hallintaa.

WSDL:n käyttötapa on pitkälti tarvesidonnainen. Mikäli ollaan tekemässä uutta sovellusta, voidaan WSDL:ää käyttää rajapinnan kuvauksena ja tämän jälkeen generoida automaattisesti sekä asiakaspuolen asiakaskantaluokat (stub) että palvelinrunkoluokat (skeleton). Mikäli kyseessä on olemassa oleva toteutus, on WSDL mahdollista generoida yleensä tuon olemassa olevan toteutuksen pohjalta.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="tns:http://cs.joensuu.fi/jhirvone/tervepalvelu"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="tns:http://cs.joensuu.fi/jhirvone/tervepalvelu"
xmlns:intf="tns:http://cs.joensuu.fi/jhirvone/tervepalvelu"
xmlns:tns1="http://dto.tervepalvelu.jhirvone.cs.joensuu.fi"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:types>
  <schema elementFormDefault="qualified"
        targetNamespace="http://dto.tervepalvelu.jhirvone.cs.joensuu.fi"
        xmlns="http://www.w3.org/2001/XMLSchema">
   <complexType name="TervePalveluData">
    <sequence>
     <element name="arvo" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
  </schema>
  <schema elementFormDefault="qualified"
        targetNamespace="tns:http://cs.joensuu.fi/jhirvone/tervepalvelu"
        xmlns="http://www.w3.org/2001/XMLSchema">
   <import namespace="http://dto.tervepalvelu.jhirvone.cs.joensuu.fi"/>
   <element name="data" type="tns1:TervePalveluData"/>
   <element name="sanoTerveReturn" type="xsd:string"/>
  </schema>
 </wsdl:types>
   <wsdl:message name="sanoTerveResponse">
      <wsdl:part element="impl:sanoTerveReturn" name="sanoTerveReturn"/>
   </wsdl:message>
   <wsdl:message name="sanoTerveRequest">
      <wsdl:part element="impl:data" name="data"/>
   </wsdl:message>
   <wsdl:portType name="TervePalveluImpl">
      <wsdl:operation name="sanoTerve" parameterOrder="data">
         <wsdl:input message="impl:sanoTerveRequest" name="sanoTerveRequest"/>
         <wsdl:output message="impl:sanoTerveResponse" name="sanoTerveResponse"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="TervePalveluSoapBinding" type="impl:TervePalveluImpl">
      <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="sanoTerve">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="sanoTerveRequest">
            <wsdlsoap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="sanoTerveResponse">
            <wsdlsoap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="TervePalveluImplService">
      <wsdl:port binding="impl:TervePalveluSoapBinding" name="TervePalvelu">
         <wsdlsoap:address location="http://localhost:8080/axis/services/TervePalvelu"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

Kuva 13 - Yksinkertainen WSDL-kuvaustiedosto

4.3.2 SOAP

SOAP (Simple Object Access Protocol) on yksinkertainen XML-pohjainen protokolla, jonka avulla voidaan välittää viestipohjaista tietoa eri järjestelmien välillä (W3C, 2007). SOAP on ympäristöriippumaton protokolla, jonka avulla tiedonvaihto onnistuu eri sovellusten välillä,

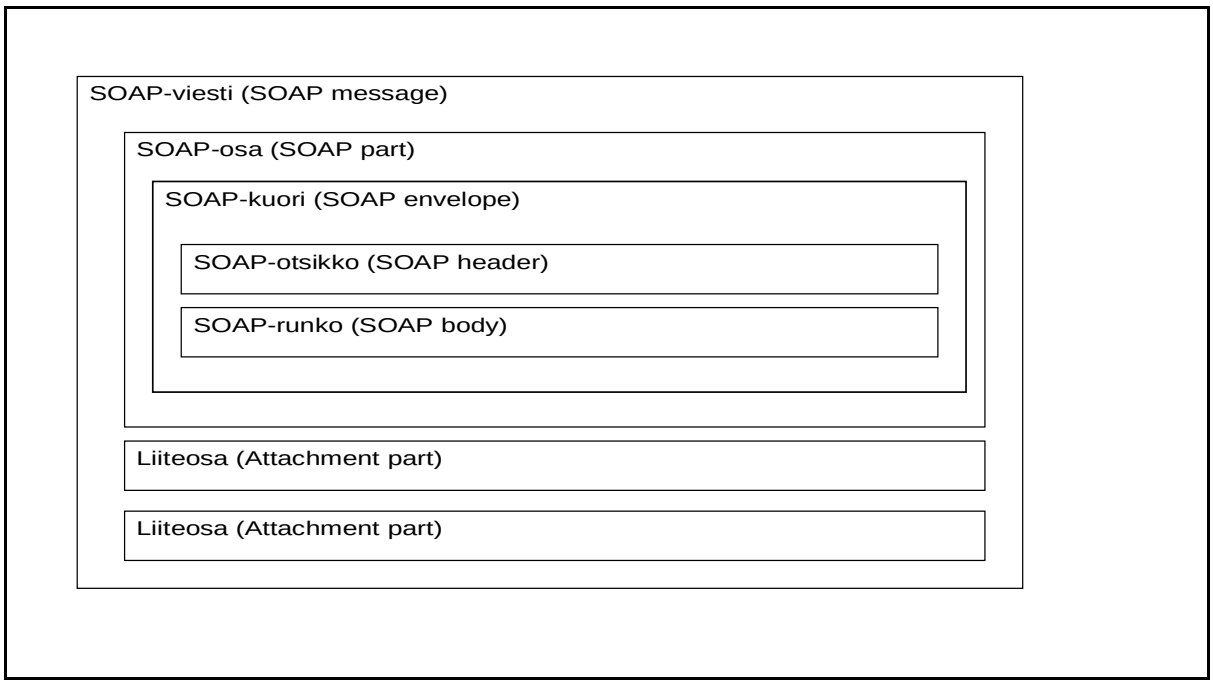
niiden toteutustekniikasta tai käyttöympäristöstä riippumatta. SOAP ei itsessään ole mikään uusi keksintö, se vastaa pitkälti esimerkiksi CORBA-tyyppisiä RPC-kutsuja, mutta sen etuina pidetään yleisesti helppokäyttöisyyttä, avoimuutta ja soveltuvuutta internet-käyttöön (Kiviniemi, 2002). Kuvassa 14 on yksinkertainen esimerkki SOAP-sanomasta.

SOAP on toimintaperiaatteeltaan yksisuuntainen protokolla, mutta sovellukset voivat rakentaa sen varaan monimutkaisempia vuorovaikutuskäytäntöjä, kuten pyyntö/vastaus tai pyyntö/useampia vastauksia. SOAP ei itsessään ota kantaa sen avulla välitettävään tietoon, siirtotien luotettavuuteen tai palomuureihin, vaan sen tarkoituksena on tarjota helposti laajennettava tapa vaihtaa tietoa kahden tai useamman sovelluksen välillä. Lisäksi se kertoo tarvittavat toimenpiteet, jonka SOAP-solmun on tehtävä, kun se vastaanottaa SOAP-viestin (W3C, 2007).

```
<?xml version="1.0" encoding="utf-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:ns0="http://cs.joensuu.fi/~jhirvone/TervePalvelu"
      env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <env:Body>
      <ns0:sanoTerve>
            <String_1 xsi:type="xsd:string">Jukka</String_1>
      </ns0:sanoTerve>
   </env:Body>
</env:Envelope>
```

Kuva 14 – Yksinkertainen SOAP-sanoma

SOAP-viesti muodostuu *SOAP-kuoresta* (SOAP envelope), joka pitää sisällään erillisen, valinnaisen alkuosion eli *otsikon* (SOAP header) ja varsinaisen *viestirungon* (SOAP body). Lisäksi viestiin voi kuulua yksi tai useampi liiteosa, joiden olemassaolo ei ole pakollista. SOAP-viestin rakennetta havainnollistaa kuva 15. Vapaaehtoisissa otsikkotiedoissa yleensä kerrotaan viestin välittämiseen liittyviä asioita viestin lähetysketjussa, ja viestirungossa on varsinaiselle vastaanottajalle välitettävä varsinainen asiasisältö. Viestiotsikko voi muuttua lähetysketjun aikana, ja SOAP-protokolla ei itsessään ota kantaa sen sisältöön tai edes viestin reititykseen viestiketjussa. Vastaanottajan edes ei välttämättä tarvitse ymmärtää otsikkoa, mikäli viestissä ei sitä erikseen vaadita (mustUndestand-atribuutti).



Kuva 15 – SOAP-viestin rakenne (FCS Partners, 2005)

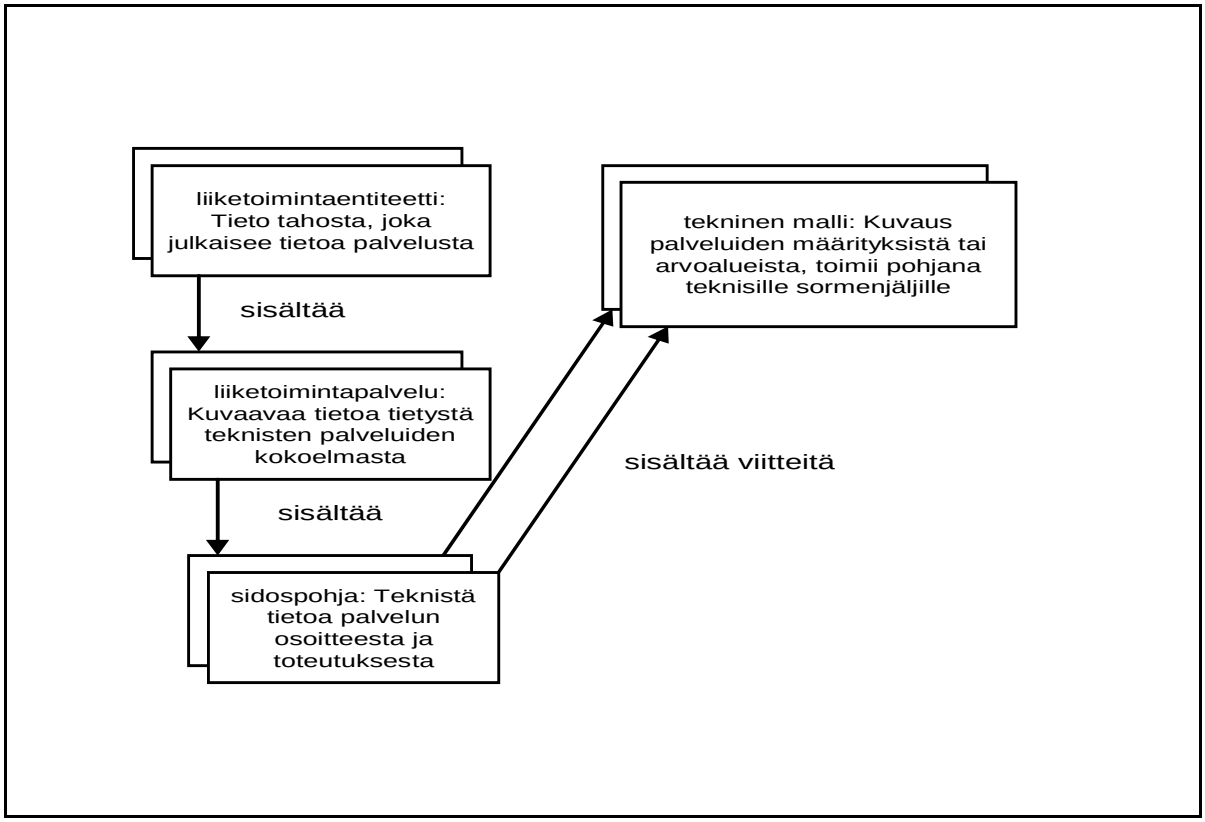
4.3.3 UDDI

Web-palveluiden peruskulmakiviin kuuluu myös Universal Description, Discovery and Integration (UDDI), joka määrittää standardin tavan julkaista ja löytää verkkopohjaisia ohjelmistokomponentteja palvelukeskeisessä arkkitehtuurissa (Oasis, 2004a). Standardi määrittää protokollan, jonka avulla voidaan kommunikoida web-palveluista koostuvan rekisterin kanssa ja mekanismit tietueiden jakamiseen ja delegointiin toisten rekisterien kanssa. Toisin sanoen, UDDI tarjoaa standardinmukaiset menetelmät palvelun löytämiseen, palvelun käyttöönottamiseen ja palvelun metadatan hallintaan.

UDDI mahdollistaa palveluiden helpon hallittavuuden ja parantaa niiden uudelleenkäyttöä. Sen avulla voidaan julkaista tietoa web-palveluista ja toteuttaa niille organisaatiokohtainen kategorisointi. Lisäksi palveluita voidaan hakea, joko organisaation sisältä tai sen ulkopuolelta, annetuilla hakutekijöillä. UDDI:ssa voidaan kuvata palveluiden turvallisuuteen liittyviä asioita, kuten palvelun käyttämiseksi vaadittava protokolla tai sen käynnistämiseen vaadittavat parametrit. Myös palveluiden vikasietoisuutta voidaan kehittää UDDI:n avulla, eristämällä sovellukset palveluiden muutoksista (tarjoamalla vikasietoinen ja älykäs reititys) tai niiden virhetilanteista (Oasis, 2004b)

UDDI:n (Oasis, 2004b) tietomalli koostuu pysyvien tietomallien kuvauksista eli entiteeteistä. Entiteetit kuvataan XML:n avulla ja talletetaan pysyvästi UDDI-solmuihin. Enteettien mahdollisia tyyppejä ovat:

- *liiketoimintaentiteetti* (businessEntity), joka kuvaa organisaation, yrityksen tai tahon, joka tarjoaa web-palveluita
- *liiketoimintapalvelu* (businessService), joka kuvaa toisiinsa liittyviä web-palveluita, jotka kuuluvat liiketoimintaentiteetille
- *sidospohja* (bindingTemplate), joka kuvaa tarvittavan tiedon tietyn web-palvelun käyttämiseksi
- *tekninen malli* (tModel), joka on palvelutyypin teknisen määrityksen abstraktio; se organisoi palvelun tyyppitiedon ja asettaa sen saataville rekisteristä
- *julkaisukokoelma* (publisherAssertion), joka kuvaa liiketoimintaentiteettien suhteen toisiinsa, yhden liiketoimitaentiteetin kannalta katsottuna
- *tilaus* (subscription), joka kuvaa voimassa olevan pyynnön kertoa muutoksista siihen liittyville entiteeteille.



Kuva 16 - UDDI:n tietomalli (Oasis, 2004b)

Kyseisten entiteettien välisiä suhteita havainnollistaa kuva 16, jonka mukaan, liiketoimintaentiteetti tarjoaa liiketoimintapalveluita, jotka sisältävät teknisen ja liiketoiminnallisen kuvauksen tarjotuista palveluista. Liiketoimintapalvelut sisältävät sidospohjia, jotka viittaavat määrityksiin tai muihin palvelun teknisiin kuvaksiin. Sidospohjat viittaavat tietomalleihin tai teknisiin määrityksiin, joita tarvitaan kyseisen palvelun käyttöönottamiseksi. Kukin sidospohja viittaa yhteen tai useampaan tModel-palvelutyyppiin, joka kuvaa muun muassa kuka julkaisi mallin, kategoriat, joihin palvelu kuuluu ja viitteet esimerkiksi rajapintakuvauksiin, viestityyppeihin tai viestiprotokolliin. Kun UDDI:n perustietotyyppejä tallennetaan UDDI-rekisteriin, jokaiselle tietotyypille määrätään yksilöllinen tunniste, jota kutsutaan UDDI-avaimeksi.

Eräs tärkeä UDDI:n piirre on se, että se mahdollistaa web-palveluista rekisteriin tallennetun tiedon semanttisen rakenteen. UDDI:ssa käyttäjät voivat kategorisoida palveluita ja niihin liittyvää tietoa samanaikaisesti usealla eri tavalla, ja kategorisointeja voidaan lisätä rekisteröinteihin myös jälkikäteen.

4.4 Web-palveluiden tietoturvasta ja yhteensopivuudesta

Web-palveluiden tietoturvaan liittyviä standardeja on useita, keskeisimpänä mainittakoon Web Services Security (WS-Security, WSS). WS-Security (2006) standardin ideana on viedä web-palveluiden tietoturva siirtotietasolta viestitasolle. Yleensä web-palvelut luottavat siirtotien tuomaan tietoturvaan (Transport Layer Security, TLS), ja koska HTTP-protokolla on yleisimmin käytetty siirtotie web-palveluissa, luotetaan palveluissa usein vielä tänä päivänä HTTP-perusautentikointiin (HTTP-basic authentication) ja SSL-tekniikoihin (Shin, 2003). Tämä aiheuttaa ongelmia, mikäli viestit liikkuvat viestiketjuissa, koska kaikkien viestin välittävien osapuolten välillä tulee olla luotettu yhteys. Lisäksi ongelmia teettää koko palveluketjun luotettavuus, murtautumalla yhteen palveluun saadaan vapaat kädet kommunikointiin muiden palveluiden kanssa. Siirtotien salaaminen voi aiheuttaa myös sen, että mahdollisten hyökkäysten havaitseminen voi olla vaikeata, koska tällöin myös hyökkäykset tulevat kryptattuina.

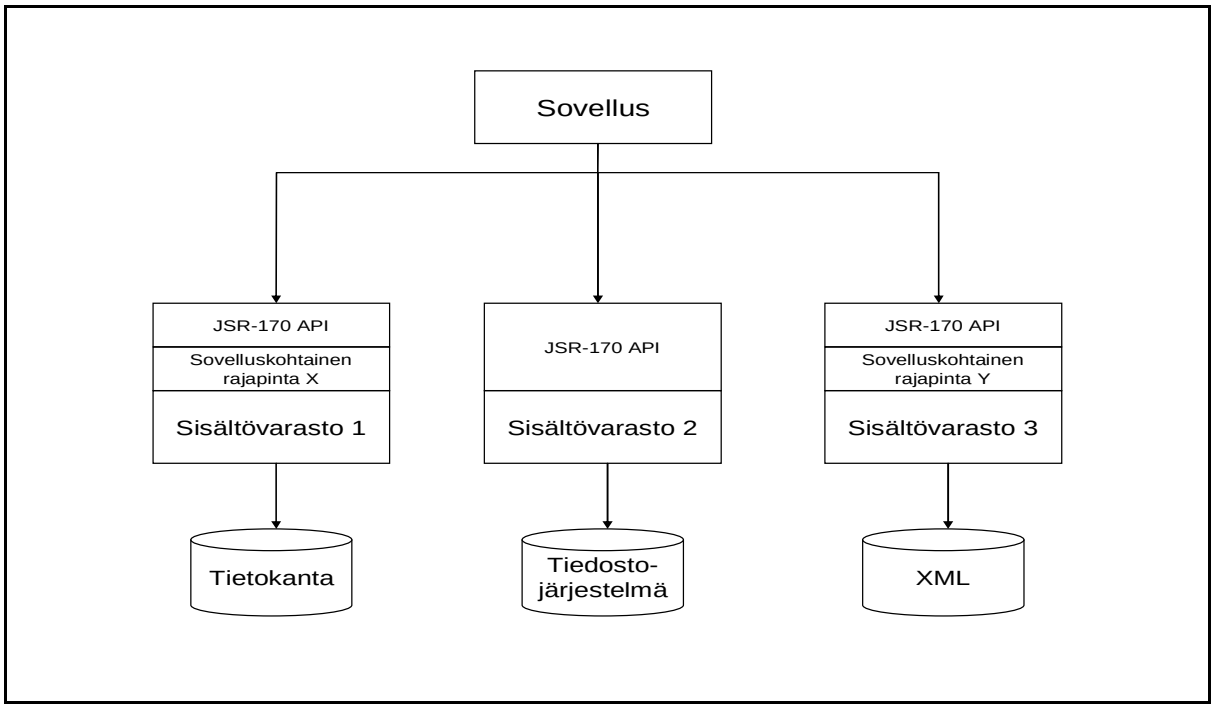
WS-Security (2006) perustuu ajatukseen, jonka mukaan suojaamalla pelkkä yksittäinen viesti koko viestiliikenneprotokollan sijaan päästään huomattavasti kevyempään lopputulokseen. Tässä ajatusmallissa tietoturvaan liittyvä informaatio kulkee viestin mukana, ja mikäli viestin välittäjällä tai vastaanottajalla ei ole tarvittavaa tietoturvainfrastruktuuria ja se ei ole luotettu, pysyy viesti salattuna ja se voidaan välittää seuraavalle vastaanottajalle. WS-Security (2006) toteuttaa viestiliikenteen salaamisen lisäämällä kolmitasoisen tietoturvan SOAP-viesteille. *Tunnistusvaltuuksien* (Authentication Tokens) avulla asiakasohjelmistojen on mahdollista lähettää vakioiduilla tavoilla SOAP-otsikossa käyttäjätunnus ja salasanapareja tai X.509 sertifikaatteja tunnistusta varten. Lisäksi WS-Security versiossa 1.1 on mahdollisuus esimerkiksi Kerberos- ja SAML (Security Assertion Markup Language)-valtuuksien käyttöön. *XML-salakirjoitus* (XML-encryption) mahdollistaa SOAP-viestirungon salaamisen XML-Encryption (2002) standardin mukaisesti. Yleensä tiedon salaamiseen on mahdollista käyttää useita erityyppisiä algoritmeja, kuten RSA-MD5 tai RSA-SHA1. *XML:n sähköinen allekirjoittaminen* (XML Digital Signatures) on WS-Security standardissa mahdollista W3C:n XML Digital Signature (2002) standardin avulla. XML:n sähköisen allekirjoittamisen avulla vastaanottaja voi varmistua siitä, että viesti on tullut vain ja ainoastaan sen oikealta lähettäjältä. Tyypillisesti sähköinen allekirjoitus hoidetaan laskemalla tarkistussumma viestin sisällöstä, mikäli viestiä on muutettu, muuttuu digitaalinen allekirjoitus epäkelvoksi.

