XML ja monikanavajulkaiseminen

Hanna Tahvanainen

21.12.2005

Joensuun yliopisto
Tietojenkäsittelytiede
Pro gradu –tutkielma

TIIVISTELMÄ

Internetissä olevan tiedon ja palveluiden määrä on kasvanut viimeisen vuosikymmenen aikana räjähdysmäisen nopeasti. Palveluita halutaan käyttää paikasta ja tilanteesta riippumatta, mihin kellonaikaan tahansa. Monikanavajulkaiseminen on tuonut avun tähän ongelmaan mahdollistamalla saman sisällön muuntamisen eri muotoihin ja sen seurauksena materiaaliin voidaan päästä käsiksi eri päätelaitteiden kautta.

Tässä tutkielmassa perehdymme empiirisesti esimerkkisovelluksen avulla XML:n käyttöön monikanavajulkaisujärjestelmässä sekä tutustumme XSL-muunnoksen tekemiseen kun kohdetiedoston tyyppinä on HTML, WML, RTF ja PDF. Näissä muunnoksissa käytetään Xalanin XSLT-prosessoria ja Cocoon-julkaisukehystä.

Avainsanat: monikanavajulkaiseminen, XML, HTML, WML, RTF, PDF, XSLT, Cocoon

SISÄLLYSLUETTELO

1 JOHDANTO

Internet on saavuttanut suuren suosion räjähdysmäisen nopeasti. Yhä useampia palveluja on mahdollisuus käyttää verkossa ja ne ovat kaikkien ulottuvilla. Tämä on ainutlaatuista, koska ensimmäistä kertaa informaatiotekniikan historiassa on olemassa kommunikointitapa, joka antaa mahdollisuuden tavoittaa henkilön olipa hän missä tahansa. Tämän seurauksena Internetissä jaettavan informaation määrä on kasvanut nopeasti.

Nykypäivän kiireisessä yhteiskunnassa ihmisten ajankäyttö on muuttunut harkitsevammaksi. Vapaa-aikaa osataan arvostaa ja pakollisia asioita halutaan hoitaa silloin kuin itselle sopii eikä tiettyjen ennaltamääriteltyjen kellonaikojen mukaan.

Teknologian kehitys on tuonut avun tähän asiaan. Nykyään on yhä enemmän saatavilla sellaisia laitteita ja palveluja, jotka mahdollistavat asioiden hoitamisen muuallakin kuin paikan päällä. Markkinoilla olevien erilaisten päätelaitteiden määrä kasvaa jatkuvasti. Niiden avulla pyritään helpottamaan materiaalin jakamista ihmisille, riippumatta siitä missä he ovat. Internet-materiaalin tarjoajat pyrkivät tietenkin tavoittamaan myös ne ihmiset, jotka eivät halua istua kotona tai työpaikalla tietokoneen äärellä päästäkseen käsiksi haluamaansa informaatioon. Tämän takia langattomat päätelaitteet ovat kasvattaneet suosiotaan. Lähes jokaista erilaista päätelaitetta varten tarvitaan erimuotoista materiaalia ja monikanavajulkaisujärjestelmät ovat tulleet avuksi tähän ongelmaan. Näillä järjestelmillä käsitellään verkossa tarjottavaa materiaalia niin, että sitä voidaan katsoa minkä tahansa päätelaitteen kautta.

Monikanavajulkaisemisen perusteknologiaksi on vakiintumassa *XML* (eXtensible Markup Language), joka soveltuu tähän tarkoitukseen hyvin koska se mahdollistaa sisällön ja ulkoasun muokkaamisen erikseen. XML:n avulla sisällöntuottajat voivat pitäytyä ainoastaan päätehtävässään eli sisällön tuottamisessa, jättäen ulkonäölliset seikat muiden huoleksi. Sisällöntuottajat määrittävät XML:llä eri elementit (otsikot, kuvat jne.) [8], joita käsitellään sitten jokaisessa kanavassa sille soveltuvalla tavalla. Kanavan ulkonäkömääritelmät sisällölle voidaan vastaavasti määritellä XSL:n avulla ja näin automatisoida koko monikanavajulkaisuprosessi huomattavan suurelta osin.

Tutkielmassa perehdymme esimerkkisovelluksen avulla XML:n käyttöön monikanavajulkaisujärjestelmässä sekä tutustumme XSL-muunnoksen tekemiseen kun kohdetiedoston tyyppinä on HTML, WML, RTF ja PDF. Luvussa 2 tarkastelemme monikanavaisuuteen liittyviä asioita kuten kanavatukea, erilaisia päätelaitteita ja teknologioita, skaalautuvuutta, hallintaa ja personointia. Tämän lisäksi käymme läpi monikanavaisuuden heikkouksia ja vahvuuksia. Luvussa 3 käymme läpi XML-teknologioita eli XML, XSL, DOM ja SAX. XSL:ään liittyen käymme läpi myös XSL-muunnoksen ja –muotoilun sekä ilmaisukieli XPath:n ja XSL- ja XSLT-prosessorit. Luvussa 4 käymme läpi monikanavajulkaisujärjestelmän toteuttamisen kahdella eri tavalla: Xalan-Javan sekä Cocoonin avulla. Toteutukset on havainnollistettu esimerkkisovelluksen avulla. Luvussa 5 on tutkielman yhteenveto.