Nykyisin viestiliikenteen salaamiseen liittyvä välinetuki on vaihtelevaa, mutta uusimmissa sovelluspalvelimissa on mahdollista suojata web-palveluiden viestit käyttämällä edellä mainittuja menetelmiä. Haasteita viestiliikenteen salaukselle tuottaa mahdolliset eri valmistajien omat tulkinnat osin keskeneräisistä standardeista, mikä tuo mukanaan yhteensopivuusongelmia. Yhteensopivuusongelmien minimoinniksi on perustettu avoin organisaatio WS-I (Web Services Interoperability Organization), jonka tehtävänä on edesauttaa yhteen toimivien palvelujen tuottamista.

4.5 JSR 170 – Rajapinta sisältövarastoille

JSR 170 (2005) (Java Specification Request) kuvaa standardin tavan käyttää erilaisia sisältövarastoja (tunnetaan myös nimellä Java Content Repository, JCR). Se pitää sisällään esimerkiksi sisältövaraston ohjelmallisen kutsumisen vakiomuotoista Java-rajapintaa hyödyntämällä. Vakiomuotoinen rajapinta sallii erilaisten, valmistajakohtaisten sisältövarastojen helpon hyödynnettävyyden vaikkapa järjestelmäintegraattorien toimesta. Lisäksi se mahdollistaa asiak-

kaalle sisältövarastojen vaihdettavuuden ilman, että niihin integroituja sovelluksia tarvitsee muuttaa. Asiaa havainnollistaa kuva 17.

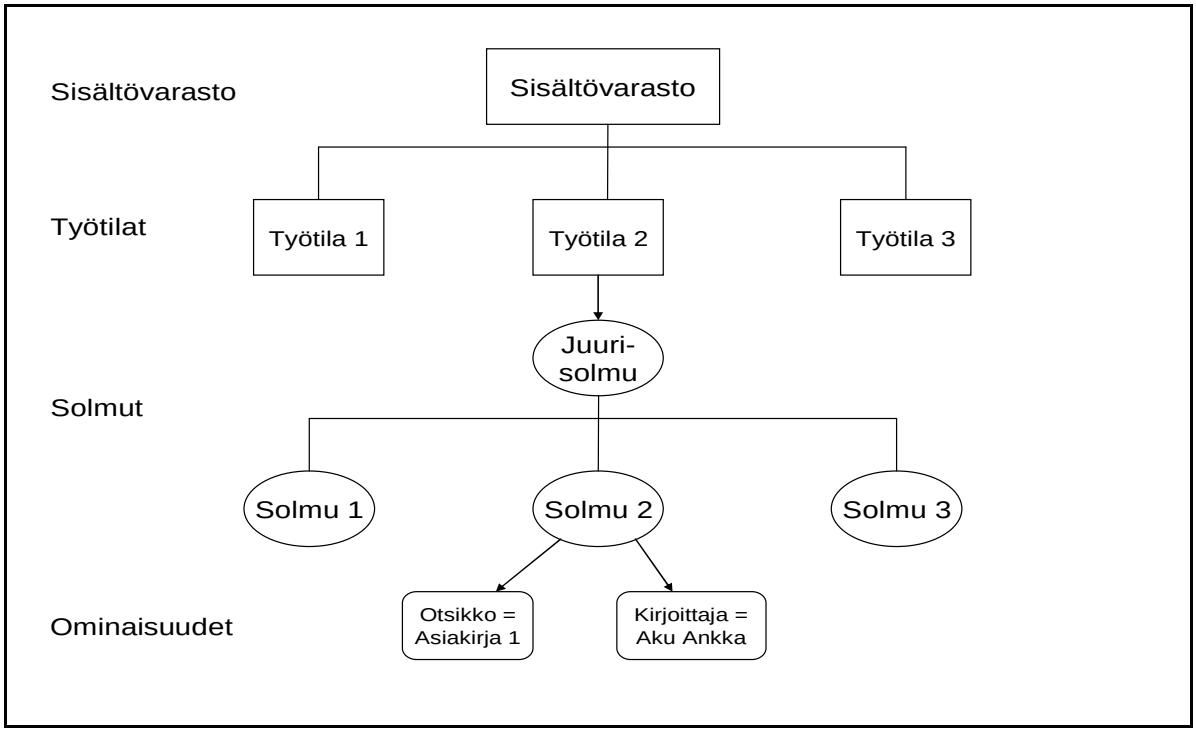


Kuva 17 – Sisältövarastot ja JSR 170 -rajapinta (Patil, 2006)

API:n suunnittelussa on noudatettu seuraavia periaatteita (JSR 170, 2005):

- *Riippumattomuus arkkitehtuurista, tietolähteestä tai protokollasta*: API on itsessään Java-rajapinta, mutta se voidaan toteuttaa eri tavoin. Lisäksi se mahdollistaa sekä *hierarkkisen, polkuihin* perustuvan sisältöjen osoituksen että *suoran, uniikkiin avaimeen pohjautuvan* osoituksen.
- *Helppous ohjelmoijan näkökulmasta*: Rajapinta on suunniteltu yksinkertaiseksi ja suoraviivaiseksi, ja sen yksinkertainen objektimalli keskittyy vain olennaiseen.
- *Sallii helpon toteutuksen niin monelle olemassa olevalle sisältövarastolle kuin mahdollista.* Rajapinnan suunnittelussa on otettu huomioon suurimpien toimittajien ratkaisut, ja liittäminen niihin on todettu suhteellisen helpoksi.
- *Vakioi myös hieman monimutkaisempia toiminnallisuuksia, joita tarvitaan edistyneemmissä sisältökeskeisissä sovelluksissa*. Koska tämän vaatimuksen ja edellisten välillä on ristiriitaa, on rajapinta jaettu kahteen yhteensopivuustasoon sekä joukkoon valinnaisia toiminnallisuuksia.

Ensimmäinen taso sisältää toiminnallisuuden vain varastosta lukemista varten. Tämä sisältää varaston sisällön lukemisen lisäksi sisältötyyppien määritysten introspektion, nimiavaruuksien perustuen, sisällön muunnon XML-muotoon ja hakutoiminnallisuudet. Toiminnallisuus kattaa suurimmalta osaltaan portaalisovellusten ja sisällön esityssivupohjien tarpeet. Ensimmäinen taso on myös suunniteltu helposti toteutettavissa olevaksi lisäksi olemassa oleviin sisältövarastoihin. Toinen taso pitää sisällään toiminnallisuuden sisältövarastoon kirjoittamiseksi, asettamisen sisällölle, edistyneemmän tuen nimiavaruuksille ja sisällön tuonnin XML-muotoisen aineiston pohjalta. Lisäksi määrityksessä on kuvattu valinnaisia ominaisuuksia, joita soveltuva varasto voi tukea, kuten transaktiot, sisällön versiointi, tarkkailu, pääsynhallinta, lukitukset ja hakujen erityispiirteet.



Kuva 18 – JCR-sisältövaraston hierarkia

Sisältövarasto on hierarkkinen (rakennetta havainnollistaa kuva 18) kokonaisuus, joka jakautuu erilaisiin työtiloihin (Barik, 2005). Työtilat koostuvat *alkioista*, jotka voivat olla joko *ominaisuuksia* tai *solmuja*. Jokaisella solmulla voi olla nolla tai useampia lapsia tai niihin voidaan liittää nolla tai useampia ominaisuuksia. Jokaisella solmulla on vain ja ainoastaan yksi ensisijainen solmutyyppi, joka määrittää solmun piirteet, kuten mitä ominaisuuksia ja mitä lapsia sillä saa olla. Tämän lisäksi solmulla voi olla yksi tai useampi sekatyyppi (mixin),

jotka määrittävät lisäpiirteitä (esimerkiksi versioitavuus, lukittavuus, viitattavuus) solmulle. Juurisolmu on ainoa solmu, jolla ei ole vanhempaa ja muilla solmuilla on vain ja ainoastaan yksi vanhempi. Ominaisuuksilla on yksi vanhempi, ja niillä ei voi olla lapsia, sillä ne ovat puun lehtiä. Kaikki varsinainen sisältö talletetaan ominaisuuksiin (Patil, 2006).

Myös JCR-sisältövarastosta on tulossa uudempi versio, 2.0, jota työstetään Java-määritys-pyyntönumerolla JSR 283 (2007). Uudessa versiossa pyritään standardoimaan piirteitä sisältövaraston hallinnointiin liittyen, kuten pääsynhallinta, työtilojen ja solmujen hallinta, sekä sisällön tai sisältövaraston rakennusmenetelmät kestävän kehityksen kannalta. Lisäksi samassa yhteydessä pyritään parantamaan sisältövarastojen keskinäistä yhteensopivuutta lisäämällä standardeja solmutyyppejä, mukaan lukien solmutyypit, jotka soveltuvat metatiedon välittämiseen ja lokalisointiin. Lisäksi versiossa kehitetään jo olemassa olevia piirteitä ja toiminnallisuuksia.

5 WEB-PALVELUIDEN TOTEUTTAMINEN JAVALLA

Web-palveluiden toteuttamiseksi Javalla on nyttemmin syntynyt erilaisia työkaluja ja kirjastoja, jotka helpottavat web-palveluiden toteutusta ja käyttöönottoa. Tässä luvussa tarkastelemme yleisimpiä menetelmiä web-palveluiden toteuttamiseen.

5.1 JAX-RPC

Java-ohjelmointirajapinta XML-pohjaisille etäaliohjelmakutsuille (JAX-RPC) helpottaa web-palveluiden rakentamista (Armstrong & al., 2005). Se määrittää suhteet XML-tyyppien ja Java-tyyppien välille, piilottaen varsinaisen XML-sanomatoteutuksen ohjelmoijalta metodikutsujen taakse. Palvelinpuolella kehittäjä yksilöi julkaistavat etäaliohjelmat määrittämällä kutsuttavat metodit Java-rajapinnan avulla. Lisäksi kehittäjä toteuttaa yhden tai useamman luokan, jotka toteuttavat kyseiset metodit. Asiakasohjelmat toteutetaan luomalla paikallinen proxy-objekti, joka esittää palvelua, ja tämän jälkeen yksinkertaisesti kutsutaan kyseisen proxy-objektin metodeja. JAX-RPC:n avulla kehittäjien ei tarvitse generoida etäkutsujen edellyttämiä SOAP-viestejä tai niiden vastauksia, sillä se tapahtuu JAX-RPC-ajoympäristön toimesta.

JAX-RPC:n avulla toteutetut palvelut ja niiden asiakkaat ovat ympäristöriippumattomia, eli toteutettuja palveluita voidaan periaatteessa hyödyntää mistä ohjelmointiympäristöstä tahansa. Vastaavasti myös JAX-RPC:llä toteutetuilla asiakkailla voidaan kutsua palvelua, joka on toteutettu muulla kuin Java-kielellä. JAX-RPC nojautuu W3C:n standardeihin (HTTP, SOAP ja WSDL).

Kuten mainittu, JAX-RPC määrää Java-ohjelmointikielen tyyppien väliset suhteet XML/WSDL-määrityksiin. Esimerkiksi JAX-RPC määrää relaation `java.lang.String`-luokan ja `xsd:string`-tietotyypin välille. Vaikkakin tuki erilaisille tietotyypeille on sangen kattava, ei JAX-RPC tue täysin kaikkia luokkia, jotka kuuluvat Java-standardiin. Puutteistaan huolimatta (Loughran & Smith, 2005) on JAX-RPC:stä ja sen toteutuksista tullut laajalle levinnyt tapa toteuttaa web-palveluita Java-ympäristöön.

5.2 Axis-sovelluskehys

Apache Axis (2008) on laajalle levinnyt, avoimeen lähdekoodiin perustuva web-palveluiden generointiin, kehittämiseen ja ajamiseen tarkoitettu sovelluskehys. Se toteuttaa SOAP-rajapinnan, pitäen sisällään toteutuksen JAX-RPC standardista. Axis-kehyksen avulla kehittäjä voi kirjoittaa Java-ohjelmakoodia ja julkaista sen web-palveluna, tai kehittäjä voi ottaa valmiin WSDL-kuvauksen ja generoida sen ja Axis-kehyksen avulla palvelusta sekä asiakaskantaluokat että palvelinrunkoluokat. Ensimmäistä lähestymistapaa nimitetään kokoavaksi (bottom-up) kehittämiseksi, ja jälkimmäistä osittavaksi (top-down) lähestymistavaksi (Flurry & Modh, 2005).

Kokoavassa palvelukehityksessä toteutetaan ensin tiedonsiirto-olio (Data Transfer Object, DTO) ja sitä käyttävä puhdas Java-luokka (Plain Old Java Object, POJO), josta generoidaan WSDL-kuvaus. Kuvassa 19 on tiedonvälitysobjekti TervePalveluData, jota käyttää kuvassa 20 esitetty TervePalveluImpl.

```
package fi.joensuu.cs.jhirvone.tervepalvelu.dto;
public class TervePalveluData {

	private String arvo;
	public String getArvo() {
		return arvo;
	}
	public void setArvo(String arvo) {
		this.arvo = arvo;
	}
}
```

Kuva 19 – TervePalveluData.java - tiedonsiirto-olio

Tiedonsiirto-olio kuvaa palvelun tietotyypit, joilla oletetaan olevan attribuutteja vastaavat asettaja- ja hakija-metodit, jotta niistä voidaan generoida tietotyyppejä vastaava skeema. Sinänsä tiedonsiirto-olion käyttö ei ole pakollista, mutta se eriyttää tyypitykset varsinaisesta toteutuksesta ja mahdollistaa objektin tietosisällön päivittämisen vain tarpeen mukaan. Tämä on usein hyödyllistä, kun halutaan minimoida verkon aiheuttama viive tiedonsiirrossa.

```
package fi.joensuu.cs.jhirvone.tervepalvelu;
import fi.joensuu.cs.jhirvone.tervepalvelu.dto.*;
public class TervePalveluImpl {
      public String sanoTerve(TervePalveluData data)
      {
            String tervehdys = null;
            if (data !=  null){
                  tervehdys = "Terve: " + data.getArvo() + "!";
            }
            return tervehdys;
      }
}
```

Kuva 20 – TervePalveluImpl.java – toteutus TervePalvelusta

Varsinainen palvelukuvauksen generointi tapahtuu Axisin mukana tulevan Java2WSDL-ohjelman avulla. Kuvan 13 mukaisen WSDL-tiedoston generointi voidaan toteuttaa komennolla:

```
java org.apache.axis.wsdl.Java2WSDL
      -o TervePalvelu.wsdl -y"DOCUMENT" -u"LITERAL"
      -n"tns:http://cs.joensuu.fi/jhirvone/tervepalvelu"
      -e"fi.joensuu.cs.jhirvone.tervepalvelu.dto.TervePalveluData"
      -l"http://localhost:8080/axis/services/TervePalvelu"
      fi.joensuu.cs.jhirvone.tervepalvelu.TervePalveluImpl
```

Kokoavan palvelukehityksen etuina pidetään (Flurry & Modh, 2005) seuraavia asioita:

- olemassa olevan koodin julkaisu web-palveluna on nopeaa
- kehittäjän ei tarvitse ymmärtää WSDL:n rakennetta tai XML:ää, koska ne generoidaan työkalujen avulla
- työkalutuki on erinomainen.

Menetelmässä on myös varjopuolia (Flurry & Modh, 2005):

- Generoitujen skeemojen tietotyypit vastaavat vain ja ainoastaan Java-luokan tietotyyppejä, tällöin standardinmukaisten rajapintaskeemojen käyttö on hankalaa.
- Jos tietotyypit eivät ole yksinkertaisia tiedonsiirto-objekteja, ei niiden käsittelyssä tarvittava liiketoimintalogiikka välity skeeman mukana.
- Generoitu skeema sisältyy WSDL-kuvaukseen, joka tekee skeeman jälleenkäytön hankalaksi.
- Palvelin- ja asiakastoteutusta ei voida kehittää yhtä aikaa, palvelintoteutus on tehtävä ensin, jotta kuvaus voidaan generoida.

- Inkrementaalisten muutosten tekeminen ja hallinta on vaikeaa, koska muutokset generoinnin lähteenä toimivassa luokassa saattavat vaikuttaa merkittävästi WSDL-dokumenttiin, joka tuottaa hankaluuksia asiakaspäässä.
- Nimiavaruudet vastaavat toteutuksen pakettirakennetta, ja pakettirakenteen muutos tekee skeemojen nimiavaruuksista yhteensopimattomia. Työkalut mahdollistavat yleensä nimiavaruuksien ja pakettirakenteen vastaavuuksien asettamisen erikseen, mutta tämä on tehtävä jo generointivaiheessa.

Osittamalla tapahtuvassa kehittämisessä WSDL-kuvaus toimii rajapinnan määrityksenä sekä asiakkaan että palvelintoteutuksen toteuttajalle. Tällöin molemmat osapuolet voivat tehdä kehitystyötä samanaikaisesti, ja tämä helpottaa myös inkrementaalisten muutosten tekemistä. Tehtäessä kehitystä osittavana palvelukehityksenä, on hyödyllistä kuvata tietotyypit omassa erillisessä XML-skeemassa, jota WSDL-kuvaus hyödyntää. Tämä helpottaa merkittävästi tietotyyppien uudelleenkäyttöä esimerkiksi jossakin toisessa web-palvelussa. Lisäksi generoitujen tiedonsiirto-objektien pakettirakenne vastaa WSDL-kuvauksen nimiavaruutta, josta se voidaan tarvittaessa sitoa toteutusta vastaavaan pakettirakenteeseen sekä palvelin- että asiakaspäässä vaikuttamatta rajapinnan toimintaan.

Haittoina osittavassa kehittämisessä pidetään sitä, että palvelukuvauksen ja skeemojen toteuttaminen vaatii syvällistä ymmärrystä XSD- ja WSDL-standardeista. Tietämystä tarvitaan, vaikka hyödynnettäisiin nykyaikaisia työkaluja, sillä yhteen toimivien ja tehokkaiden palvelukuvausten tuottaminen ei ole triviaalia. Lisäksi työkaluissa ei ole välttämättä vielä kovinkaan hyvä tuki kyseiselle menetelmälle, mutta tilanne on merkittävästi parantumassa. Kuitenkin, nykyään osittavaa palvelukehittämistä pidetään suositeltuna lähestymistapana.

Esimerkkinä osittavasta palvelukehityksestä TervePalvelun palvelinrunkoluokkien generointi Axis-kehyksen mukana tulevan WSDL2Java -ohjelman avulla kuvassa 13 esitetystä WSDL-kuvausdokumentista tapahtuu komennolla:

```
java org.apache.axis.wsdl.WSDL2Java --server-side --skeletonDeploy true TervePalvelu.wsdl
```

Kyseinen komento luo palvelukuvauksen jokaiselle sidokselle palvelinrunkoluokan, joka toimii pohjana varsinaiselle toteutukselle. Lisäksi komento luo tiedostot palvelun asentamiseksi

Axis-ajoympäristöön. Usein Axis sisältyy myös erilaisiin sovelluskehittimiin (esimerkkinä Eclipse (2008)), jolloin toteutuksia voidaan generoida nopeasti muutamalla hiiren klikkauksella.

5.3 Java-liiketoimintaintegraatio

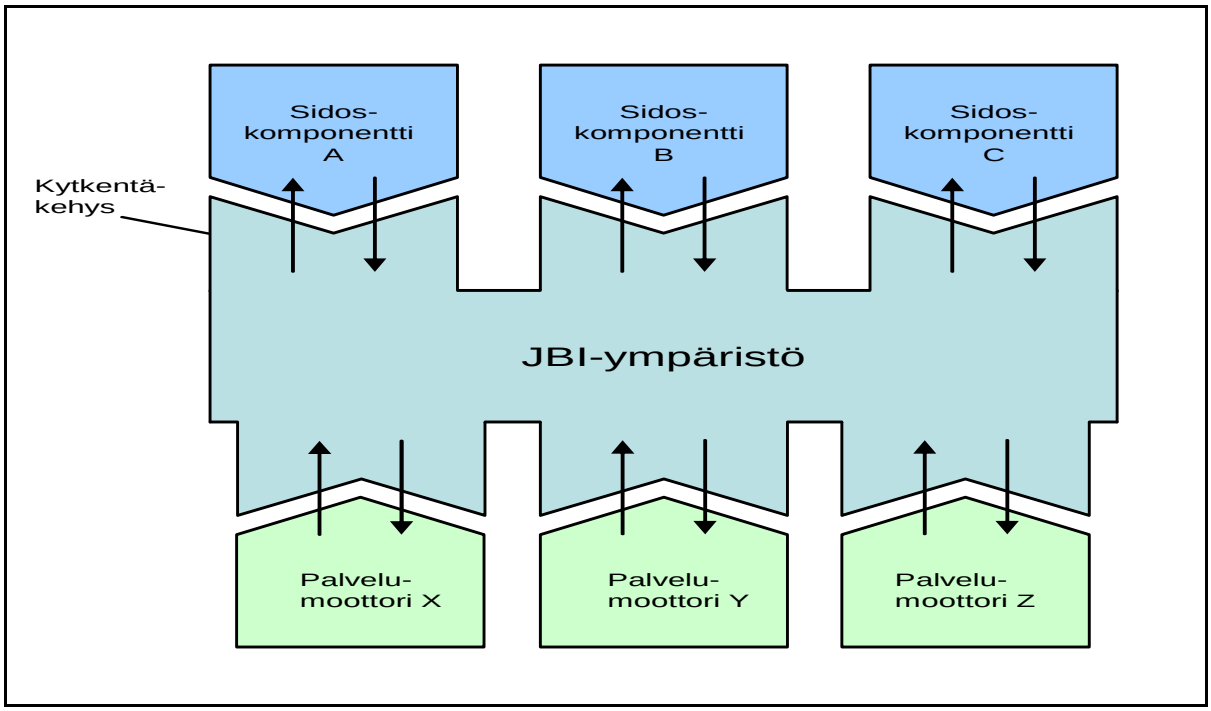
Java-liiketoimintaintegraatio (Java Business Integration, JBI) on pyrkimys standardoida integraatio-ohjelmistojen arkkitehtuuri, ja se toimii usein ESB-ratkaisun sisäisenä toteutustapana (esimerkiksi Apache Servicemix (2008)). JBI-standardin mukainen toteutus muodostaa ympäristön, johon eri osapuolet voivat tuottaa ja kytkeä erilaisia komponentteja, jotka käyttäytyvät odotetulla ja luotettavalla tavalla. Kyseinen Java-standardi (JSR 208, 2005) määrittää arkkitehtuurin, jonka avulla integraatio-ohjelmistoon voidaan liittää komponentteja, joita kutsutaan erilaisilla viestinvälitysmenetelmillä. JBI määrittää komponenttien kytkentärajapinnan kun taas komponentti määrää mitä rajapintoja JBI voi käyttää. Komponenttien välinen keskustelu tapahtuu JBI:n määrittämän välikerroksen avulla, joka huolehtii reitityksestä. Komponentit eivät siis kommunikoi suoraan keskenään, joka mahdollistaa komponenttien välisen löyhän sidonnan. Kommunikaatio on myös luonteeltaan asynkronista, koska palvelun kuluttaja ja tuottaja eivät jaa yhteistä säiettä, joka myös edesauttaa sidonnan löyhentämistä.

JBI-komponenttien tuottamat palvelut on kuvattu WSDL-kuvauskielellä (versio 1.1 tai 2.0). Komponentit on jaettu kahteen osaan:

- *Sidoskomponentit (*Binding Component, BC*)*, joiden avulla JBI-ympäristöön voidaan liittää ulkoisia palveluita, kuten muita sovelluksia. Tällaisia ovat esimerkiksi protokollat tai sovelluskohtaiset adapterit, jotka voivat olla toteutettu myös muulla kuin Java-ohjelmointikielellä.
- *Palvelumoottorit (*Service Engine, SE), jotka tarjoavat liiketoimintalogiikkaa ja muunnospalveluita muille komponenteille, ja ne voivat toimia myös tällaisten palveluiden kuluttajina. SE:t ovat yleensä Javalla toteutettuja komponentteja

JBI-ympäristön arkkitehtuuria havainnollistaa kuva 21. Sinänsä palvelumoottorit eivät eroa merkittävästi sidoskomponenteista, erottelu on tehty lähinnä käytännön syistä, jotta liiketoi-

mintalogiikka saadaan mahdollisimman helposti erotettua protokollista ja sidoksista. Molemmat komponenttityypit voivat toimia palvelun tuottajana tai kuluttajana tai molempina.



Kuva 21 – JBI-arkkitehtuuri (JSR 208, 2005)

JBI-ympäristön päätoiminto on normalisoitujen viestien välittäminen komponenttien välillä. Normalisoitu viesti kostuu abstraktista XML-viestistä ja siihen liittyvästä metatiedosta. WSDL- kuvaa abstraktin viestin rakenteen ja rajoitteet, yleensä XML-skeeman avulla. Viestit jaetaan kahteen kategoriaan, normaali- (normal) ja vikaviestit (fault). Lisäksi se kuvaa abstraktit operaatiot, eli mitä tietoja palvelun ja kuluttajan välillä vaihdetaan. Abstraktissa operaatiossa kuvataan operaation nimi, viestinvaihtomenetelmä (Message Exchange Pattern, MEP) ja viestityypit. Abstrakti palvelutyyppi kokoaa toisiinsa liittyvä abstraktit operaatiot yhteen. Palvelutyyppi saa rajapinnan nimen, jota käytetään palvelutyypin tunnistamiseen. Palvelutyyppejä voidaan määrittää myös muiden palvelutyyppien laajennuksena.

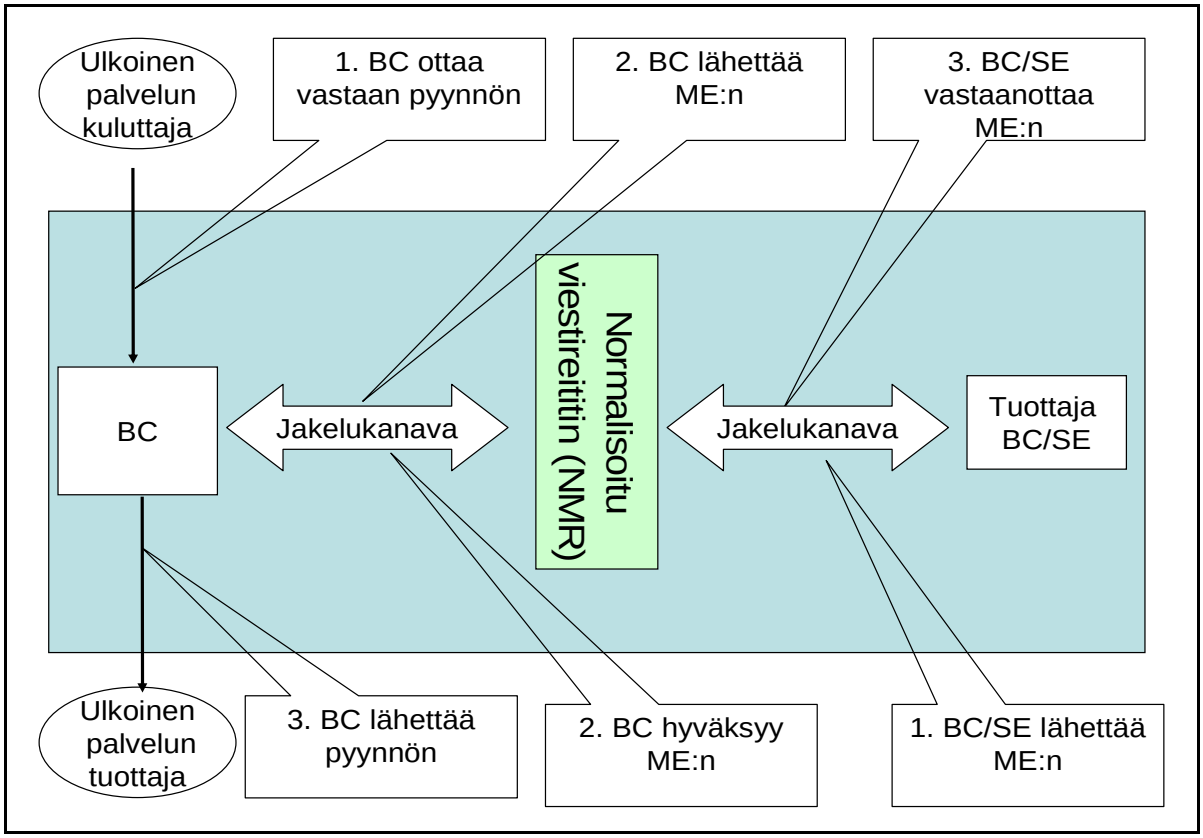
Lisäksi WSDL-yhteensopivuuden takia JBI-määritys kuvaa konkreettisen palvelumallin, joka määrittää sidostyypit, päätepisteet, jotka koostuvat päätepisteen nimestä ja sidostyypistä, sekä palvelun, joka koostuu palvelun nimestä, palvelun tyypin nimestä ja päätepisteistä. Yleensä tätä kokonaisuutta kutsutaan *palvelun päätepisteeksi*, ja yleensä sitä käytetään päätepisteen tunnistamiseen. Sinänsä JBI perustuu abstraktin palvelun malliin, jossa ei kuvata palveluiden käyttämiä protokollia ja niiden sidoksia, ja joka puolestaan mahdollistaa protokollariippu-

mattomuuden. Tämän takia konkreettisesta palvelumallista puuttuu tiedonsiirtoprotokollaan liittyvät asiat, protokollana käytetään JBI-pohjaisia viestintäsopimuksia.

Normalisoidussa viestinvaihdossa JBI-ympäristö reitittää viestejä komponentilta toiselle (kuva 22). Sidoskomponentit muuttavat vastaanottamansa viestit normalisoituun muotoon (toisin sanoen niistä poistetaan protokolliin ja tiedonsiirtomenetelmiin liittyvä informaatio). Sidoskomponentit (BC) ja palvelumoottorit (SE) kommunikoivat normalisoidun viestireitittimen (Normalized Message Router, NMR) kanssa jakelukanavaa hyödyntäen. Jakelukanava tarjoaa kahdensuuntaisen jakelusopimuksen viestin vastaanottoon ja toimittamiseen. Muunnettuaan viestin normalisoituun muotoon, sidoskomponentti luo viestinvaihdon (Message Exchange, ME), joka on säiliö erilaisissa viestinvaihtomenetelmissä tarvittaville viesteille. Viestinvaihtoon BC lisää metatietoa suoritettavasta palvelusta ja toiminnosta, ja lähettää sen NMR:lle jakelukanavaansa hyödyntäen. NMR valitsee metatiedon perusteella viestinvaihdolle sopivan palveluntarjoajan ja reitittää ME:n sen perusteella. Palveluntarjoajan tulee hyväksyä ME jakelukanavalta.

Vastaavasti myös vastaukset tai ulospäin kohdistuvat palvelupyynnöt menevät saman kaavan mukaisesti (kuva 22). Tällöin BC/SE tuottaa normalisoidun viestin, ja asettaa sen uuteen ME:hen. ME osoitetaan palvelun päätepisteelle, ei varsinaiselle komponentille. ME lähetetään ja hyväksytään aiemmin kuvatun mukaisesti. Hyväksymisen jälkeen BC muuttaa normalisoidun viestin protokollasidonnaiseen muotoon ja lähettää sen ulkoiselle palvelulle.

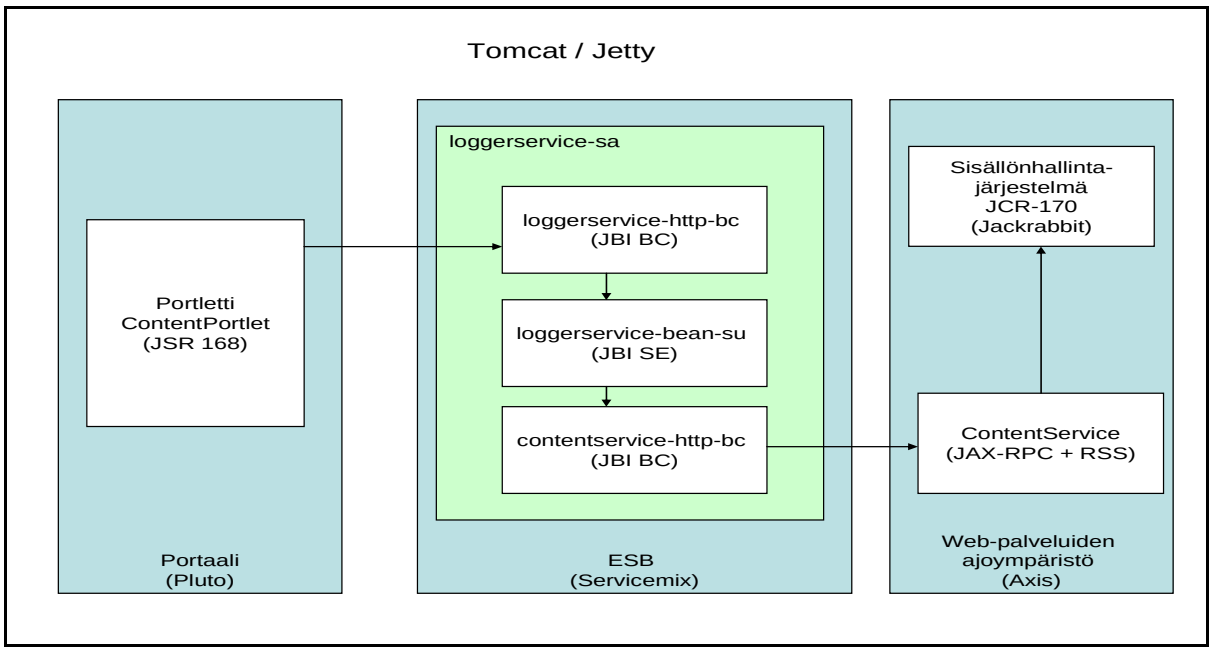
JBI-standardin (JSR 208, 2005) mukaan JBI-ympäristön hallinta tapahtuu Javan hallintalaajennusten (Java Management Extensions, JMX (2008)) avulla. JBI tukee hallintalaajennusten kautta seuraavia toiminnallisuuksia: komponenttien asennus, komponenttien elinkaaren hallinta, komponenttiartefaktien dynaaminen jakelu, sekä valvonta ja hallinta. Näiden toiminnallisuuksien tarkempi tarkastelu ei kuulu tämän tutkimuksen piiriin.



Kuva 22 - Normalisoitu viestinvaihto normalisoidun viestireitittimen kautta (JSR 208, 2005)

6 ESIMERKKISOVELLUS

Aiemmin käsittelimme portaaleja, sisällönhallintajärjestelmiä ja palvelukeskeistä integraatiota. Tässä luvussa keskitytään siihen, miten kyseiset asiat saadaan konkreettisesti yhdistettyä. Ensin asennetaan portaali (Apache Pluto) sovelluspalvelimelle ja luodaan portletti, joka asennetaan portaaliin. Sen jälkeen asennetaan ja konfiguroidaan sisällönhallintajärjestelmäksi sisältövarasto (Apache Jackrabbit). Tämän jälkeen teemme Axisin avulla web-palvelun, joka hakee tietoa sisältövarastosta. Lopuksi konfiguroimme palveluväylän (Apache ServiceMix), joka yhdistää portletin ja web-palvelun. Esimerkkisovellus on J2EE-arkkitehtuurin mukainen sovellus, jonka arkkitehtuuri on esitetty kuvassa 23. Käytetty Javan versio on 1.5 ja sovelluspalvelimena toimii pääosin Apache Tomcat 5.5 (poikkeuksena ServiceMix, joka Apache Mavenin kautta ajettaessa käyttää Jettyä (2008) HTTP-palvelimena).



Kuva 23 – Esimerkkisovelluksen arkkitehtuuri

Liitteessä 1 on kuvattu ohje esimerkkisovelluksen kokoamiseksi. Kokonaisuudessaan esimerkkisovellus on dokumentoitu myös harjoitustyönä (Hirvonen, 2008).

6.1 Portaaliympäristö (Apache Pluto) ja portlettirajapinta

Apache Pluto (2008) on JSR 168 -rajapinnan referenssitoteutus (jatkossa myös JSR 286:en). Se toteuttaa Portlet API:n mukaisen portlettisäiliön, jonne voidaan asentaa rajapinnan toteuttavia portletteja. Tämän lisäksi asennettuja portletteja voidaan hyödyntää ja lopulta ne voi-

daan poistaa. Portlettisäiliö ei ole yksistään toimiva kokonaisuus, vaan se laajentaa servlettisäiliön toiminnallisuutta ja uudelleenkäyttää sen toimintalogiikkaa. Lisäksi Pluto toteuttaa lähinnä kehitystä ja testaamista varten varsinaisen portaalin, jonka avulla portleteista voidaan muodostaa portlettisivuja.

Portlettisovelluksen asentamiseksi Plutoon tarvitaan kaksi vaihetta (Apache Pluto, 2008).

1. Koostaminen (Assembly): Kaikki portlettisovellukset on koostettava Pluton koostajaprosessin avulla. Koostajaprosessi lisää jakeluun Pluton tarvitsemia ohjaustietoja jakelua varten. Tämän lisäksi se lisää J2EE-standardin mukaiseen sovelluksen konfigurointitiedostoon (web.xml) servletin ja sen relaation kuvauksen (servlet mapping). Tätä servlettiä käytetään portletteihin kohdistuvien pyyntöjen ohjaamiseen portaalisovellukselle.
2. Jakelu (Deployment): Koostamisen jälkeen portlettisovellukset asennetaan samaan servlettisäiliöön, jossa portaalisovellus on käynnissä.

Portlettirajapinta (JSR 168, 2003) sisältää yleisen portlettiluokan (GenericPortlet), joka toteuttaa portletti-rajapinnan ja tarjoaa perustoiminnallisuuden, jota kehittäjät voivat laajentaa (ks. liite 4: ContentPortlet). Portletin elinkaari koostuu portletin alustamisesta (init), asiakkailta tulevien pyyntöjen käsittelystä (processAction), näkymän muodostamisesta (render), ja lopulta se poistetaan käytöstä (destroy). Portaalisäiliö vastaa portlettisovellusten lataamisesta ja käynnistämisestä, jonka jälkeen portaalisäiliö alustaa portletin, jotta se voi vastata käyttäjiltä tuleviin pyyntöihin. Alustuksen yhteydessä portletti voi ottaa käyttöön tarvitsemansa resurssit, ja suorittaa kertaluontoisia toiminnallisuuksia, jotka voivat olla resurssien käytön kannalta kalliita. Alustaminen tapahtuu kutsumalla portletin toteuttamaa init-metodia. Init-metodille välitetään konfiguointiobjekti, joka toteuttaa PortletConfig-rajapinnan ja tarjoaa pääsyn käynnistysparametreihin ja resursseihin. Mikäli alustuksen yhteydessä tapahtuu virheitä, ei portlettisäiliö saa päästää portlettia aktiiviseen tilaan, vaan sen tulee vapauttaa portlettiolio.

Portletin määritys voi sisältää joukon asetustietoja, ja niiden oletusarvoja. Ajon aikana portletti voi lukea, lisätä ja muokata noita asetustietoja. Asetustietoja portletti käyttää yleensä oman toiminnallisuutensa ja sisällön mukauttamiseen. Oletuksena asetustiedot luetaan portletin jakelukuvauksesta (deployment descriptor), mutta portaalisäiliö voi tarjota muitakin tapoja

niiden hallinnointiin. Kun portletti asetetaan portaalisivulle, siihen liitetään samalla asetustieto-olio. Tätä portlettiolion ja preferenssiolion ilmentymää kutsutaan portletti-ikkunaksi, ja niiden välisestä yhteydestä huolehtii portaalisäiliö. Kuvassa 24 alustetaan portletti ja haetaan siihen liittyviä asetustietoja.

```
public void init(javax.portlet.PortletConfig config)
      throws UnavailableException {
  Channel chan = new Channel();
  try {
      this.setServiceName(config.getInitParameter("service-name"));
      this.setServiceURL(config.getInitParameter("service-url"));

      chan.setDescription(config.getInitParameter("channel-desc"));
      chan.setTitle(config.getInitParameter("channel-title"));
      chan.setLink(config.getInitParameter("channel-link"));
      } catch (Exception e) {
            throw new UnavailableException(
                  "Required config parameters not found", -1);
      }
      this.setChannel(chan);
}
```

Kuva 24 - Sisältöportletin alustaminen ja asetustietojen haku

Kun portletti on saatu käyntiin, voi portlettisäiliö ohjata sille pyyntöjä käsittelyä varten. Käsittely jakaantuu kahteen osaan, renderöintiin, ja toimintojen prosessointiin. Käsittelyä varten on toteutettava metodit *render* ja *processAction*, ja niitä vastaavia kutsuja kutsutaan renderöintipyynnöksi ja toimintopyynnöksi. Yleensä renderöinti- ja toimintopyynnöt käynnistyvät niitä vastaavien portlettiosoitteiden (portletURL) perusteella, asiakkaan toimesta. Portlettiosoite kohdistuu aina vain yhteen portlettiin, ja niitä voi olla kahta tyyppiä, toiminto-osoite ja piirto-osoite. Normaalisti toiminto-osoitetta vastaava portlettikutsu jakaantuu yhdeksi toimintopyynnöksi ja useaksi renderöintipyynnöksi. Renderöintipyyntöjä tehdään usein portaalisivun portlettien määrän verran. Toimintopyyntö suoritetaan ensin kuitenkin loppuun saakka. Renderöintipyyntöjen keskinäistä järjestystä ei ole kiinnitetty, vaan ne tapahtuvat satunnaisessa järjestyksessä samanaikaisesti. Jos pyyntö kohdistuu taas suoraan renderöintiosoitteeseen, piirretään yleensä koko näkymä uusiksi (toisin sanoen kutsutaan jokaisen portletin render-metodia), pois lukien ne portletit, joiden sisältö löytyy mahdollisesta välimuistista. Tällöin ei suoriteta mitään processAction-metodia. Voi myös olla, ettei portletille kohdistu yhtään pyyntöä koko sen elinkaaren aikana.

Toimintopyynnön aikana portletti yleensä päivittää tilaansa pyynnössä saatujen parametrien avulla. Se saa parametrinaan *ActionRequest*- ja *ActionResponse*-oliot. ActionRequest-oliolta voidaan lukea pyynnön tarkemmat parametrit, portletti-ikkunan tila, portletin toimintamuoto (tila), konteksti, istunto ja sen asetustiedot. Toimintopyynnön aikana käyttäjä voidaan myös uudelleenohjata johonkin osoitteeseen, jolloin portlettisäiliön on lähetettävä uudelleenohjaus takaisin loppukäyttäjän päätelaitteelle ja päätettävä pyynnön suoritus. Toimintopyyntö voi aiheuttaa myös portletti-ikkunan tilan tai portletin toimintamuodon vaihdoksen. Tämä tehdään ActionResponse-olion kautta. Toimintamoodin vaihdon tulee yleensä vaikuttaa renderöinti-pyyntöön, mutta näin ei käy välttämättä kaikissa tilanteissa (esimerkiksi oikeuksien puuttumisen takia). Myös ikkunan tilan muuttuminen vaikuttaa yleensä kyseisen portletin renderöinti-pyyntöihin, mutta mitään oletuksia tästä ei voida tehdä, koska portaalisäiliössä voidaan riippuvuuksien takia yliajaa portletti-ikkunan tila.

```
public void doHelp(RenderRequest req, RenderResponse res)
                  throws PortletException, IOException {

      res.setContentType("text/html");
      PortletContext context = getPortletContext();
      PortletRequestDispatcher dispatcher = context
                        .getRequestDispatcher("/WEB-INF/jsp/help.jsp");
      dispatcher.include(req, res);
}
```

Kuva 25 – Renderöinnin delegointi JSP-sivulle

Renderöintipyyntöä käytetään yleensä sisällön esittämiseen pyynnössä saatujen parametrien pohjalta. Sekin saa kaksi parametria, *RenderRequest*- ja *RenderResponse*-objektit. RenderRequest-objektista (kuten ActionRequest -objektista) voidaan lukea pyynnön parametrit, ikkunan tila, konteksti, istunto ja asetustiedot. Sisältö tuotetaan RenderResponse-kirjoittajan avulla, tai sen tuottaminen voidaan delegoida servletille tai JSP-sivulle (Java Server Page). Kuvassa 25 on ContentPortlet-luokan doHelp-metodi, jossa delegoidaan sisällön tuottaminen help.jsp-sivulle PortletRequestDispatcher-olion avulla.