2 MONIKANAVAJULKAISEMINEN

Tässä luvussa tutustutaan monikanavaisuuteen, sen heikkouksiin ja vahvuuksiin sekä esitellään erilaisia päätelaitteita ja teknologioita. Luvussa käydään myös läpi monikanavaisuuteen kiinteästi liittyviä asioita: skaalautuvuutta, hallintaa ja personointia.

2.1 S.W.O.T.-analyysi

Korpiaho et al. [2] analysoivat monikanavapalveluiden vahvuuksia (Strengths), heikkouksia (Weaknesses), mahdollisuuksia (Opportunities) ja uhkia (Threats) kuvan 1 esittämällä tavalla.

Suurin *vahvuus* monikanavaisia palveluita tarjoavalle yritykselle on mahdollisuus tavoittaa palvelun kohderyhmä sen fyysisestä sijainnista riippumatta. Tämä mahdollistaa sen että asiakas pääsee käyttämään palvelua toistuvasti, silloin kun hänelle itselleen sopii. Personoinnin avulla voidaan käyttäjä saada palaamaan palveluun uudelleen, kun palvelun käyttöliittymä voidaan muokata käyttäjälle sopivaan muotoon esim. päätelaitteen mukaan.

Suurin *heikkous* monikanavaisissa järjestelmissä liittyy WAP-päätelaitteiden teknisiin rajoituksiin sekä päätelaitteiden ja palvelimen väliseen hitaaseen yhteyteen. Käyttäjän näkökulmasta myös palveluiden kustannukset ovat kohtalaisen kalliita. Palveluntarjoajan näkökulmasta langattoman ”stand-alone”-palvelun toteuttaminen on melko suuri investointi ja käyttäjien tarpeiden kerääminen eli näkökulman ymmärtäminen on todellinen haaste.

Monikanavajärjestelmäpalveluja tarjoava yritys on pioneeri mobiilipalveluissa ja saa siitä arvokasta kokemusta. Tämän kokemuksen avulla yrityksellä on *mahdollisuus* saavuttaa markkinajohtajan asema monikanavajärjestelmäpalveluiden tarjoamisessa. Kehittäjien näkökulmasta monikanavajärjestelmät antavat vapauden keskittyä varsinaisen sisällön tuottamiseen, sillä järjestelmä huolehtii sisällön muokkaamisesta kanavaan sopivaksi. Tämä nopeuttaa myös palveluiden muokkaamista: palvelu voidaan helposti siirtää uudelle päätelaitteelle.

Epäonnistuminen monikanava- ja varsinkin langattomien järjestelmien markkinoinnissa on aiheuttanut sen, ettei ihmisillä ole kovin selkeää kuvaa näistä palveluista. Mikäli järjestelmät

eivät toimi asiakkaan näkökulmasta kuten on mainostettu, markkinasijoitukset saattavat mennä hukkaan. Palveluntarjoaja ja laskuttaja vaikuttavat markkinoilla olevien langattomien palveluiden kilpailukykyyn. Esimerkikkinä tästä ovat suljetut portaalit, joista pääsee vain palveluntarjoajan omiin palveluihin estäen käyttäjien pääsyn muiden palveluntarjoajien palveluihin. Mikäli nämä suljetut portaalit yleistyvät, langattomien palveluiden tarjoaminen tulee olemaan monimutkaisempaa ja kalliimpaa.

Vahvuudet • paikkariippumattomuus • aikariippumattomuus • monipuolisempia ominaisuuksia personoinnin kautta	**Heikkoudet** • päätelaitteiden tekniset rajoitukset • servereiden ja päätelaitteiden väliset hitaat yhteydet • kalliit palvelut
Mahdollisuudet • kokemus • markkinajohtajuus/etulyöntiasema • kehittyvän teknologian mukanaantuomat palveluvalikoimat • paikannustiedon mukanaantuoma lisäarvo	**Uhat** • huono maine • riippuvuus palveluntarjoajan laskutuksesta • teknologia ei yleisty

Kuva 1. Matriisi S.W.O.T-analyysista [2].

2.2 Kanavatuki

Tässä kohdassa tarkastelemme kanavatukeen liittyviä asioita, sen monipuolisuutta ja -ulotteisuutta. Kanavalla voidaan tarkoittaa montaa eri asiaa, mutta rajatakseni aluetta hieman olen kohdistanut kanaviin liittyvän tarkasteluni kuitenkin vain päätelaitteisiin.