Portletin toimintamuoto kuvaa portletin tilaa ja sen perusteella portletin suorittamaa toiminnallisuutta. Toimintamuotoa voidaan muuttaa ohjelmallisesti. Yleensä portletilla on erilaisia tehtäviä, ja se tuottaa kunkin tehtävän mukaista sisältöä. Portlettimääritys (JSR 168, 2003) määrittää portletin toimintamuodoiksi katselu (View), muokkaus (Edit) ja ohje (Help). Näistä

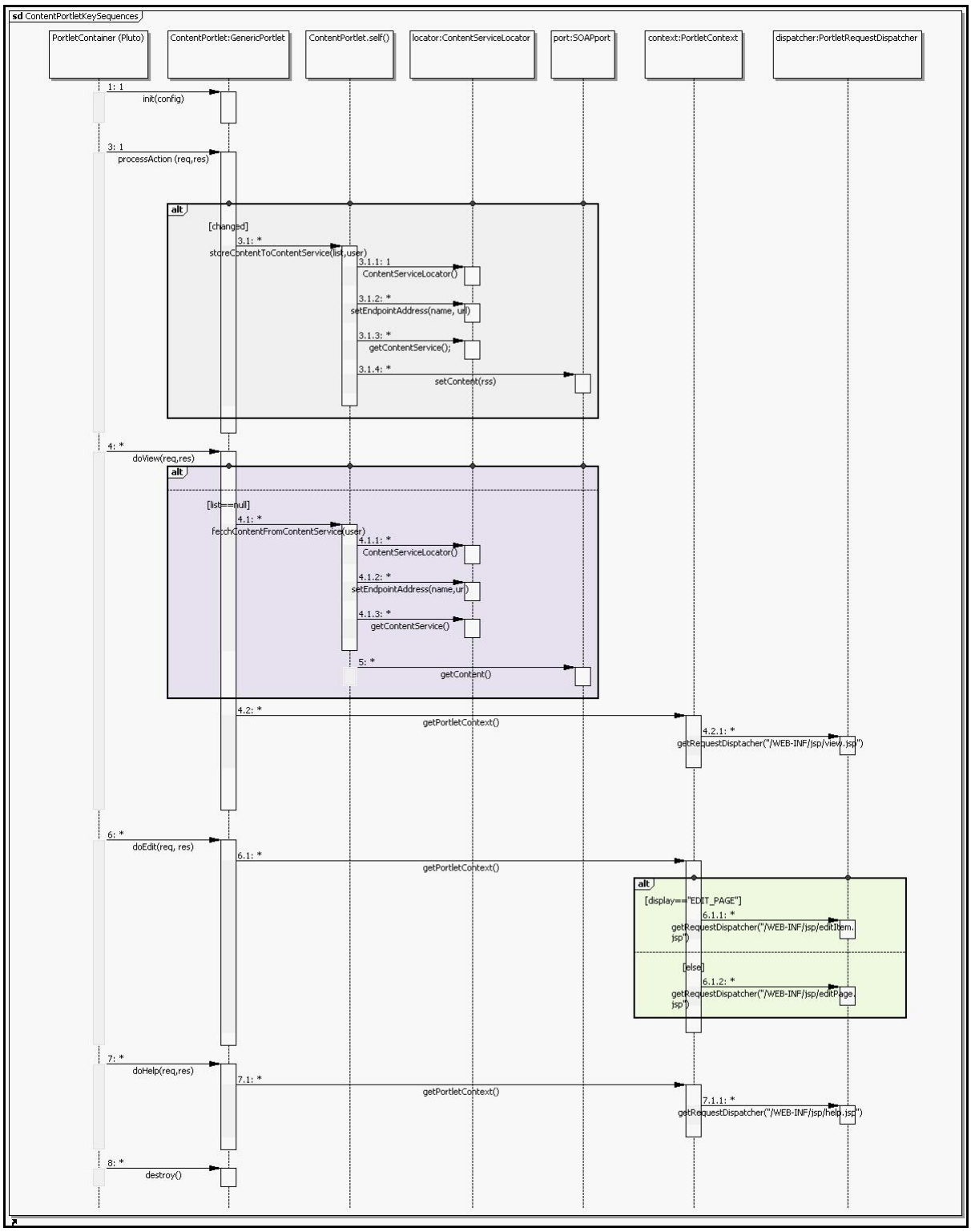
portletin on tuettava vähintään katselutilaa. Katselutila on tarkoitettu normaaliin portletin sisällön esittämiseen yhden tai useamman sivun avulla, jolloin loppukäyttäjä voi esimerkiksi navigoida näiden sivujen välillä tai hyödyntää niissä olevaa toiminnallisuutta, tai pelkästään katsella niiden tietosisältöä. Muokkaus-toimintamuodossa olevan portletin odotetaan tarjoavan toiminnallisuutta, jonka avulla loppukäyttäjä voi muuttaa portletin käyttäytymistä, yhden tai useamman näytön avulla. Tyypillisesti muokkaustilassa voidaan asettaa tai muokata portletin asetustietoja. Ohje-toimintamuodossa portletin tulisi tarjota portlettiin liittyvää avustavaa tietoa. Se voi muodostua yhdestä yksinkertaisesta ohjesivusta tai se voi tarjota asiayhteydestä riippuvan ohjeistuksen. Näiden ennalta määrättyjen toimintamuotojen lisäksi portaalitoimittajat voivat tuottaa lisäpiirteinä uusia toimintomuotoja. Portletti voi hyödyntää vain niitä toimintamuotoja, mitä sitä suorittava portaalisäiliö sille tarjoaa. Portletti voi itse päättää sen tukemista toimintamuodoista, tämä tapahtuu portletin jakelukuvauksen avulla.

JSR 168 -määrityksen mukaisia portletti-ikkunan mahdollisia tiloja ovat normaali (normal), minimoitu (minimized) ja maksimoitu (maximized). Portletti-ikkunan tila vaikuttaa yleensä siihen, miten paljon portletti tulostaa informaatiota. Normaalitilassa portletti-ikkuna yleensä jakaa näytön muiden portlettien kanssa. Maksimoituna portletti voi olla ainoa portletti näytöllä, jolloin se voi tuottaa näytölle lisäsisältöä tarvitessaan. Minimoituna portletin tulisi välttää kaikenlaista tulostamista, mikäli mahdollista niin kokonaan. Kuten toimintamuotoja, myös portletti-ikkunan tiloja voidaan muuttaa ohjelmallisesti. Lisäksi portaalitoimittajat voivat toteuttaa portaaliinsa omia portletti-ikkunoiden tiloja, joiden hyödyntämisestä portletti ilmoittaa jakelukuvauksessaan.

Kun portlettisäiliö päättää, että portletin elinkaari on saapunut päätökseen, se kutsuu portletin destroy-metodia. Tämän metodikutsun suorituksen aikana tulee portletin tallentaa kaikki sen pysyvät tilat ja tämän jälkeen vapauttaa kaikki varaamansa resurssit. Portlettisäiliö voi kutsua destroy-metodia vaikkapa vapauttaakseen resursseja tai silloin kun sitä ollaan sulkemassa. Portlettisäiliön tulisi sallia kaikkien käynnissä olevien pyyntöjen suorittaminen loppuun, ennen kuin destroy-metodia kutsutaan. Kun destroy-metodi on suoritettu, vapauttaa portlettisäiliö portlettiobjektin automaattista roskien keruuta varten.

JSR 168 -määrityksen mukainen abstrakti luokka GenericPortlet tarjoaa perustoiminnallisuuden renderöintipyyntöjen käsittelyyn. Sen render-metodi asettaa portletin otsikoksi portletin jakelukuvauksesta saatavan parametrin ja kutsuu doDispatch-metodia, joka sisältää toimin-

nallisuutta pyynnön käsittelyyn portletin toimintamuodon perusteella. Tämä metodi toimittaa pyynnön portletin tukemien toimintamuotojen perusteella doView-, doEdit- tai doHelp- metodille. Mikäli portletti-ikkunan tila on minimoitu, ei GenericPortlet kutsu mitään sen renderöintimetodeista.



Kuva 26 – ContentPortlet-luokan pelkistetty sekvenssikaavio

Toteuttamani ContentPortlet-luokka (esitetty liitteessä 4) laajentaa GenericPortlet-luokan toiminnallisuutta. Portletti toimii kuvan 23 mukaisen ContentService-palvelun asiakkaana ja loppukäyttäjän käyttöliittymänä, ja sen keskeisimmät interaktiot on esitetty kuvassa 26. Käyttöliittymän avulla loppukäyttäjä voi ylläpitää omaa linkkilistaansa, joka tallentuu ContentService-palvelun takana olevaan sisältösäiliöön. Se toteuttaa portletin elinkaaren mukaiset metodit. Alustamisen yhteydessä luetaan portletin asetustietoja (kuva 24). Toimintapyyntöjen käsittelystä vastaa processAction-metodi, joka alustaa listan käyttäjän istuntoon, johon sisältö talletetaan. Olemassa oleva sisältö käydään hakemassa sisältövarastosta, fetchContentFromContentService-metodin avulla, joka hyödyntää generoituja Axis-luokkia. Käyttäjän valitsemasta toiminnallisuudesta riippuen listaan joko lisätään RSS-viestimuodon mukaisia alkioita, tai niitä muokataan tai poistetaan. Jos sisältö muuttui, niin lopuksi uusi sisältö tallennetaan palvelurajapinnan yli pysyvään tietovarastoon (storeContentToContentService-metodi). Renderöintimetodit doView, doEdit ja doHelp delegoivat tulostuksen vastaavalle JSP-sivulle.

6.2 Sisältövarasto (Apache Jackrabbit) ja sen konfigurointi

Seuraavaksi käymme läpi sisältövaraston, Apache Jackrabbit, arkkitehtuuria ja keskeisimpiä toiminnallisuuksia. Lisäksi käymme läpi sisältövaraston keskeisimmät toimitusmallit ja konfuguroinnin osaksi esimerkkisovellusta.

Apache Jackrabbit (2008) on avoimen lähdekoodin toteutus JSR 170 -määrityksen mukaisesta sisältövarastosta. Se koostuu kolmesta kerroksesta, sisältösovelluskerroksesta, sisältövarastorajapinnasta ja sisältövaraston toteutuskerroksesta. *Sisältösovellukset* voivat olla mitä tahansa JSR 170 -rajapintaa tukevia sovelluksia, joista osa voi olla hyvinkin yleiskäyttöisiä, kuten esimerkiksi WebDAV-palvelin, tai ne voivat olla myös erikoistuneempia sovelluksia, jotka hyödyntävät sisältövarastoa tietyntyyppisen informaation tallennuspaikkana. Sisältövarasto helpottaa suuren hierarkkisen tietovaraston luomista, ja sen avulla sovellus voi hyödyntää sisältövaraston tarjoamia palveluita, kuten versiointia, transaktioiden hallintaa, kyselyitä ja nimiavaruuksia. Yleiset sisältösovellukset eivät ota varsinaisesti kantaa siihen mitä tietosisältöä niillä tallennetaan, kun taas erikoistuneet sisältösovellukset yleensä kuvaavat tietosisältöä tai tietomallia, joka määrää usein myös sovelluksen käyttötarkoituksen. Esimerkkinä erikoistuneista sisältösovelluksista on esimerkiksi DVD-levyjen hallintasovellus, tai työnkulku.

Sisältövarastorajapintakerros toteuttaa varsinaisen JSR 170 -rajapinnan lisäksi myös siitä pois jätettyjä piirteitä. Näiden piirteiden toteuttaminen jo olemassa olevaan, muuhun kuin Java-pohjaiseen, sisältövarastoon on suhteellisen hankalaa, josta syystä ominaisuudet on jätetty pois rajapintamäärityksestä. Lisäksi sisältövarastorajapintakerrokseen on lisätty JSR 170 -rajapinnasta puuttuvat ylläpidolliset ja hallinnalliset toiminnallisuudet, kuten käyttäjien hallintarajapinta, solmutyyppien hallintarajapinta ja synkroninen tarkkailurajapinta.

Sisältövaraston toteutuskerros on Jackrabbitin kerroksista suurin ja se sisältää eniten toiminnallisuutta. Sisältövaraston toteutuskerros on jaettu kolmeen vaikutusalueeseen, joita ovat: varastonäkökulma, työtilanäkökulma ja istuntonäkökulma. Jokainen sisältövarastoon kohdistuva toiminnallisuus voidaan kohdistaa vähintään yhteen vaikutusalueeseen, ja jotkut funktiot voivat kohdistua useampaan vaikutusalueeseen.

Koska JSR 170 sallii useita erilaisia toimitusmalleja (deployment models) sisältövarastoille, sisältövaraston toteuttajan vastuulle jää ko. toteutukseen soveltuvien toimitusmallien valinta. Apache Jackrabbit (2008) tukee kolmea varsinaista toimitusmallia:

- *(Verkko-)sovellukseen sisällytetty toimitusmalli* on malli, jossa Apache Jackrabbit toimitetaan jonkun muun sovelluksen mukana. Tällöin sisältövarasto on ainoastaan kyseisen sovelluksen hyödynnettävissä. Apache Jackrabbit on toteutettu tätä toimitusmallia silmälläpitäen, ja se mahdollistaa hyvin kevyen toimituksen, eikä se vaadi verkkokerrosta. Yksittäiset sisältövarastot käynnistetään ja sammutetaan ne sisältävien sovellusten toimesta.
- *Jaettu J2EE-resurssi* tarkoittaa toimitusmallia, jossa Apache Jackrabbit konfiguroidaan sovelluspalvelimen resurssiksi, josta se on muiden sovellusten hyödynnettävissä. Tällöin useampi (samalle sovelluspalvelimelle asennettu) sovellus voi hyödyntää yhteistä sisältövarastoa. Myöskään tässä mallissa ei tarvita verkkokerrosta. Sisältövaraston käynnistämisestä ja sammuttamisesta vastaa tällöin sovelluspalvelin.
- *Erillinen sisältövarastopalvelin* on malli, jossa Apache Jackrabbit asennetaan kokonaan omaksi kokonaisuudekseen. Tämä sallii sisältövaraston skaalaamisen erikseen, ja tällöin sen hyödyntäminen on mahdollista myös täysin erillisistä sovelluksista. Tässä mallissa sisältövarastoa hyödynnetään pitkälti kuten relaatiotietokantaa, ja se edellyttää verkkokerrosta sisältövaraston ja sitä hyödyntävän asiakassovelluksen välillä.

Asiakassovellus hyödyntää varastoa JSR 170 -rajapinnan avulla haluamallaan tiedonsiirtoprotokollalla (esimerkiksi RMI).

JSR 170 -rajapinnan mukanaan tuoman abstraktion avulla on toimitusmallia helppo vaihtaa myöhemmin, mikäli sille on tarvetta. Esimerkkisovelluksessa käytämme sovellukseen sisällytettyä mallia, joka toteutetaan Apache Tomcat-sovelluspalvelimen version 5.5 kanssa seuraavasti:

1. Liitetään tarvittavat kirjastot ja riippuvuudet osaksi sovellusta.
2. Luodaan resurssi sovelluksen kontekstiin.
3. Hyödynnetään resurssia sovelluksessa.

```
<Context>
<Resource
  name="jcr/repository"
  auth="Container"
  type="javax.jcr.Repository"
  factory="org.apache.jackrabbit.core.jndi.BindableRepositoryFactory"
  configFilePath="/repository/repository.xml"
  repHomeDir="/repository/repo"/>
</Context>
```

Kuva 27 - Resurssin luominen sovelluksen kontekstiin (context.xml)

Esimerkkisovelluksen tapauksessa tarvittavista riippuvuuksista huolehtii Apache Maven (2008), joka on avoimen lähdekoodin projektinhallintaohjelmisto. Mavenin vaatima ohjaustiedosto pom.xml ContentService-komponenttia varten löytyy liitteestä 2. Resurssi luodaan esimerkkisovelluksen (tarkemmin ContentService-komponentin) kontekstiin, Tomcat-sovelluspalvelimelle, liittämällä tiedosto context.xml osaksi web-sovellusta. Kuvassa 27 on esitetty esimerkkisovelluksen context.xml -tiedosto. Tämän jälkeen sisältövarastoa voidaan hyödyntää helposti hakemalla sitä vastaava resurssi kontekstista. Tätä havainnollistaa esimerkkisovelluksen ContentBindingImpl-luokan metodi getEmbeddedRepository(), joka on esitetty kuvassa 28. Kun sisältövarasto on haettu kontekstista, luodaan siihen istunto kirjautumalla käyttäen tunnusta ja salasanaa. Tämän jälkeen on istunnon juurisolmu haettavissa.

Esimerkkisovelluksen RSS-pohjaisen viestin juurisolmussa on käyttäjä, jonka lapsisolmuna on kanava (channel). Kanavan ominaisuuksina on otsikko, linkki, kuvaus sekä alkioiden

määrä. Kanavan lapsisolmuina on 1-n kappaletta alkioita, joilla ominaisuuksina on otsikko, linkki ja kuvaus.

Muutoin ContentService-komponentti edustaa tyypillistä web-palvelun palvelinkomponenttia, sen runkototeutus generoidaan liitteen 2 mukaisesta wsdl-tiedostosta Apache Maven-ohelmistoa hyödyntäen, jonka jälkeen generoitu toteutusluokka korvataan liitteessä 2 olevalla toteutuksella (ContentServiceBindingImp). Se toteuttaa getContent- ja setContent-metodit, joiden tarkoituksena on hakea tai tallentaa käyttäjän henkilökohtaiset sisällöt.

```
private Repository getEmbeddedRepositroy()
      throws NamingException,RepositoryException{
  InitialContext context = new InitialContext();
  Context environment = (Context) context.lookup("java:comp/env");
  Repository repository = (Repository)
      environment.lookup("jcr/repository");
  return repository;
}
```

Kuva 28 - Sisältövaraston hakeminen sovelluksen kontekstista

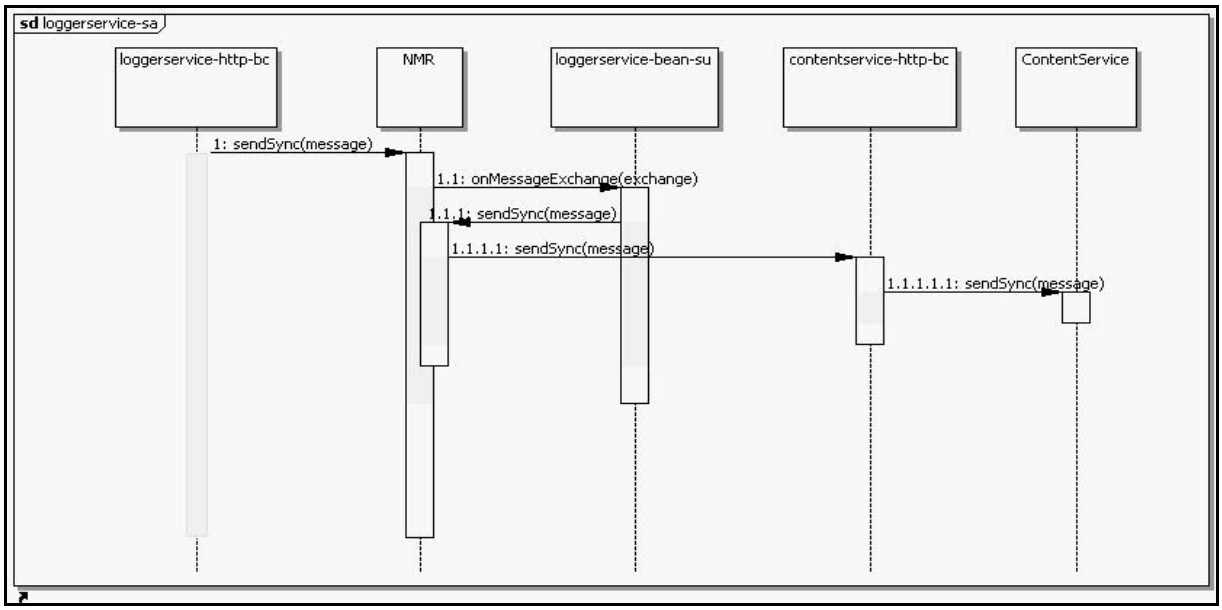
6.3 Palveluväylän (Apache ServiceMix) käyttö reititykseen

Apache ServiceMix (2008) toteuttaa JSR 208 mukaisen rajapinnan normalisoidulle viestipalvelulle ja reitittimelle, sekä JBI:n määrittämät JMX-hallintalaajennukset (MBeans). JBI-komponenttien asennus ja hallinta tapahtuvat ServiceMixissä Ant- tai (rajoitetusti) Maven-työkalua hyödyntäen. ServiceMix sisältää joukon valmiita JBI-komponentteja, joiden avulla se tukee esimerkiksi WS-BPEL-standardia (2007).

ESB:n käytön havainnollistamiseksi esimerkkisovellukseen on toteutettu JBI-komponentti lokitietojen tallentamista varten (liite 3). Lokitukseen tarkoitettu JBI-komponentti paketoidaan yhdeksi kokonaisuudeksi, *palvelumoduuliksi* (service assembly). Palvelumoduuli (loggerservice-sa) koostuu kahdesta sidoskomponentista (loggerservice-http-bc ja contentservice-http-bc) ja yhdestä palvelumoottorista (loggerservice-bean-su). Sidoskomponentissa loggerservice-http-bc konfiguroidaan yksi palvelun päätepiste, joka ottaa vastaan palvelupyyntöjä toimiessaan palvelun kuluttajan roolissa ja ohjaa ne palvelumoottorille. Palvelumoottori on Java-komponentti, joka kirjoittaa pyynnön lokiin ja käynnistää normalisoidun viestinvaihdon sisältöpalvelun kanssa, normalisoidun

viestireitittimen (NMR) kautta. Sisältöpalvelun kanssa kommunikoidaan toisen sidoskomponentin, contentservice-http-bc kanssa, joka toimii palvelun tuottajan roolissa.

Kuvassa 29 on esitetty loggerservice-sa -komponentin sisäiset riippuvuudet ja suhde ContentService-komponenttiin sekvenssikaaviona esitettynä. Kuvasta on havaittavissa, että liikennöinti komponenttien välillä tapahtuu normalisoidun reitittimen kautta.



Kuva 29 - loggerservice-sa -komponentin pelkistetty sekvenssikaavio

Molemmat sidoskomponentit hyödyntävät servicemix-http-komponenttia, joka on valmis toteutus http-sidosta varten. Komponentti tukee SOAP-protokollan versiota 1.1 ja 1.2. Lisäksi se tukee kaikkia JBI:n tukemia viestinvaihtomenetelmiä joko palvelun kuluttajan tai tuottajan roolissa. JBI tukemat viestinvaihtomenetelmät ovat (JSR 208, 2005):

- Vain sisään (In-Only): Kuluttaja lähettää viestin tuottajalle ilman mahdollisuutta virheenkäsittelyyn.
- Mukautuvasti sisään (Robust-In-Only): Kuluttaja lähettää viestin tuottajalle. Tuottaja voi halutessaan vastata virhesanomalla (fault).
- Sisään ja ulos (In-Out): Kuluttaja lähettää sanoman, johon se voi odottaa tuottajan vastaavan. Vastaus voi olla myös virhesanoma.

- Sisään ja mahdollisesti ulos (In-Optional-Out). Kuluttaja lähettää sanoman, johon se voi saada vastauksen. Sekä kuluttaja että tuottaja voivat muodostaa virhesanoman viestinvaihdon aikana.

Sidoskomponenteista molemmat noudattavat sisään ja ulos -tyyppistä viestinvaihtomenetelmää. Esimerkki sidoskomponentin konfiguroinnista on esitetty kuvassa 30.

```
<beans xmlns:http="http://servicemix.apache.org/http/1.0"
      xmlns:ls="http://cs.joensuu.fi/jhirvone/loggerservice">
      <http:endpoint service="ls:inputReceiver" endpoint="soap"
            role="consumer"
            targetService="ls:logger"
            targetEndpoint="log"
            locationURI="http://0.0.0.0:8192/loggerservice/"
            defaultMep="http://www.w3.org/2004/08/wsdl/in-out"
            soap="true"
            soapVersion="1.1" />
</beans>
```

Kuva 30 - loggerservice-http-bc sidoskomponentin konfiguraatio

Palvelumoottori loggerservice-bean-su sisältää Java-luokan LoggerService, joka toteuttaa MessageExchangeListener-rajapinnan (esitetty liitteessä 3). Loggerservice-bean-su on itsessään servicemix-bean-komponentti, jonka tarkoituksena on helpottaa integrointia JBI-väylän kanssa Java-beanien (POJO) avulla (Apache Servicemix, 2008). Tällöin POJO:n avulla voidaan kontrolloida JBI-viestinvaihtoa ja se voidaan prosessoida toteuttajan haluamalla tavalla, joka kuitenkin edellyttää ohjelmakoodausta. Kun POJO toteuttaa MessageExchangeListener-rajapinnan, jokainen viestinvaihto ohjataan luokan onMessageExchange- metodille. Tällöin jokainen suoritettava viestinvaihto on aktiivisessa tilassa (`ExchangeStatus.ACTIVE`) niin kauan kunnes se terminoidaan asettamalla tila hyväksytyksi (`ExchangeStatus.DONE`) tai virhetilaksi (`ExchangeStatus.ERROR`). Itse onMessageExchange-metodissa muunnetaan sisään tulevan viestin sisältö merkkijonoksi, jonka jälkeen hyödynnetään ServiceMixin Client API:a tekemällä uusi asiakas, joka asetetaan kutsumaan contentservice-http-bc-komponenttia NMR:n kautta (kuva 31). Tämän jälkeen saatu vastaus kirjoitetaan lokiin ja palautetaan se suoraan sellaisenaan kutsuvalle komponentille.

```
ServiceMixClient client = new ServiceMixClientFacade(context);
QName service = new QName(SERVICE_NAMESPACE, DESTINATION_SERVICE,
            SERVICE_NAMESPACE_PREFIX);
InOut outexchange = client.createInOutExchange();
outexchange.setService(service);
NormalizedMessage message = outexchange.getInMessage();
message.setContent(new StreamSource(new StringReader(messageString)));
client.sendSync(outexchange);
```

Kuva 31 – Palvelun kutsuminen JBI-komponentista

7 YHTEENVETO

Tässä tutkielmassa perehdyttiin portaalien ja sisällönhallintajärjestelmien integrointiin palvelukeskeisen arkkitehtuurin mukaisesti Java-ympäristössä. Toteuttajan näkökulmasta keskeisimpiä asioita ovat kehittämiseen liittyvät standardit ja niihin liittyvien teknologioiden tiedostaminen.

Portaalit koostuvat portaalisäiliöstä, portlettisivuista ja portleteista. Portaalin tarkoituksena on koota eri tietolähteistä koostuva informaatio käyttäjälle yhtenäiseen näkymään. Niiden merkitys, toiminnallisuus ja tekniikka on määritetty JSR 168 -rajapinnassa, joka kuvaa standardinmukaisen tavan portaalisovellusten toteuttamiseen.

Sisällönhallintajärjestelmien yhtenä tarkoituksena on toimia keskitettyinä sisältövarastoina, joista tietosisältö pyritään hyödyntämään mahdollisimman tehokkaasti integroimalla sisällönhallintajärjestelmät muihin sovelluksiin. Sisällönhallinnan prosessiin liittyy keskeisesti valtuuttaminen, varastointi ja julkaisu. JSR 170 määrittää standardin rajapinnan sisältövaraston ohjelmalliseen kutsumiseen, sekä hierarkkisen tavan sisällön ja siihen liittyvän metatiedon varastoimiseen.

Yritysjärjestelmien integrointimenetelmistä tutustuttiin tarkemmin integrointiin palvelukeskeisen arkkitehtuurin mukaisesti. Palvelukeskeisessä arkkitehtuurissa sovellukset rakennetaan komponenteista, joiden sisältämä tietosisältö ja tarjoamat rajapinnat kuvataan, yleensä WSDL-kuvauskielellä. Komponentit integroidaan keskenään löyhän sidonnan periaatteen mukaisesti, jolloin palvelun kuluttajan ei tarvitse tietää juuri mitään palvelun tarjoajan tekemästä palvelun toteutuksesta. Palveluiden integroinnissa käytetään viestipohjaista liikennöintiä, ja viestit toteutetaan usein SOAP-protokollan mukaisesti. Palvelukuvaukset varastoidaan palveluhakemistoon, jotta niiden hallinnointi, löydettävyys ja sitä kautta uudelleenkäytettävyys säilyy palveluiden määrän kasvaessa. Web-palveluiden tietoturvanäkökulmaa tarkastellessa on syytä tutustua viestiliikenteen salaamiseen WS-Security suosituksen mukaisesti.

Palveluväylä on olennainen osa palvelukeskeistä arkkitehtuuria. Palveluväylän tehtävänä on yhdistää palvelun tuottajat ja kuluttajat viestien siirtotiestä, viestinvälitystavasta tai protokollasta riippumatta. Se helpottaa palveluiden hallintaa, ja mahdollistaa palveluiden yhdistämisen

organisaation prosessien mukaisiksi palveluketjuiksi ja uusiksi palveluiksi konfiguroimalla. JBI-standardi kuvaa Java-pohjaisen mallin palveluväylän ja sen komponenttien toteuttamiseen sekä paketointiin.

Palvelukeskeisen arkkitehtuurin etuina pidetään, pitkällä aikavälillä tarkasteltuna, sen tuomaa organisaation muutosherkkyyttä, sen vähentämää kompleksisuutta, uudelleenkäytettävyyttä ja sen mukanaan tuomaa ohjelmistojen parempaa yhteensopivuutta. Omien kokemuksieni mukaan ainakin ohjelmistojen yhteensopivuus on selkeästi kehittynyt viime vuosina. Lisäksi etenkin integraatioiden uudelleenkäytettävyys ja tuotteistaminen on aiempaa helpompaa. Kompleksisuuden vähenemiseen kannattaa suhtautua hieman varauksella, sillä vaikka palvelut itsessään löyhän sidonnan mukaisesti vähentävät järjestelmien kompleksisuutta, tuovat uudet standardit, ohjelmistokerrokset, välineet ja menetelmät haasteita sovelluksen kehittäjälle vaadittavan osaamisen kannalta katsottuna.

Nähtäväksi (ja myöhemmin tutkittavaksi) jää, vaikuttaako palvelukeskeinen arkkitehtuuri esimerkiksi kehittyvien prosessien mallinnusominaisuuksien kautta merkittävästi organisaation muutosherkkyyteen tulevaisuudessa. Muita mielestäni kiinnostavia jatkotutkimuksen aiheita ovat esimerkiksi tästä tutkimuksesta ulos jääneet sisältöstandardit, tai vaihtoehtoiset menetelmät palvelukomponenttien tuottamiseen (esim. SCA-standardi ja muut kuin JAX-RPC -pohjaiset sovelluskehykset).

Portaalien määrä tullee jatkossakin kasvamaan, joten niissä tarvittavan sisällön määrä kasvaa entisestään. Tämän takia portaaleja ja sisällönhallintajärjestelmiä tullaan integroimaan enenevissä määrin myös tulevaisuudessa. Palvelukeskeinen arkkitehtuuri on epäilemättä yksi merkittävimmistä uudistuksista yritysjärjestelmien sovelluskehityksessä ja integroinnissa sitten internetin ja olio-ohjelmoinnin. Palvelukeskeisen arkkitehtuurin potentiaalin hyödyntämisessä ollaan kuitenkin vielä vasta alkutaipaleella, mutta sen keskeisten periaatteiden tiedostaminen ja ymmärtäminen on askel oikeaan suuntaan.

8 VIITELUETTELO

Addey, D., Ellis, J., Suh, P., Thiemecke, D. (2002) *Content Management Systems*. Glasshaus Ltd, Birmingham.

Aiken, P., Finkelstein C. (1999) *Building Corporate Portals with XML*. McGraw-Hill, New York.

Armstrong, E., Bodoff, S., Carson, D., Evans, I., Fisher, M., Green, D., Haase, K., Jendrok, E., Pawlan, M., Stearns, B. (2003) *The J2EE 1.4 Tutorial*. Sun Microsystems, Inc., Santa Clara. URL: http://www.it.uom.gr/host/EPEAEKII/KSpms/shmeiwseis/Yliko/J2EETutorial.pdf (22.2.2008).

Apache Axis (2008) *The Apache XML Project.*, WWW-sivusto, Axis Development Team, URL: http://ws.apache.org/axis/index.html (23.2.2008).

Apache Jackrabbit (2008) *Apache Jackrabbit – The Open Source Content Repository for Java*, WWW-sivusto, Apache Software Foundation, URL: http://jackrabbit.apache.org/index.html (23.2.2008).

Apache Maven (2008) *Apache Maven.* WWW-sivusto, Apache Software Foundation, URL: http://maven.apache.org (23.2.2008).

Apache Pluto (2008) *The Apache Pluto Project.* WWW-sivusto, Apache Software Foundation, URL http://portals.apache.org/pluto/ (23.2.2008).

Apache ServiceMix (2008) *Apache ServiceMix*, WWW-sivusto, Apache Software Foundation, URL: http://servicemix.apache.org/home.html (23.2.2008).

Banke, K., Krafzig, D., Slama D. (2005) *Enterprise SOA*. Pearson Education, Inc., Indiana.

Barik T. (2005) *Introducing the Java Content Repository API*. IBM CoURL: http://www.ibm.com/developerworks/java/library/j-jcr/ (23.2.2008).

Boiko, B. (2002) *Content Management Bible*. Hungry Minds, Inc., New York.

Browning, P, Lowndes, M. (2002) *JISC TechWatch Report: Content Management Systems*, Bristol University, Bristol.

Cerami, E. (2002) *Web Services Essentials*. O'Reilly & Associates, Inc. (Saatavana myös: http://www.oreilly.com/catalog/webservess/chapter/ch06.html, 23.2.2008).

Chappell, D. (2004) *Enterprise Service Bus*. O'Reilly Media, Inc., USA.

Clark, D. (2008) Content Management and the Separation of Presentation and Content. *Technical Communication Quarterly*, 17(1), 35-60.

Dias, C. (2001) Corporate Portals: a literature review of a new concept in Information Management. *International Journal of Information Management*, 21 (toim. Hills P.), Elsevier Science Ltd., Oxford, 2001, 269-287.

Dovey, M. (2001) *JISC Technology Watch Report: Java Portals.* Oxford University, Oxford.

Doyle, L. (2002) *Content Management Systems Workshop Report*, Fourth Institutional Web Management Workshop, Bathin yliopisto.

Eclipse (2008), *Eclipse.org*. WWW-sivusto, URL: http://www.eclipse.org/ (23.2.2008).

Erasala N., Yen D., Rajkumar T.M. Enterprise Application Integration in the electronic commerce world *Computer Standards & Interfaces*, 2189 (toim. Schumny H.), Elsevier Science Ltd., Oxford 2002, 1-14.

Erl, T. (2005) *Service-Oriented Architecture Concepts, Technology, and Design.* Prentice Hall PTR, Indiana.

FCS Partners (2005) *WebServices –ohjemointi Javalla, osa 1*. FCS Partners Oyj, 2005.

Firestone, J. (2002) *Enterprise Information Portals and Knowledge Management*. Butterworth-Heinemann, New York.

Flurry,G, Modh M. (2005) *WebServices development patterns.* IBM, URL http://www.ibm.com/developerworks/websphere/library/techarticles/0511_flurry/0511_flurry.html (23.2.2008).

Gilbert, M., Caldwell F., Hayward S. (2002) *The Smart Enterpise Suite Is Coming: Do We Need It?* Gartner Research.

Gold-Bernstein, B. (1999) EAI Market Segmentation. *EAI Journal*. (July/August).

Google Picasaweb (2008), *Google Picasa Verkkoalbumit.* WWW-sivusto, URL: http://picasaweb.google.fi/home (26.2.2008).

Haajanen J. (2004) *Future e-Business Solutions: Top Down or Bottom(s) Up? – Technological vs. Business Drive Approach to e-Business.* Helsingin yliopisto, tietojenkäsittelytieteen laitos. (Saatavana myös http://www.cs.helsinki.fi/u/chande/courses/cs/WSA/seminar/JanneSavukoski/Savukoski-WS_EAI.pdf, 23.2.2008).

Ferreira, I., Harris-Jones, C. (2004) *Portals and EAI*. Ovum Research.

Hau, T., Ebert, N., Hochstein, A., Brenner, W. (2008) Where to Start with SOA: Criteria for Selecting SOA Projects. *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual* (toim. Sprague, R. Jr.), IEEE Computer Society, Los Alamitos, California, 314-314.

Hirvonen, J. (2008) *Systemoinnin menetelmien harjoitustyö.* Joensuun yliopisto, tietojenkäsittelytiede.

Jetty (2008), *Jetty WebServer*. WWW-sivusto, URL: http://www.mortbay.org/ (23.2.2008).

Jokela, S. (2001) *Metadata Enhanced Content Management in Media Companies.* Helsingin yliopisto, tietojenkäsittelytiede.

JMX (2008) *Java Management Extensions (JMX) Technology*. WWW-sivusto, URL: http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/ (23.2.2008)

JSR 016 (2001) *J2EE Connector Architecture Specification 1.0*. Sun Microsystems. URL: http://java.sun.com/j2ee/connector/download.html (21.2.2008).

JSR 168 (2003) *Java Portlet Specification 1.0.* Sun Microsystems. URL: http://www.jcp.org/en/jsr/detail?id=168 (23.2.2008).

JSR 170 (2005) *Content Repository for Java Technology API.* Sun Microsystems. URL: http://jcp.org/en/jsr/detail?id=170 (23.2.2008).

JSR 208 (2005) *Java Business Integration Specification 1.0.* Sun Microsystems. URL: http://www.jcp.org/en/jsr/detail?id=208 (23.2.2008).

JSR 283 (2007) *Content Repository for Java Technology API Version 2.0.* Sun Microsystems. URL: http://jcp.org/en/jsr/detail?id=283 (23.2.2008).

JSR 286 (2007) *Java Portlet Specification 2.0.* Sun Microsystems. URL: http://jcp.org/en/jsr/detail?id=286 (23.2.2008).

Kalakota, R., Robinson, M. (2001). *e-Business 2.0 – Roadmap for success*. Addison-Wesley, USA.

Kiviniemi, P. (2002) *SOAP*. Helsingin yliopisto, tietojenkäsittelytiede. URL: http://www.cs.helsinki.fi/u/pjkivini/semi2/SOAP.pdf (31.01.2005).

Kendall, S., Waldo J., Wollrath A. Wyant G. (1994) *Note on Distributed Computing*, Sun Microsystems Laboratories, Inc. Mountain view, Canada. (Saatavana myös: http://research.sun.com/techrep/1994/smli_tr-94-29.pdf, 23.2.2008).

Loughran, S., Smith E. (2005) *Rethinking the Java SOAP Stack.* HP Laboratories, Bristol. URL: http://www.hpl.hp.com/techreports/2005/HPL-2005-83.pdf (23.2.2008).

Murray, G. (1999) *The Portal is the desktop*. http://www.e-promag.com/eparchive/index.cfm?fuseaction=viewarticle&ContentID=166&websiteid= (6.1.2004).

O'Murchu, I., Breslin, J., Decker S. (2004) Online Social and Business Networking Communities. *Proceedings of the ECAI 2004 Workshop on Application of Semantic Web Technologies to WebCommunities*, Valencia, Espanja. (Saatavana myös: http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-107/paper2.pdf, 23.2.2008)

OASIS (2004a) UDDI *Executive Overview: Enabling Service-Oriented Architecture.* Organization for the Advancement of Structured Information Standards. WWW-sivusto, http://uddi.org/pubs/uddi-exec-wp.pdf (23.2.2008).

OASIS (2004b) *Introduction to UDDI: Important Features and Functional Concepts.* Organization for the Advancement of Structured Information Standards. WWW-sivusto, http://uddi.org/pubs/uddi-tech-wp.pdf (23.2.2008).

OASIS UDDI (2004) *UDDI Version 3.0.2.* OASIS UDDI Specification TC, Oasis Open, 2004 URL: http://www.uddi.org/pubs/uddi_v3.htm, (23.2.3008).

OECD (2007) *Participative Web and User Created Content – Web 2.0, Wikis and Social Networking*. OECD Publishing. (Saatavana myös: http://213.253.134.43/oecd/pdfs/browseit/9307031E.PDF, 23.2.2008).

Patil, S. (2006) *What is Java Content Repository.* WWW-sivusto,URL: http://www.onjava.com/pub/a/onjava/2006/10/04/what-is-java-content-repository.html?page=1 (23.2.2008).

Pilgrim, M. (2002) *What is RSS?* URL: http://www.xml.com/pub/a/2002/12/18/dive-into-xml.html (23.2.2008).

Plumtree Software (1999) Plumtree Corporate Portal 3.0. URL: http://www.findarticles.com/cf_dls/m4PRN/1999_July_12/55126930/p1/article.jhtml (6.1.2004).

Richards M. (2006) *The Role of the Enterprise Service Bus.* URL: http://www.infoq.com/presentations/Enterprise-Service-Bus (23.2.2008).

Reynolds, H., Koulopoulos, T. (1999) Enterprise Knowledge Has a Face. *Intelligent Enterprise*, 2 (5), 29-34, URL: http://www.intelligententerprise.com/db_area/archives/1999/993003/feat1.shtml (23.2.2008).

Rockley, A., Kostur, P., Manning, S. (2003) *Managing Enterprise Content.* New Riders, Indiana.

Rosendahl, M., Rune, E. (2000) *Enterprise Application integration – Strategies for the Telecommunication Market.* Uppsala University, Uppsala.

Sadhwani, D., Samtani G. (2002) *EAI and Web Services - Easier Enterprise Application Integration?* URL: http://www.webservicesarchitect.com/content/articles/samtani01.asp (23.2.2008).

SCA (2007) *Service Component Architecture Specifications.* Open Service Oriented Arcitecture collaboration, WWW-sivusto, URL: http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications (23.2.2008).

Shilakes, C., Tylman, J. (1998) *Enterprise Information Portals*. Merrill Lynch & Co., New York.

Shin, S. (2003) *Secure Web services.* WWW-sivusto, URL: http://www.javaworld.com/javaworld/jw-03-2003/jw-0321-wssecurity.html (23.2.2008).

Sidorof, T. (2005) *Semanttiset portaalit*. Helsingin yliopisto, tietojenkäsittelytiede. (Saatavana myös: http://www.seco.tkk.fi/publications/2005/sidoroff-semanttiset-portaalit-2005.ps, 23.2.2008).

Swartz, A. (2000) *RDF Site Summary (RSS) 1.0.* RSS-DEV Working Group. URL: http://web.resource.org/rss/1.0/ (23.2.2008).

Tahvanainen, H. (2006) *XML ja monikanavajulkaiseminen.* Joensuun yliopisto, tietojenkäsittelytiede. (Saatavana myös: ftp://cs.joensuu.fi/pub/Theses/2005_MSc_Tahvanainen_Hanna.pdf, 31.01.2006).

Tost, A. (2006) Building a powerful, reliable SOA with JMS and WebSphere ESB – Part 1, *IBM WebSphere Developer Technical Journal*. IBM, URL: http://www.ibm.com/developerworks/websphere/techjournal/0602_tost/0602_tost.html (23.2.2008).

W3C (2001) *Web Service Description Language (WSDL) 1.1*. World Wide Web Consortium, 2000. URL: http://www.w3.org/TR/wsdl (23.2.2008).

W3C (2004a) *Resource Description Framework (RDF).* World Wide Web Consortium, 2004. URL: http://www.w3.org/RDF/ (23.2.2008).

W3C (2004b) *Web Services Architecture.* World Wide Web Consortium, 2004. (Saatavana myös: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/, 23.2.2008).

W3C (2004c) *XML Schema Part 2: Datatypes Second Edition.* World Wide Web Consortium, 2004. URL: http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/ (23.2.2008).

W3C (2005) *Web Service Semantics – WSDL-S.* World Wide Web Consortium, 2005. URL: http://www.w3.org/Submission/WSDL-S/ (21.2.2008).

W3C (2006) *XML 1.0, Fourth Edition.* World Wide Web Consortium, 2006. URL: http://www.w3.org/TR/2006/REC-xml-20060816/ (23.2.2008).

W3C (2007) *SOAP Version 1.2 Part 0: Primer (Second Edition).* World Wide Web Consortium, 2007. URL http://www.w3.org/TR/2007/REC-soap12-part0-20070427/ (23.2.2008).

W3Schools (2006) *Introduction to XML Schema.* WWW-sivusto, http://www.w3schools.com/schema/schema_intro.asp (23.2.2008).

Wikipedia (2008) *RSS (file format).* Wikipedia, the Free Encyclopedia, 2008. URL: http://en.wikipedia.org/wiki/RSS (4.3.2008).

WS-TX (2007) *Web Services Transaction 1.1* Oasis Web Services Transaction Technical Committee, 2007. URL: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-tx (21.2.2008).

WS-BPEL (2007) *Web Services Business Process Execution Language Version 2.0.* OASIS Open, 2007. URL: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html (23.2.2007).

WS-Reliability (2004) *Web Services Reliability 1.1* OASIS Open, 2004. URL: http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/wsrm-ws_reliability-1.1-spec-os.pdf (21.2.2008).

WS-Security (2006) *Web Services Security: SOAP Message Security 1.1.* OASIS Open, 2006. URL: http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-errata-os-SOAPMessageSecurity.htm (23.2.2008).

WSRP (2006) *Web Services for Remote Portlets Specification v.2.0, Public Review Draft.* Oasis Open, 2006. URL: http://www.oasis-open.org/committees/download.php/18617/wsrp-2.0-spec-pr-01.html (23.2.2006).

XML Digital Signature (2002) *XML-Signature Syntax and Processing.* W3C Recommendation, 2002. URL: http://www.w3.org/TR/xmldsig-core/ (23.2.2008).

XML Encryption (2002) *XML Enryption Syntax and Processing.* W3C Recommendation, 2002. URL: http://www.w3.org/TR/xmlenc-core/ (23.2.2008).

LIITE 1: Ohje esimerkkisovelluksen kokoamiseksi

Esimerkkisovelluksen koostaminen:

1. Asenna JDK 5.0 (Java Development Kit) (Update 14) (saatavilla osoitteesta http://java.sun.com/javase/downloads/index_jdk5.jsp) esimerkiksi hakemistoon c:\esimerkkisovellus\jdk.
2. Asenna mukana tuleva JRE (Java Runtime Enviroment) esimerkiksi hakemistoon c:\esimerkkisovellus\jre.
3. Aseta JAVA_HOME ympäristömuuttuja osoittamaan JDK asennushakemistoon (vaikkapa komennolla set JAVA_HOME=c:\esimerkkisovellus\jdk), ja lisäksi lisää molempien asennettujen sovellusten bin-hakemistot PATH-ympäristömuuttujaan. (esimerkiksi komennolla set PATH=%PATH%;c:\esimerkkisovellus\jdk\bin;c:\esimerkkisovellus\jre\bin).
4. Lataa Pluton versio 1.1.4 Tomcatin kera (kirjoitushetkellä pluto-current-bundle.zip) (saatavilla osoitteesta http://portals.apache.org/pluto/) ja pura se hakemistoon c:\esimerkkisovellus.
5. Lataa Apache Ant versio 1.7.0, tiedosto apache-ant-1.7.0-bin.zip (saatavilla http://ant.apache.org/) ja pura se hakemistoon c:\esimerkkisovellus.
6. Aseta ANT_HOME ympäristömuuttuja Ant-asennushakemistoon (esim. komennolla set ANT_HOME=c:\esimerkkisovellus\apache-ant-1.7.0) ja lisää sen bin-alihakemisto PATH-ympäristömuuttujaan (esimerkiksi komennolla set PATH=%PATH%;%ANT_HOME%\bin).
7. Lataa ja pura Maven 2.0.7, tiedosto maven-2.0.7-bin.zip (http://maven.apache.org/) hakemistoon c:\esimerkkisovellus.
8. Aseta MAVEN_HOME (set MAVEN_HOME=C:\esimerkkisovellus\maven-2.0.7) ja lisää bin-hakemisto PATH-ympäristömuuttujaan (set PATH=%PATH%;%MAVEN_HOME%\bin).
9. Lisää MAVEN_HOME\conf hakemistossa olevaan settings.xml-tiedostoon kohtaan <servers>.
   ```
   <server>
     <id>pluto</id>
     <username>pluto</username>
     <password>pluto</password>
   </server>.
   ```
10. Lataa Apache Axis 1.4, tiedosto axis-bin-1_4.zip (http://ws.apache.org/axis/) ja pura se hakemistoon c:\esimerkkisovellus.
11. Luo hakemisto c:\esimerkkisovellus\repository\conf ja kopioi sinne tiedostot repository.xml ja jaas.config.
12. Luo hakemisto c:\esimerkkisovellus\repository\repository.
13. Luo hakemisto c:\esimerkkisovellus\components.
14. Aja em. hakemistossa komento:
    ```
    mvn archetype:create
    -DgroupId=fi.joensuu.cs.jhirvone.contentservice
    -DartifactId=contentservice
    -DarchetypeArtifactId=maven-archetype-webapp
    ```
15. Luo hakemisto c:\esimerkkisovellus\components\contentservice\src\main\xml ja kopioi sinne ContentService.wsdl ja RSS20.xsd.
16. Kopioi pom.xml ja project.properties hakemistoon c:\esimerkkisovellus\components\contentservice ja muuta properties-tiedostoa, mikäli tarvetta.
17. Hakemistossa c:\esimerkkisovellus\components\contentservice aja komento mvn generate-sources.
18. Kopioi hakemistoon c:\esimerkkisovellus\components\contentservice\src\webapp\WEB-INF tiedostot web.xml ja server-config.wsdd.
19. Luo hakemisto c:\esimerkkisovellus\components\contentservice\src\webapp\META-INF ja kopioi sinne context.xml.
20. Korvaa C:\esimerkkisovellus\components\contentservice\src\java\fi\joensuu\cs\jhirvone\contentservice\service muodostunut tiedosto ContentBindingImpl.java tiedosto liitteen 2 mukaisella tiedostolla.
21. Aja hakemistossa C:\esimerkkisovellus\components\contentservice komento mvn –Dmaven.test.skip=true package.
22. Käynnistä toinen komentorivikehoite ja aseta siihen em. ympäristömuuttujat, siirry hakemistoon C:\esimerkkisovellus\pluto-1.1.4\bin ja käynnistä Tomcat komennolla catalina run.
23. Aja hakemistossa C:\esimerkkisovellus\components\contentservice komento mvn –Dmaven.test.skip=true tomcat:deploy, joka asentaa contentservice-palvelun Tomcat-sovelluspalvelimelle.
24. Aja hakemistossa C:\esimerkkisovellus\components\contentservice komento mvn integration-test, joka suorittaa palvelun asentamisen axis-ympäristöön. Voit varmentua asennuksen onnistumisesta avaamalla selaimeen osoitteen http://localhost:8080/contentservice/services.
25. Luo hakemisto C:\esimerkkisovellus\components\loggerservice-bean-su. Kopioi sinne vastaava pom.xml.
26. Luo hakemisto C:\esimerkkisovellus\components\loggerservice-bean-su\src\main\java\fi\joensuu\cs\jhirvone\loggerservice ja kopioi sinne LoggerService.java.

27. Luo hakemisto C:\esimerkkisovellus\components\loggerservice-bean-su\src\main\resources ja kopioi sinne log4j.properties ja xbean.xml.
28. Hakemistossa C:\esimerkkisovellus\components\loggerservice-bean-su aja komento mvn install.
29. Luo hakemisto C:\esimerkkisovellus\components\loggerservice-http-bc ja kopioi sinne vastaava pom.xml.
30. Luo hakemisto C:\esimerkkisovellus\components\loggerservice-http-bc\src\main\resources ja kopioi sinne vastaava xbean.xml.
31. Hakemistossa C:\esimerkkisovellus\components\loggerservice-http-bc aja komento mvn install.
32. Luo hakemisto C:\esimerkkisovellus\components\contentservice-http-bc ja kopioi sinne vastaava pom.xml.
33. Luo hakemisto C:\esimerkkisovellus\components\contentservice-http-bc\src\main\resources ja kopioi sinne vastaava xbean.xml.
34. Hakemistossa C:\esimerkkisovellus\components\contentservice-http-bc aja komento mvn install.
35. Luo hakemisto C:\esimerkkisovellus\loggerservice-sa ja kopioi sinne vastaava pom.xml. Aja komento mvn install.
36. Käynnistä servicemix komennolla
    mvn -Dlog4j.configuration= \
    file:C:\esimerkkisovellus\components\loggerservice-bean-su\src\main\resources\log4j.properties \
    jbi:servicemix.
37. Käynnistä kolmas komentokehoite ja aseta siihen ympäristömuuttujat.
38. Hakemistossa c:\esimerkkisovellus\components aja komento:
    mvn archetype:create
    -DgroupId=fi.joensuu.cs.jhirvone.contentservice
    -DartifactId=contentportlet
    -DarchetypeArtifactId=maven-archetype-portlet
39. Poista hakemistosta C:\esimerkkisovellus\components\contentportlet\src\main\java\fi\joensuu\cs\jhirvone\ContentService tiedosto MyPortlet,java. Kopioi samaan hakemistoon tiedosto ContentPortlet.java.
40. Luo hakemisto C:\esimerkkisovellus\components\contentportlet\src\main\webapp\META-INF ja kopioi sinne context.xml.
41. Muokkaa hakemistossa C:\esimerkkisovellus\components\contentportlet\src\main\webapp\WEB-INF tiedostot portlet.xml ja web.xml esitetyn mukaiseksi.
42. Poista hakemistosta C:\esimerkkisovellus\components\contentportlet\src\main\webapp generoidut jsp-sivut. Luo alikansio jsp ja kopioi sinne editItem.jsp, editPage.jsp, help.jsp ja view.jsp.
43. Kopioi hakemistoon c:\esimerkkisovellus\components\contentportlet vastaava pom.xml.
44. Samassa hakemistossa suorita komento mvn generate-sources package.
45. Asenna portletti suorittamalla komento mvn tomcat:deploy.
46. Avaa selaimella osoite http://localhost:8080/pluto ja kirjaudu tunnuksilla pluto / pluto
47. Valitse Pluto Admin välilehti. Valitse Portlet Aplications kohdasta ContentPortlet ja viereisestä valintalaatikosta contentportlet ja paina AddPortlet.
48. Jos kaikki edellä mainitut kohdat ovat menneet oikein, etusivulle ilmestyy contentportlet, jota käyttämällä saa luotua oman linkkilistan.

LIITE 2: Esimerkkisovelluksen lähdekoodit, contentservice-komponentti

C:\esimerkkisovellus\components\contentservice\pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                http://maven.apache.org/maven-v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>fi.joensuu.cs.jhirvone.contentservice</groupId>
        <artifactId>contentservice</artifactId>
        <packaging>war</packaging>
        <version>1.0-SNAPSHOT</version>
        <name>contentservice Maven Webapp</name>
        <url>http://maven.apache.org</url>
        <properties>
                <axis.version>1.4</axis.version>
        </properties>
        <dependencies>
                <dependency>
                        <groupId>org.apache.jackrabbit</groupId>
                        <artifactId>jackrabbit-api</artifactId>
                        <version>1.2.2</version>
                </dependency>
                <dependency>
                        <groupId>org.apache.jackrabbit</groupId>
                        <artifactId>jackrabbit-core</artifactId>
                        <version>1.2.2</version>
                </dependency>
                <dependency>
                        <groupId>org.apache.jackrabbit</groupId>
                        <artifactId>jackrabbit-jcr-commons</artifactId>
                        <version>1.2.2</version>
                </dependency>
                <dependency>
                        <groupId>commons-logging</groupId>
                        <artifactId>commons-logging</artifactId>
                        <version>1.0.4</version>
                        <scope>compile</scope>
                </dependency>
                <dependency>
                        <groupId>junit</groupId>
                        <artifactId>junit</artifactId>
                        <version>4.4</version>
                        <scope>test</scope>
                </dependency>
                <dependency>
                        <groupId>axis</groupId>
                        <artifactId>axis</artifactId>
                        <version>${axis.version}</version>
                </dependency>
                <dependency>
                        <groupId>axis</groupId>
                        <artifactId>axis-jaxrpc</artifactId>
                        <version>${axis.version}</version>
                </dependency>
                <dependency>
                        <groupId>axis</groupId>
                        <artifactId>axis-wsdl4j</artifactId>
                        <version>1.5.1</version>
                </dependency>
                <dependency>
                        <groupId>axis</groupId>
                        <artifactId>axis-saaj</artifactId>
                        <version>${axis.version}</version>
                </dependency>
                <dependency>
                        <groupId>axis</groupId>
                        <artifactId>axis-ant</artifactId>
                        <version>${axis.version}</version>
                </dependency>
                <dependency>
                        <groupId>xalan</groupId>
                        <artifactId>xalan</artifactId>
```

```
                        <version>2.7.0</version>
                </dependency>
        </dependencies>
        <build>
                <finalName>contentservice</finalName>
                <plugins>
                        <plugin>
                                <groupId>org.apache.maven.plugins</groupId>
                                <artifactId>maven-compiler-plugin</artifactId>
                                <configuration>
                                        <source>1.5</source>
                                        <target>1.5</target>
                                </configuration>
                        </plugin>
                        <plugin>
                                <groupId>org.codehaus.mojo</groupId>
                                <artifactId>tomcat-maven-plugin</artifactId>
                                <configuration><server>pluto</server> </configuration>
                        </plugin>
                        <plugin>
                                <groupId>org.codehaus.mojo</groupId>
                                <artifactId>axistools-maven-plugin</artifactId>
                                <configuration>
                                        <sourceDirectory>src/main/xml</sourceDirectory>
                                        <testSourceDirectory>src/test</testSourceDirectory>
                                        <testCases>false</testCases>
                                        <runTestCasesAsUnitTests>false</runTestCasesAsUnitTests>
                                        <serverSide>true</serverSide>
                                        <outputDirectory>src/main/java</outputDirectory>
                                        <subPackageByFileName>false</subPackageByFileName>
                                        <inputFiles>
                                                <inputFile>target/classes/deploy.wsdd</inputFile>
                                        </inputFiles>
                                        <isServerConfig>true</isServerConfig>
                                </configuration>
                                <executions>
                                        <execution>
                                                <phase>generate-sources</phase>
                                                <goals> <goal>wsdl2java</goal> </goals>
                                        </execution>
                                </executions>
                        </plugin>
                </plugins>
        </build>
</project>
```

c:\esimerkkisovellus\repository\conf\jaas.config:

```
Jackrabbit {
  org.apache.jackrabbit.core.security.SimpleLoginModule required anonymousId="anonymous";
};
```

c:\esimerkkisovellus\repository\conf\repository.xml:

```
<!DOCTYPE Repository PUBLIC "-//The Apache Software Foundation//DTD Jackrabbit 1.2//EN"
                            "http://jackrabbit.apache.org/dtd/repository-1.2.dtd">
<!-- Example Repository Configuration File -->
<Repository>
    <FileSystem class="org.apache.jackrabbit.core.fs.local.LocalFileSystem">
        <param name="path" value="${rep.home}/repository"/>
    </FileSystem>
    <Security appName="Jackrabbit">
        <AccessManager class="org.apache.jackrabbit.core.security.SimpleAccessManager">
        </AccessManager>
        <LoginModule class="org.apache.jackrabbit.core.security.SimpleLoginModule">
           <param name="anonymousId" value="anonymous"/>
        </LoginModule>
    </Security>
    <Workspaces rootPath="${rep.home}/workspaces" defaultWorkspace="default"/>
    <Workspace name="${wsp.name}">
        <FileSystem class="org.apache.jackrabbit.core.fs.local.LocalFileSystem">
            <param name="path" value="${wsp.home}"/>
        </FileSystem>
        <PersistenceManager
class="org.apache.jackrabbit.core.persistence.db.DerbyPersistenceManager">
            <param name="url" value="jdbc:derby:${wsp.home}/db;create=true"/>
```

```
            <param name="schemaObjectPrefix" value="${wsp.name}_"/>
        </PersistenceManager>
        <SearchIndex class="org.apache.jackrabbit.core.query.lucene.SearchIndex">
            <param name="path" value="${wsp.home}/index"/>
            <param name="useCompoundFile" value="true"/>
            <param name="minMergeDocs" value="100"/>
            <param name="volatileIdleTime" value="3"/>
            <param name="maxMergeDocs" value="100000"/>
            <param name="mergeFactor" value="10"/>
            <param name="maxFieldLength" value="10000"/>
            <param name="bufferSize" value="10"/>
            <param name="cacheSize" value="1000"/>
            <param name="forceConsistencyCheck" value="false"/>
            <param name="autoRepair" value="true"/>
            <param name="analyzer"
               value="org.apache.lucene.analysis.standard.StandardAnalyzer"/>
            <param name="queryClass" value="org.apache.jackrabbit.core.query.QueryImpl"/>
            <param name="idleTime" value="-1"/>
            <param name="respectDocumentOrder" value="true"/>
            <param name="resultFetchSize" value="2147483647"/>
        </SearchIndex>
    </Workspace>
    <Versioning rootPath="${rep.home}/version">
        <FileSystem class="org.apache.jackrabbit.core.fs.local.LocalFileSystem">
            <param name="path" value="${rep.home}/version"/>
        </FileSystem>
        <PersistenceManager
               class="org.apache.jackrabbit.core.persistence.db.DerbyPersistenceManager">
            <param name="url" value="jdbc:derby:${rep.home}/version/db;create=true"/>
            <param name="schemaObjectPrefix" value="version_"/>
        </PersistenceManager>
    </Versioning>
    <SearchIndex class="org.apache.jackrabbit.core.query.lucene.SearchIndex">
        <param name="path" value="${rep.home}/repository/index"/>
    </SearchIndex>
</Repository>
```

c:\esimerkkisovellus\components\contentservice\src\main\xml\RSS20.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns="http://blogs.law.harvard.edu/RSS20.xsd"
        targetNamespace="http://blogs.law.harvard.edu/RSS20.xsd"
        version="1.1.0">
        <xs:element name="rss">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="channel" type="channel" />
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:complexType name="channel">
                <xs:sequence>
                        <xs:element maxOccurs="1" minOccurs="1" name="title"
                                type="xs:string">
                        </xs:element>
                        <xs:element maxOccurs="1" minOccurs="1" name="link"
                                type="xs:string">
                        </xs:element>
                        <xs:element maxOccurs="1" minOccurs="1" name="description"
                                type="xs:string">
                        </xs:element>
                        <xs:element default="en-us" maxOccurs="1" minOccurs="0"
                                name="language" type="xs:string">
                        </xs:element>
                </xs:sequence>
        </xs:complexType>
</xs:schema>
```

c:\esimerkkisovellus\components\contentservice\src\main\xml\ContentService.wsdl:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns="http://cs.joensuu.fi/jhirvone/types"
        xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
        xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
        xmlns:rss="http://blogs.law.harvard.edu/RSS20.xsd"
```

```
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://cs.joensuu.fi/jhirvone/contentservice/service"
xmlns:types="http://cs.joensuu.fi/jhirvone/types"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://cs.joensuu.fi/jhirvone/contentservice/service">
<wsdl:types>
        <xs:schema xmlns:ref="http://blogs.law.harvard.edu/RSS20.xsd"
                xmlns:rss_type="http://cs.joensuu.fi/jhirvone/types"
                targetNamespace="http://cs.joensuu.fi/jhirvone/contentservice/service">
                <xs:import namespace="http://blogs.law.harvard.edu/RSS20.xsd"
                        schemaLocation="RSS20.xsd" />
                <xs:complexType name="RSSMessageType">
                        <xs:sequence>
                           <xs:element maxOccurs="1" minOccurs="1"
                                nillable="true" name="channel" type="rss:channel"/>
                           <xs:element maxOccurs="1" minOccurs="0"
                                nillable="true" name="userid" type="tns:userIdType" />
                        </xs:sequence>
                </xs:complexType>
                <xs:simpleType name="userIdType">
                        <xs:restriction base="xs:string" />
                </xs:simpleType>
                <xs:element name="userid" type="tns:userIdType" />
                <xs:element name="status" type="xs:string" />
                <xs:element name="rss" type="tns:RSSMessageType" />
                <xs:element name="setContentFault" type="xs:string" />
                <xs:element name="getContentFault" type="xs:string" />
        </xs:schema>
</wsdl:types>
<wsdl:message name="getContentRequestMessage">
        <wsdl:part element="tns:userid" name="getContentRequestMessage" />
</wsdl:message>
<wsdl:message name="getContentResponseMessage">
        <wsdl:part element="tns:rss" name="getContentResponseMessage" />
</wsdl:message>
<wsdl:message name="setContentRequestMessage">
        <wsdl:part element="tns:rss" name="setContentRequestMessage" />
</wsdl:message>
<wsdl:message name="setContentResponseMessage">
        <wsdl:part element="tns:status" name="setContentResponseMessage" />
</wsdl:message>
<wsdl:message name="setContentFaultMsg">
        <wsdl:part element="tns:setContentFault" name="setContentFault" />
</wsdl:message>
<wsdl:message name="getContentFaultMsg">
        <wsdl:part element="tns:getContentFault" name="getContentFault" />
</wsdl:message>
<wsdl:portType name="SOAPport">
        <wsdl:operation name="getContent">
                <wsdl:input message="tns:getContentRequestMessage" />
                <wsdl:output message="tns:getContentResponseMessage" />
                <wsdl:fault message="tns:getContentFaultMsg" name="getContentFault" />
        </wsdl:operation>
        <wsdl:operation name="setContent">
                <wsdl:input message="tns:setContentRequestMessage" />
                <wsdl:output message="tns:setContentResponseMessage" />
                <wsdl:fault name="setContentFault" message="tns:setContentFaultMsg">
                </wsdl:fault>
        </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="ContentBinding" type="tns:SOAPport">
        <soap:binding style="document"
                transport="http://schemas.xmlsoap.org/soap/http" />
        <wsdl:operation name="getContent">
                <soap:operation soapAction="getContent" />
                <wsdl:input>
                  <soap:body parts="getContentRequestMessage" use="literal" />
                </wsdl:input>
                <wsdl:output>
                  <soap:body parts="getContentResponseMessage" use="literal" />
                </wsdl:output>
                <wsdl:fault name="getContentFault">
                        <soap:fault name="getContentFault" use="literal" />
                </wsdl:fault>
        </wsdl:operation>
```

```
                <wsdl:operation name="setContent">
                        <soap:operation soapAction="setContent" />
                        <wsdl:input>
                          <soap:body parts="setContentRequestMessage" use="literal" />
                        </wsdl:input>
                        <wsdl:output>
                          <soap:body parts="setContentResponseMessage" use="literal" />
                        </wsdl:output>
                        <wsdl:fault name="setContentFault">
                                <soap:fault name="setContentFault" use="literal" />
                        </wsdl:fault>
                </wsdl:operation>
        </wsdl:binding>
        <wsdl:service name="ContentService">
                <wsdl:port binding="tns:ContentBinding" name="ContentService">
                        <soap:address
                location="http://localhost:8080/contentservice/services/ContentService" />
                </wsdl:port>
        </wsdl:service>
</wsdl:definitions>
```

C:\esimerkkisovellus\components\contentservice\project.properties:

```
axis.home = c:/esimerkkisovellus/axis-1_4
service.name=contentservice
service.wsdl = src/main/xml/${service.name}.wsdl
src.dir = src/main/java
build.dir = target
target.dir = target
webserver.axis.dir = c:/esimerkkisovellus/pluto-1.1.4/webapps
wsdl.namespace = http://cs.joensuu.fi/jhirvone/ContentService/service
wsdl.package = fi.joensuu.cs.jhirvone.contentservice.ContentService
type.namespace = http://blogs.law.harvard.edu/RSS20_xsd
type.package = edu.harvard.law.blogs.RSS20_xsd
dist.file = contentservice.war
deploy.file = ${src.dir}/fi/joensuu/cs/jhirvone/ContentService/service/deploy.wsdd
undeploy.file = ${src.dir}/fi/joensuu/cs/jhirvone/ContentService/service/undeploy.wsdd
admin.servletpath=contentservice/services/AdminService
deploy.host = localhost
deploy.port = 8080
tomcat.home=c:/esimerkkisovellus/pluto-1.1.4/
tomcat.manager.url=http://${deploy.host}:${deploy.port}/manager
tomcat.manager.username = pluto
tomcat.manager.password = pluto
```

C:\esimerkkisovellus\components\contentservice\src\main\webapp\WEB-INF\web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
        xmlns="http://java.sun.com/xml/ns/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
        <display-name>contentservice</display-name>
        <resource-env-ref>
                <description>Content Repository</description>
                <resource-env-ref-name>jcr/repository</resource-env-ref-name>
                <resource-env-ref-type>javax.jcr.Repository</resource-env-ref-type>
        </resource-env-ref>
        <servlet>
                <display-name>Apache-Axis Servlet</display-name>
                <servlet-name>AxisServlet</servlet-name>
                <servlet-class>org.apache.axis.transport.http.AxisServlet</servlet-class>
        </servlet>
        <servlet>
                <display-name>Axis Admin Servlet</display-name>
                <servlet-name>AdminServlet</servlet-name>
                <servlet-class>org.apache.axis.transport.http.AdminServlet</servlet-class>
                <load-on-startup>100</load-on-startup>
        </servlet>
        <servlet-mapping>
                <servlet-name>AxisServlet</servlet-name>
                <url-pattern>/servlet/AxisServlet</url-pattern>
        </servlet-mapping>
        <servlet-mapping>
                <servlet-name>AxisServlet</servlet-name>
```

```
            <url-pattern>*.jws</url-pattern>
        </servlet-mapping>
        <servlet-mapping>
            <servlet-name>AxisServlet</servlet-name>
            <url-pattern>/services/*</url-pattern>
        </servlet-mapping>
        <servlet-mapping>
            <servlet-name>AdminServlet</servlet-name>
            <url-pattern>/servlet/AdminServlet</url-pattern>
        </servlet-mapping>
        <welcome-file-list>
            <welcome-file>index.html</welcome-file>
            <welcome-file>index.htm</welcome-file>
            <welcome-file>index.jsp</welcome-file>
            <welcome-file>default.html</welcome-file>
            <welcome-file>default.htm</welcome-file>
            <welcome-file>default.jsp</welcome-file>
        </welcome-file-list>
</web-app>
```

C:\esimerkkisovellus\components\contentservice\src\main\webapp\WEB-INF\server-config.wsdd:

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
        xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
        <globalConfiguration>
           <parameter name="adminPassword" value="admin" />
           <parameter name="enableNamespacePrefixOptimization"
                value="false" />
           <parameter name="dotNetSoapEncFix" value="true" />
           <parameter name="disablePrettyXML" value="true" />
           <parameter name="attachments.implementation"
                value="org.apache.axis.attachments.AttachmentsImpl" />
           <parameter name="sendXsiTypes" value="true" />
           <parameter name="sendMultiRefs" value="true" />
           <parameter name="sendXMLDeclaration" value="true" />
           <requestFlow>
                <handler type="java:org.apache.axis.handlers.JWSHandler">
                        <parameter name="scope" value="session" />
                </handler>
                <handler type="java:org.apache.axis.handlers.JWSHandler">
                        <parameter name="scope" value="request" />
                        <parameter name="extension" value=".jwr" />
                </handler>
           </requestFlow>
        </globalConfiguration>
        <handler name="LocalResponder"
           type="java:org.apache.axis.transport.local.LocalResponder" />
        <handler name="URLMapper"
           type="java:org.apache.axis.handlers.http.URLMapper" />
        <handler name="Authenticate"
           type="java:org.apache.axis.handlers.SimpleAuthenticationHandler" />
        <service name="AdminService" provider="java:MSG">
           <parameter name="allowedMethods" value="AdminService" />
           <parameter name="enableRemoteAdmin" value="false" />
           <parameter name="className" value="org.apache.axis.utils.Admin" />
           <namespace>http://xml.apache.org/axis/wsdd/</namespace>
        </service>
        <service name="Version" provider="java:RPC">
           <parameter name="allowedMethods" value="getVersion" />
           <parameter name="className" value="org.apache.axis.Version" />
        </service>
        <service name="ContentService" provider="java:RPC" style="document"
           use="literal">
           <operation name="getContent" qname="getContent"
                returnQName="ns1:rss" returnType="ns1:RSSMessageType"
                soapAction="getContent"
                xmlns:ns1="http://cs.joensuu.fi/jhirvone/contentservice/service">
                <parameter qname="ns1:userid" type="ns1:userIdType" />
                <fault class="fi.joensuu.cs.jhirvone.contentservice.service.GetContentFaultMsg"
                        qname="ns1:getContentFault" type="xsd:string"
                        xmlns:xsd="http://www.w3.org/2001/XMLSchema" />
           </operation>
           <operation name="setContent" qname="setContent"
                returnQName="ns2:status" returnType="xsd:string"
                soapAction="setContent"
                xmlns:ns2="http://cs.joensuu.fi/jhirvone/contentservice/service"
```

```
                xmlns:xsd="http://www.w3.org/2001/XMLSchema">
                <parameter qname="ns2:rss" type="ns2:RSSMessageType" />
                <fault class="fi.joensuu.cs.jhirvone.contentservice.service.SetContentFaultMsg"
                       qname="ns2:setContentFault" type="xsd:string" />
            </operation>
            <parameter name="allowedMethods" value="setContent getContent" />
            <parameter name="typeMappingVersion" value="1.2" />
            <parameter name="wsdlPortType" value="SOAPport" />
            <parameter name="className"
                value="fi.joensuu.cs.jhirvone.contentservice.service.ContentBindingImpl" />
            <parameter name="wsdlServicePort" value="ContentService" />
            <parameter name="wsdlTargetNamespace"
                value="http://cs.joensuu.fi/jhirvone/contentservice/service" />
            <parameter name="wsdlServiceElement" value="ContentService" />
            <parameter name="schemaUnqualified"
                value="http://cs.joensuu.fi/jhirvone/contentservice/service,
                       http://blogs.law.harvard.edu/RSS20.xsd" />
             <typeMapping
                deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
                encodingStyle="" qname="ns3:channel"
                serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
                type="java:edu.harvard.law.blogs.RSS20_xsd.Channel"
                xmlns:ns3="http://blogs.law.harvard.edu/RSS20.xsd" />
            <typeMapping
                deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
                encodingStyle="" qname="ns5:RSSMessageType"
                serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
                type="java:fi.joensuu.cs.jhirvone.contentservice.service.RSSMessageType"
                xmlns:ns5="http://cs.joensuu.fi/jhirvone/contentservice/service" />
            <typeMapping
                deserializer="org.apache.axis.encoding.ser.SimpleDeserializerFactory"
                encodingStyle="" qname="ns7:userIdType"
                serializer="org.apache.axis.encoding.ser.SimpleSerializerFactory"
                type="java:java.lang.String"
                xmlns:ns7="http://cs.joensuu.fi/jhirvone/contentservice/service" />
            <typeMapping
                deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
                encodingStyle="" qname="ns11:item"
                serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
                type="java:edu.harvard.law.blogs.RSS20_xsd.Item"
                xmlns:ns11="http://blogs.law.harvard.edu/RSS20.xsd" />
        </service>
        <transport name="http">
            <requestFlow>
                <handler type="URLMapper" />
                <handler type="java:org.apache.axis.handlers.http.HTTPAuthHandler" />
            </requestFlow>
            <parameter name="qs:list"
                value="org.apache.axis.transport.http.QSListHandler" />
            <parameter name="qs:wsdl"
                value="org.apache.axis.transport.http.QSWSDLHandler" />
            <parameter name="qs:method"
                value="org.apache.axis.transport.http.QSMethodHandler" />
        </transport>
        <transport name="local">
            <responseFlow>
                <handler type="LocalResponder" />
            </responseFlow>
        </transport>
</deployment>
```

C:\esimerkkisovellus\components\contentservice\src\main\webapp\META-INF\context.xml:

```
<Context reloadable="true" antiJARLocking="true" antiResourceLocking="true">
<Resource name="jcr/repository"
          auth="Container"
          type="javax.jcr.Repository"
          factory="org.apache.jackrabbit.core.jndi.BindableRepositoryFactory"
          configFilePath="/esimerkkisovellus/repository/conf/repository.xml"
          repHomeDir="/esimerkkisovellus/repository"/>
</Context>
```

C:\esimerkkisovellus\components\contentservice\src\main\java\fi\joensuu\cs\jhirvone\contentservice\service\ContentBindingImpl.java:

```
package fi.joensuu.cs.jhirvone.contentservice.service;
import javax.jcr.Node;
import javax.jcr.NodeIterator;
import javax.jcr.Repository;
import javax.jcr.RepositoryException;
import javax.jcr.Session;
import javax.jcr.SimpleCredentials;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import edu.harvard.law.blogs.RSS20_xsd.Channel;
import edu.harvard.law.blogs.RSS20_xsd.Item;
public class ContentBindingImpl implements
        fi.joensuu.cs.jhirvone.contentservice.service.SOAPport {

        public fi.joensuu.cs.jhirvone.contentservice.service.RSSMessageType getContent(
                        java.lang.String getContentRequestMessage)
                           throws java.rmi.RemoteException,
                           fi.joensuu.cs.jhirvone.contentservice.service.GetContentFaultMsg{
                Channel ch = new Channel();
                String user = getContentRequestMessage;
                if (user == null || user.trim() == "") {
                        throw new GetContentFaultMsg("not valid user");
                }
                try {
                        // login to repository
                        Repository repository = this.getEmbeddedRepository();
                        SimpleCredentials cred = new SimpleCredentials(user, "".toCharArray());
                        Session session = repository.login(cred, null);
                        // get root node
                        Node root = session.getRootNode();
                        // check if user has nodes
                        if (root.hasNode(user)) {
                                // get content and create channel-object for output
                                Node channel = root.getNode(user);
                                ch.setTitle(channel.getProperty("title").getString());
                                ch.setLink(channel.getProperty("link").getString());

                                ch.setDescription(channel.getProperty(
                                        "description").getString());
                                try {
                                        int amount = Integer.parseInt(channel.getProperty(
                                                        "itemCount").getString());
                                        Item[] items = new Item[amount];
                                        for (int i = 0; i < amount;) {
                                                Node node = channel.getNode("item" + i);

                                                Item item = new Item();
                                                item.setTitle(node.getProperty(
                                                        "title").getString());
                                                item.setLink(node.getProperty(
                                                        "link").getString());
                                                item.setDescription(node.getProperty(
                                                        "description").getString());
                                                items[i] = item;
                                                i++;
                                        }
                                        ch.setItem(items);
                                } catch (Exception e) {
                                        // no items
                                        ch.setItem(null);
                                        throw new GetContentFaultMsg(
                                                "No content found for user "+ user);
                                }
                        } else
                                throw new GetContentFaultMsg(
                                        "No content found for user " + user);
                        session.logout();
                } catch (NamingException e) {
                        e.printStackTrace();
                        throw new GetContentFaultMsg("Naming exception");
                } catch (RepositoryException e) {
                        e.printStackTrace();
                        throw new GetContentFaultMsg("RepositoryException");
                } catch (Exception e) {
                        e.printStackTrace();
                        throw new GetContentFaultMsg("Unknown exception");
```

```
		}
		// return content and user as requested
		return new RSSMessageType(ch, user);
	}
	public java.lang.String setContent(
		fi.joensuu.cs.jhirvone.contentservice.service.RSSMessageType
			setContentRequestMessage)
			throws java.rmi.RemoteException,
			fi.joensuu.cs.jhirvone.contentservice.service.SetContentFaultMsg {
		String status = "OK";
		try {
			String user = setContentRequestMessage.getUserid();
			Repository repository = this.getEmbeddedRepository();
			SimpleCredentials cred = new SimpleCredentials(user, "".toCharArray());
			Session session = repository.login(cred, null);
			// Workspace ws = session.getWorkspace();
			Node root = session.getRootNode();
			Channel ch = setContentRequestMessage.getChannel();
			// remove existing
			if (root.hasNode(user)) {
				NodeIterator nodes = root.getNodes(user);
				while (nodes.hasNext()) nodes.nextNode().remove();
				session.save();
			}
			// Store content
			Node channel = root.addNode(user);
			channel.setProperty("title", ch.getTitle());
			channel.setProperty("link", ch.getLink());
			channel.setProperty("description", ch.getDescription());
			if (ch.getItem() != null) {
				channel.setProperty("itemCount", ch.getItem().length);
				for (int i = 0; i < ch.getItem().length;) {
					Node item = channel.addNode("item" + i);
					Item it = ch.getItem(i);
					if (it != null) {
						item.setProperty("title", it.getTitle());
						item.setProperty("link", it.getLink());
						item.setProperty("description",
							it.getDescription());
					}
					i++;
				}
			}
			session.save();
			session.logout();
		} catch (NamingException e) {
			status = "ERROR - Repository not configured correctly";
			e.printStackTrace();
			throw new SetContentFaultMsg(status);
		} catch (RepositoryException e) {
			status = "ERROR - Error accessing repository";
			e.printStackTrace();
			throw new SetContentFaultMsg(status);
		} catch (Exception e) {
			status = "ERROR - Unknown error";
			e.printStackTrace();
			throw new SetContentFaultMsg(status);
		}
		return status;
	}
	private Repository getEmbeddedRepository() throws NamingException,
			RepositoryException {
		InitialContext context = new InitialContext();
		Context environment = (Context) context.lookup("java:comp/env");
		Repository repository = (Repository) environment
				.lookup("jcr/repository");
		return repository;
	}
}
```

LIITE 3: Esimerkkisovelluksen lähdekoodit, JBI-komponentit

C:\esimerkkisovellus\components\loggerservice-bean-su\pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                http://maven.apache.org/maven-v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>fi.joensuu.cs.jhirvone.loggerservice</groupId>
        <artifactId>loggerservice-bean-su</artifactId>
        <packaging>jbi-service-unit</packaging>
        <version>0.1</version>
        <name>LoggerService</name>
        <url>http://cs.joensuu.fi/jhirvone</url>
        <properties>
                <servicemix-version>3.2.1</servicemix-version>
        </properties>
        <dependencies>
                <dependency>
                        <groupId>org.apache.servicemix</groupId>
                        <artifactId>servicemix-bean</artifactId>
                        <version>${servicemix-version}</version>
                </dependency>
                <dependency>
                        <groupId>org.apache.servicemix</groupId>
                        <artifactId>servicemix-core</artifactId>
                        <version>${servicemix-version}</version>
                        <scope>provided</scope>
                </dependency>
                <dependency>
                        <groupId>log4j</groupId>
                        <artifactId>log4j</artifactId>
                        <version>1.2.14</version>
                        <scope>compile</scope>
                </dependency>
        </dependencies>
        <build>
                <sourceDirectory>src/main/java</sourceDirectory>
                <resources>
                        <resource>
                                <directory>src/main/resources</directory>
                                <includes>
                                        <include>**/*</include>
                                </includes>
                        </resource>
                </resources>
                <plugins>
                        <plugin>
                                <groupId>org.apache.servicemix.tooling</groupId>
                                <artifactId>jbi-maven-plugin</artifactId>
                                <version>${servicemix-version}</version>
                                <extensions>true</extensions>
                        </plugin>
                        <plugin>
                                <groupId>org.apache.maven.plugins</groupId>
                                <artifactId>maven-compiler-plugin</artifactId>

                                <configuration>
                                        <source>1.5</source>
                                        <target>1.5</target>
                                </configuration>
                        </plugin>
                </plugins>
        </build>
</project>
```

C:\esimerkkisovellus\components\loggerservice-bean-su\src\main\java\fi\joensuu\cs\jhirvone\loggerservice\LoggerService.java:

```
package fi.joensuu.cs.jhirvone.loggerservice;
```

```
import java.io.StringReader;
import org.apache.servicemix.MessageExchangeListener;
import javax.jbi.JBIException;
import javax.jbi.messaging.*;
import javax.annotation.Resource;
import javax.jbi.component.ComponentContext;
import org.apache.log4j.Logger;
import org.apache.servicemix.client.ServiceMixClient;
import org.apache.servicemix.client.ServiceMixClientFacade;
import javax.xml.namespace.QName;
import javax.xml.transform.stream.StreamSource;
import org.apache.servicemix.jbi.util.MessageUtil;
import org.apache.servicemix.jbi.jaxp.SourceTransformer;
/**
 * @author Jukka Hirvonen
 *
 * @since 19.10.2007
 */
public class LoggerService implements MessageExchangeListener {
        @Resource
        DeliveryChannel channel;
        @Resource
        ComponentContext context;
        private static Logger log = Logger.getLogger(LoggerService.class);
        final static String SERVICE_NAMESPACE = "http://cs.joensuu.fi/jhirvone/loggerservice";
        final static String SERVICE_NAMESPACE_PREFIX = "ls";
        final static String DESTINATION_SERVICE = "outputSender";
        /**
         * Method that receives all messages. This method implements
         * MessageExchangeListener's main method.
         *
         * @param exchange
         *
         * @throws MessagingException
         */
        public void onMessageExchange(MessageExchange exchange)
                        throws MessagingException {
                InOut inOut = (InOut) exchange;
                if (inOut.getStatus() == ExchangeStatus.DONE) {
                        log.info("***** DONE EXCHANGE ******");
                        return;
                } else if (inOut.getStatus() == ExchangeStatus.ERROR) {
                        log.error("***** ERROR IN EXCHANGE ******");
                        return;
                }
                log.info("***** BEGIN LOGGING ******");
                String messageString = "";
                try {
                        // Let's save stream to string, so we can reread it later.
                        messageString = new SourceTransformer().contentToString(inOut
                                        .getInMessage());
                } catch (Exception e) {
                        log.error("Invalid request or parser error: ", e);
                        exchange.setStatus(ExchangeStatus.ERROR);
                        channel.send(exchange);
                        return;
                }
                log.info("Exchange (Request): " + inOut.toString());
                log.info("Exchange (Got message): " + messageString);
                log.info("Forwarding request: ");
                // Create a new client with same context, so we can find other
                // pre-configured services
                ServiceMixClient client = new ServiceMixClientFacade(context);
                QName service = new QName(SERVICE_NAMESPACE, DESTINATION_SERVICE,
                                SERVICE_NAMESPACE_PREFIX);
                InOut outexchange = client.createInOutExchange();
                outexchange.setService(service);
                NormalizedMessage message = outexchange.getInMessage();
                message.setContent(new StreamSource(new StringReader(messageString)));
                client.sendSync(outexchange);
                try {

                        client.close();
                } catch (JBIException e) {
                        log.error("JBI Exception:", e);
                        exchange.setStatus(ExchangeStatus.ERROR);
```

```
                channel.send(exchange);
                return;
        }
        String responseString = "";
        try {
                log.info("Response status: " + outexchange.getStatus());
                // Again we have to save response stream to string for reuse
                responseString = new SourceTransformer()
                                .contentToString(outexchange.getOutMessage());

                // now we log message (this consumes stream if used)
                log.info("Got response: " + responseString);
        } catch (Exception e1) {
                log.error("Invalid response", e1);
                exchange.setFault(outexchange.getFault());
                exchange.setStatus(ExchangeStatus.ERROR);
                channel.send(exchange);
                return;
        }

        // Let's create response message, we begin with empty message
        NormalizedMessage outMessage = exchange.createMessage();

        // content is set as source stream generated from response string
        outMessage.setContent(new StreamSource(new StringReader(responseString)));

        // setting created message as response
        MessageUtil.transferToOut(outMessage, exchange);
        log.info("Returning response");
        log.info("Exchange (response): " + exchange.toString());
        channel.send(exchange);
        log.info("***** DONE LOGGING ******");

        }

}
```

C:\esimerkkisovellus\components\loggerservice-bean-su\src\main\resources\xbean.xml:

```
<beans xmlns:bean="http://servicemix.apache.org/bean/1.0"
        xmlns:ls="http://cs.joensuu.fi/jhirvone/loggerservice">
        <!-- Logger Bean -->
        <bean:endpoint service="ls:logger" endpoint="log">
                <bean:bean>
                        <bean
                                class="fi.joensuu.cs.jhirvone.loggerservice.LoggerService" />
                </bean:bean>
        </bean:endpoint>
</beans>
```

C:\esimerkkisovellus\components\loggerservice-bean-su\src\main\resources\log4j.properties:

```
log4j.rootLogger=INFO,out,stdout
log4j.logger.fi.joensuu.cs.jhirvone=INFO
log4j.appender.TR=org.apache.log4j.RollingFileAppender
log4j.appender.TR.File=c:\esimerkkisovellus\loggerservice.log
log4j.appender.TR.layout=org.apache.log4j.PatternLayout
#log4j.appender.TR.layout.ConversionPattern=%d{ABSOLUTE} %p %t %c - %m%n
log4j.appender.TR.layout.ConversionPattern=%d{ABSOLUTE} %-5p %c - %m%n
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %-5p %c - %m%n
log4j.appender.out=org.apache.log4j.FileAppender
log4j.appender.out.layout=org.apache.log4j.PatternLayout
log4j.appender.out.layout.ConversionPattern=%d [%-15.15t] %-5p %-30.30c{1} - %m%n
log4j.appender.out.file=loggerservice.log
log4j.appender.out.append=true
```

C:\esimerkkisovellus\components\loggerservice-http-bc\pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>fi.joensuu.cs.jhirvone.loggerservice</groupId>
        <artifactId>loggerservice-http-bc</artifactId>
```

```
        <packaging>jbi-service-unit</packaging>
        <version>0.1</version>
        <name>LoggerService-http-binding-component</name>
        <url>http://cs.joensuu.fi/jhirvone</url>
        <properties>
                <servicemix-version>3.2.1</servicemix-version>
        </properties>
        <dependencies>
                <dependency>
                        <groupId>org.apache.servicemix</groupId>
                        <artifactId>servicemix-http</artifactId>
                        <version>${servicemix-version}</version>
                </dependency>
        </dependencies>
        <build>
                <resources>
                        <resource>
                                <directory>src/main/resources</directory>
                                <includes>
                                        <include>**/*</include>
                                </includes>
                        </resource>
                </resources>
                <plugins>
                        <plugin>
                                <groupId>org.apache.servicemix.tooling</groupId>
                                <artifactId>jbi-maven-plugin</artifactId>
                                <version>${servicemix-version}</version>
                                <extensions>true</extensions>
                        </plugin>
                </plugins>
        </build>
</project>
```

C:\esimerkkisovellus\components\loggerservice-http-bc\src\main\resources\xbean.xml:

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns:http="http://servicemix.apache.org/http/1.0"
        xmlns:ls="http://cs.joensuu.fi/jhirvone/loggerservice">
        <http:endpoint service="ls:inputReceiver" endpoint="soap"
                role="consumer" targetService="ls:logger" targetEndpoint="log"
                locationURI="http://0.0.0.0:8192/loggerservice/"
                defaultMep="http://www.w3.org/2004/08/wsdl/in-out" soap="true"
                soapVersion="1.1" />
</beans>
```

C:\esimerkkisovellus\components\contentservice-http-bc\pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>fi.joensuu.cs.jhirvone.loggerservice</groupId>
        <artifactId>contentservice-http-bc</artifactId>
        <packaging>jbi-service-unit</packaging>
        <version>0.1</version>
        <name>ContentService-http-binding-component</name>
        <url>http://cs.joensuu.fi/jhirvone</url>
        <properties>
                <servicemix-version>3.2.1</servicemix-version>
        </properties>
        <dependencies>
                <dependency>
                        <groupId>org.apache.servicemix</groupId>
                        <artifactId>servicemix-http</artifactId>
                        <version>${servicemix-version}</version>
                </dependency>
        </dependencies>
        <build>
                <resources>
                        <resource>
                                <directory>src/main/resources</directory>
                                <includes>
                                        <include>**/*</include>
```

```
                        </includes>
                </resource>
        </resources>
        <plugins>
                <plugin>
                        <groupId>org.apache.servicemix.tooling</groupId>
                        <artifactId>jbi-maven-plugin</artifactId>
                        <version>${servicemix-version}</version>
                        <extensions>true</extensions>
                </plugin>
        </plugins>
    </build>
</project>
```

C:\esimerkkisovellus\components\contentservice-http-bc\src\main\resources\xbean.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns:http="http://servicemix.apache.org/http/1.0"
       xmlns:ls="http://cs.joensuu.fi/jhirvone/loggerservice">
       <http:endpoint service="ls:outputSender" endpoint="contentEndpoint"
              role="provider" soap="true" soapVersion="1.1"
              locationURI="http://localhost:8080/contentservice/services/ContentService"
              defaultMep="http://www.w3.org/2004/08/wsdl/in-out" />
</beans>
```

LIITE 4: Esimerkkisovelluksen lähdekoodit, contentportlet

C:\esimerkkisovellus\components\contentportlet\src\main\java\fi\joensuu\cs\jhirvone\contentservice\ContentPortlet.java:

```
package fi.joensuu.cs.jhirvone.contentservice;
import fi.joensuu.cs.jhirvone.contentservice.client.*;
import edu.harvard.law.blogs.RSS20_xsd.*;
import javax.portlet.ActionRequest;
import javax.portlet.ActionResponse;
import javax.portlet.GenericPortlet;
import javax.portlet.PortletContext;
import javax.portlet.PortletException;
import javax.portlet.PortletRequest;
import javax.portlet.PortletRequestDispatcher;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;
import javax.portlet.UnavailableException;
import java.io.IOException;
import java.io.PrintWriter;
import java.lang.reflect.InvocationTargetException;
import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.Arrays;
import javax.portlet.PortletSession;
import javax.xml.rpc.ServiceException;
public class ContentPortlet extends GenericPortlet {
	private String serviceName;
	private String serviceURL;
	private String jspPath;
	private ArrayList<Item> list;
	private Channel channel;
	public void init(javax.portlet.PortletConfig config)
			throws PortletException {
		Channel chan = new Channel();
		try {
			this.jspPath = config.getInitParameter("jsp-path");
			this.setServiceName(config.getInitParameter("service-name"));
			this.setServiceURL(config.getInitParameter("service-url"));
			chan.setDescription(config.getInitParameter("channel-desc"));
			chan.setTitle(config.getInitParameter("channel-title"));
			chan.setLink(config.getInitParameter("channel-link"));
		} catch (Exception e) {
			e.printStackTrace();
			throw new UnavailableException(
					"Required config parameters not found", -1);
		}
		this.setChannel(chan);
		System.out.println(getServiceName());
		System.out.println(getServiceURL());
		super.init(config);
	}
	public void destroy() {
		setChannel(null);
	}
	public void processAction(ActionRequest req, ActionResponse res)
			throws PortletException {
		System.out.println(getServiceName());
		System.out.println(getServiceURL());
		PortletSession session = req.getPortletSession(true);
		String user = req.getRemoteUser();
		list = (ArrayList) session.getAttribute("LinkList",
				PortletSession.APPLICATION_SCOPE);
		String action = req.getParameter("ACTION");
		String itemParam = req.getParameter("ITEM_ID");
		boolean changed = false;
		int itemId = -1;
		if (itemParam != null) {
			itemId = Integer.parseInt(itemParam);
		}
		if (action.equals("DELETE")) {
			list.remove(itemId);
			changed = true;
		} else if (action.equals("EDIT")) {
```

```
                Item item = (Item) list.get(itemId);
                item.setLink(req.getParameter("LINK"));
                item.setDescription(req.getParameter("DESCRIPTION"));
                item.setTitle(req.getParameter("TITLE"));
                changed = true;
        } else if (action.equals("NEW")) {
                String desc = req.getParameter("DESCRIPTION");
                String link = req.getParameter("LINK");
                String title = req.getParameter("TITLE");
                if (desc == null || link == null || title == null) return;
                Item item = new Item();
                item.setDescription(desc);
                item.setTitle(title);
                item.setLink(link);
                if (list == null) list = new ArrayList<Item>();
                list.add(item);
                changed = true;
        }
        session .setAttribute("LinkList", list,PortletSession.APPLICATION_SCOPE);
        if (changed) storeContentToContentService(list, user);
}
public void doView(RenderRequest req, RenderResponse res)
                throws PortletException, IOException {
        res.setContentType("text/html");
        if (list == null) {
                list = fetchContentFromContentService(req.getRemoteUser());
        }
        PortletSession session = req.getPortletSession(true);
        session .setAttribute("LinkList", list,
                                        PortletSession.APPLICATION_SCOPE);
        PortletContext context = getPortletContext();
        PortletRequestDispatcher dispatcher = context
                        .getRequestDispatcher(this.jspPath + "/view.jsp");
        dispatcher.include(req, res);
}
public void doEdit(RenderRequest req, RenderResponse res)
                throws PortletException, IOException {
        res.setContentType("text/html");
        String display = req.getParameter("DISPLAY");
        PortletContext context = getPortletContext();
        System.out.println(this.channel.getTitle());
        if ("EDIT_PAGE".equals(display)) {
                PortletRequestDispatcher dispatcher = context
                                .getRequestDispatcher(this.jspPath + "/editItem.jsp");
                dispatcher.include(req, res);
        } else {
                PortletRequestDispatcher dispatcher = context
                                .getRequestDispatcher(this.jspPath + "/editPage.jsp");
                dispatcher.include(req, res);
        }
}
public void doHelp(RenderRequest req, RenderResponse res)
                throws PortletException, IOException {
        res.setContentType("text/html");
        PortletContext context = getPortletContext();
        PortletRequestDispatcher dispatcher = context
                        .getRequestDispatcher(this.jspPath + "/help.jsp");
        dispatcher.include(req, res);
}
public ArrayList<Item> fetchContentFromContentService(String user) {
        ArrayList<Item> list = null;
        ContentServiceLocator locator = new ContentServiceLocator();
        try {
                locator.setEndpointAddress(getServiceName(), getServiceURL());
                SOAPport port = locator.getContentService();
                RSSMessageType rss = port.getContent(user);
                try {
                        Channel retchan = rss.getChannel();
                        this.channel = retchan;
                        list = new ArrayList<Item>(Arrays.asList(retchan.getItem()));
                        System.out.println("LIST HAS ITEMS: " + list.size());
                } catch (NullPointerException n) {
                        list = new ArrayList<Item>();
                }
        } catch (ServiceException e) {
          System.err.println("Error - service not found or service unavailable.");
                e.printStackTrace();
```

```
                } catch (RemoteException e) {
                        System.err.println("Error - service error, is it running?");
                        e.printStackTrace();
                }
                return list;
        }
        public void storeContentToContentService(ArrayList<Item> list, String user) {
                ContentServiceLocator locator = new ContentServiceLocator();
                Channel chan = this.getChannel();
                if (!list.isEmpty()) {
                        Item[] items = new Item[list.size()];
                        list.toArray(items);
                        chan.setItem(items);
                } else  chan.setItem(null);
                RSSMessageType rss = new RSSMessageType(chan, user);
                this.setChannel(chan);
                try {
                        locator.setEndpointAddress(getServiceName(), getServiceURL());
                        SOAPport port = locator.getContentService();
                        System.out.println("Status - " + port.setContent(rss));
                } catch (ServiceException e) {
                        System.err.println("Error - service not found");
                        e.printStackTrace();
                } catch (RemoteException e) {
                        System.err.println("Error - service error, is it running?");
                        e.printStackTrace();
                }
        }
        public Channel getChannel() { return channel;}
        public void setChannel(Channel channel) {this.channel = channel;}
        public String getServiceName() {return serviceName;}
        public void setServiceName(String serviceName) {this.serviceName = serviceName;}
        public String getServiceURL() {return serviceURL;}
        public void setServiceURL(String serviceURL) {this.serviceURL = serviceURL;}
}
```

C:\esimerkkisovellus\components\contentportlet\src\main\webapp\META-INF\context.xml:

```
<Context>
<Resource name="jcr/repository"
          auth="Container"
          type="javax.jcr.Repository"
          factory="org.apache.jackrabbit.core.jndi.BindableRepositoryFactory"
          configFilePath="/repository/repository.xml"
          repHomeDir="/repository/repo"/>
</Context>
```

C:\esimerkkisovellus\components\contentportlet\src\main\webapp\WEB-INF\web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
                        "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
        <display-name>ContentPortlet</display-name>
        <description>Content Portlet for demonstration purposes.</description>
        <taglib>
                <taglib-uri>http://java.sun.com/portlet</taglib-uri>
                <taglib-location>/WEB-INF/tld/portlet.tld</taglib-location>
        </taglib>
</web-app>
```

C:\esimerkkisovellus\components\contentportlet\src\main\webapp\WEB-INF\portlet.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app
        xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
        version="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd
                        http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd">
   <portlet>
                <description>Content Portlet used for personal links </description>
                <portlet-name>contentportlet</portlet-name>
                <display-name>Personal links</display-name>
        <portlet-class>fi.joensuu.cs.jhirvone.contentservice.ContentPortlet </portlet-class>
                <init-param>
                        <name>service-name</name>
```

```
                <value>ContentService</value>
            </init-param>
            <init-param>
                <name>service-url</name>
                <value>http://localhost:8192/loggerservice/</value>
            </init-param>
            <init-param>
                <name>channel-title</name>
                <value>myChannel</value>
            </init-param>
            <init-param>
                <name>channel-desc</name>
                <value>Links for my channel</value>
            </init-param>
            <init-param>
                <name>channel-link</name>
                <value>http://cs.joensuu.fi/</value>
            </init-param>
            <init-param>
                <name>jsp-path</name>
                <value>/jsp</value>
            </init-param>
            <expiration-cache>-1</expiration-cache>
            <supports>
                <mime-type>text/html</mime-type>
                <portlet-mode>VIEW</portlet-mode>
                <portlet-mode>EDIT</portlet-mode>
                <portlet-mode>HELP</portlet-mode>
            </supports>
            <supported-locale>en</supported-locale>
            <resource-bundle>ContentPortlet</resource-bundle>
            <portlet-info>
                <title>PersonalLinks</title>
                <short-title>Personal links</short-title>
                <keywords>RSS Content Links</keywords>
            </portlet-info>
        </portlet>
</portlet-app>
```

C:\esimerkkisovellus\components\contentportlet\src\main\webapp\jsp\editItem.jsp:

```
<%@ page language="java"%>
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet"%>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<c:set var="item" value="${LinkList[param.ITEM_ID]}" />
<a href="<portlet:renderURL portletMode="edit"/>">Back to Edit AllItems</a>
<form   action="<portlet:actionURL>
        <portlet:param name="ACTION" value="EDIT"/>
        <portlet:param name="ITEM_ID" value="<%=request.getParameter("ITEM_ID")%>"/>
</portlet:actionURL>"  method="post">
<table>
        <tr class="portlet-section-header"><td>Edit item:</td></tr>
        <tr><td>Title:</td><td><input type="text" size="45" name="TITLE" value="<c:out
                value="${item.title}"/>"></td></tr>
        <tr><td>Link:</td><td><input type="text" size="45" name="LINK"
                        value="<c:out value="${item.link}"/>"></td></tr>
        <tr><td>Description:</td><td><input type="text" size="45" name="DESCRIPTION"
                        value="<c:out value="${item.description}"/>"></td></tr>
        <tr><td><input type="submit" value="Edit"></td></tr>
</table>
</form>
```

C:\esimerkkisovellus\components\contentportlet\src\main\webapp\jsp\editPage.jsp:

```
<%@ page language="java"%>
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet"%>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@ page import="javax.servlet.jsp.jstl.core.LoopTagStatus"%>
<table>
        <tr><td>Edit my links:</td></tr>
        <c:forEach var="item" items="${LinkList}" varStatus="status">
                <%
                  LoopTagStatus status = (LoopTagStatus) pageContext .getAttribute("status");
                                String itemId = Integer.toString(status.getCount() - 1);
                %>
                <tr><td><a href="<c:out value="${item.link}"/>
```

```
                                        "><c:out
                                value="${item.title}" /></a></td>
                        <td><c:out value="${item.description}" /></td>
                        <td><a  href="<portlet:renderURL>
                                        <portlet:param name="DISPLAY" value="EDIT_PAGE"/>
                                        <portlet:param name="ITEM_ID" value="<%=itemId%>"/>
                                      </portlet:renderURL>">Edit
                        </a></td>
                        <td><a href="<portlet:actionURL>
                                        <portlet:param name="ACTION" value="DELETE"/>
                                        <portlet:param name="ITEM_ID" value="<%=itemId%>"/>
                                      </portlet:actionURL>">Delete
                        </a></td></tr>
        </c:forEach>
</table>
<form  action="<portlet:actionURL><portlet:param name="ACTION" value="NEW"/>
                </portlet:actionURL>" method="post">
<table>
  <tr class="portlet-section-header"><td>Add new link:</td></tr>
  <tr><td>Title:</td><td><input type="text" size="45" name="TITLE"></td></tr>
  <tr><td>Link:</td><td><input type="text" size="45" name="LINK"></td></tr>
  <tr><td>Description:</td><td><input type="text" size="45" name="DESCRIPTION"></td></tr>
  <tr><td><input type="submit" value="Create new"></td></tr>
</table>
</form>
```

C:\esimerkkisovellus\components\contentportlet\src\main\webapp\jsp\help.jsp:

```
<h2>Help for LinkList-portlet</h2>
This portlet can be used for personal links. Links are stored in content
store via WebServices interface.
<p>Use Edit-mode to add, delete or edit links.
```

C:\esimerkkisovellus\components\contentportlet\src\main\webapp\jsp\view.jsp:

```
<%@ page language="java"%>
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet"%>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<table>
        <tr><td>My links:</td></tr>
        <c:forEach var="item" items="${LinkList}" varStatus="status">
                <tr><td><a href="<c:out value="${item.link}"/>"> <c:out
                                value="${item.title}" /> </a></td>
                        <td><c:out value="${item.description}" /></td></tr>
        </c:forEach>
</table>
```

C:\esimerkkisovellus\components\contentportlet\pom.xml:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                http://maven.apache.org/maven-v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>fi.joensuu.cs.jhirvone.contentservice</groupId>
        <artifactId>contentportlet</artifactId>
        <packaging>war</packaging>
        <version>1.0-SNAPSHOT</version>
        <name>contentportlet</name>
        <url>http://maven.apache.org</url>
        <properties>
                <pluto.version>1.1.4</pluto.version>
                <axis.version>1.4</axis.version>
                <portlet-api.version>1.0</portlet-api.version>
                <servlet-api.version>2.3</servlet-api.version>
                <jsp-api.version>2.0</jsp-api.version>
        </properties>
        <dependencies>
                <dependency>
                        <groupId>portlet-api</groupId>
                        <artifactId>portlet-api</artifactId>
                        <version>1.0</version>
                        <scope>provided</scope>
                </dependency>
                <dependency>
                        <groupId>jstl</groupId>
```

```
                <artifactId>jstl</artifactId>
                <version>1.0.2</version>
        </dependency>
        <dependency>
                <groupId>taglibs</groupId>
                <artifactId>standard</artifactId>
                <version>1.0.4</version>
        </dependency>
        <dependency>
                <groupId>axis</groupId>
                <artifactId>axis</artifactId>
                <version>${axis.version}</version>
        </dependency>
        <dependency>
                <groupId>axis</groupId>
                <artifactId>axis-jaxrpc</artifactId>
                <version>${axis.version}</version>
        </dependency>
        <dependency>
                <groupId>axis</groupId>
                <artifactId>axis-wsdl4j</artifactId>
                <version>1.5.1</version>
        </dependency>
        <dependency>
                <groupId>axis</groupId>
                <artifactId>axis-saaj</artifactId>
                <version>${axis.version}</version>
        </dependency>
        <dependency>
                <groupId>axis</groupId>
                <artifactId>axis-ant</artifactId>
                <version>${axis.version}</version>
        </dependency>
        <dependency>
                <groupId>org.apache.pluto</groupId>
                <artifactId>pluto-util</artifactId>
                <version>${pluto.version}</version>
                <scope>provided</scope>
        </dependency>
</dependencies>
<build>
        <finalName>${pom.name}</finalName>
        <plugins>
                <plugin>
                  <artifactId>maven-war-plugin</artifactId>
                  <configuration>
                    <webXml>${project.build.directory}/pluto-resources/web.xml</webXml>
                  </configuration>
                </plugin>
                <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-compiler-plugin</artifactId>
                        <configuration>
                                <source>1.5</source>
                                <target>1.5</target>
                        </configuration>
                </plugin>
                <plugin>
                  <groupId>org.codehaus.mojo</groupId>
                  <artifactId>axistools-maven-plugin</artifactId>
                  <configuration>
                    <sourceDirectory>../contentservice/src/main/xml</sourceDirectory>
                    <mappings>
                        <mapping>
                        <namespace>http://cs.joensuu.fi/jhirvone/contentservice/service
                                </namespace>
                        <targetPackage>fi.joensuu.cs.jhirvone.contentservice.client
                        </targetPackage>
                        </mapping>
                    </mappings>
                <testCases>false</testCases>
                <serverSide>false</serverSide>
                <outputDirectory>src/main/java</outputDirectory>
                <subPackageByFileName>false</subPackageByFileName>
                </configuration>
                <executions>
                        <execution>
```

```
                                <phase>generate-sources</phase>
                                <goals> <goal>wsdl2java</goal> </goals>
                        </execution>
                </executions>
                </plugin>
                <plugin>
                        <groupId>org.apache.pluto</groupId>
                        <artifactId>maven-pluto-plugin</artifactId>
                        <version>${pluto.version}</version>
                        <executions>
                                <execution>
                                        <phase>generate-resources</phase>
                                        <goals> <goal>assemble</goal></goals>
                                </execution>
                        </executions>
                </plugin>
                <plugin>
                        <groupId>org.codehaus.mojo</groupId>
                        <artifactId>tomcat-maven-plugin</artifactId>
                        <configuration>
                                <url>http://localhost:8080/manager</url>
                                <server>pluto</server>
                        </configuration>
                </plugin>
        </plugins>
    </build>
</project>
```