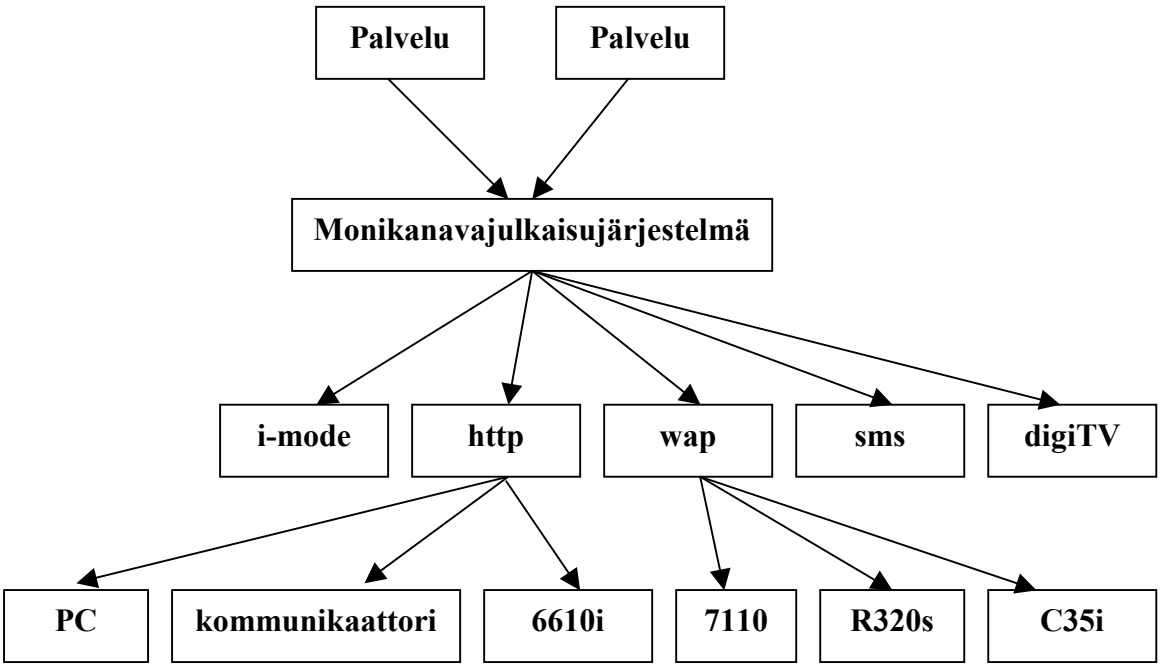
Monikanavajulkaisujärjestelmän kanavatuella tarkoitetaan sitä, minkä tyyppisten erilaisten kanavien kautta järjestelmä voi informaatiota julkaista. Nykyaikana, kun uusia päätelaitteita kehitellään markkinoille tasaiseen tahtiin, täytyy monikanavajulkaisujärjestelmän pystyä mukautumaan kanavatuen monipuolistumiseen. Kanavatuen monipuolisuudella tarkoitetaan sitä, että otetaan huomioon erityyppiset kanavat eikä pitäydytä vain esim. matkapuhelimien mahdollistamissa SMS- ja WAP- kanavissa, vaan huomioidaan myös esim. digitaalitelevisiot.

Uuden kanavatyypin lisäämisen täytyy olla yksinkertaista ja nopeaa, jotta käyttäjä pääsee hyödyntämään järjestelmän tuomia mahdollisuuksia niin helposti kuin mahdollista.

2.2.1 Vaiheistettu monikanavajulkaisu

Koska erilaisia päätelaitevaihtoehtoja on useita, ei monikavajulkaiseminen sellaisenaan ole täydellinen ratkaisu [7]. Sisällön tuottamisessa voidaan käyttää vain yhtä kriteeriä, vaikka esimerkiksi eri matkapuhelinvalmistajien WAP-selaimissa on suuriakin eroja mm. erikokoisten näyttöjen vuoksi. Tähän ongelmaan on ratkaisuna ns. vaiheistettu monikanavaisuus, jossa kanavat jakaantuvat alikanaviin muodostaen useampia tasoja valintapuussa. Jokainen näistä tasoista valitaan yhdellä kriteerillä, esimerkiksi tuotetun sisällön, käyttäjätietojen tai käytettävän päätelaitteen perusteella. Tällä hienojakoisemmalla tavalla mahdollistetaan siis useampien kriteerien huomioonottaminen sisällön tuottamisessa ja julkaisussa. Näin voidaan esimerkiksi määritellä omat WAP-ulkoasunsa Nokia 7110:lle ja huomattavasti pieninäyttöisemmälle Nokia 5210:lle.

Yksittäisten päätelaitteiden tukemisen lisäksi vaiheistetun monikanavaisuuden avulla voidaan toteuttaa myös mm. palveluiden brändäämistä ja massapersonointia. Näin voidaan paitsi tukea käyttäjää päätetasolle asti ja näin parantaa käyttökokemusta, myös lisätä erilaisten liiketoimintamallien määrää mm. markkinoiden segmentoinnin ja differoinnin kautta. Kuvassa 2 on havainnollistettu jaottelua monikanavaisuudelle.



Kuva 2. Vaiheistettu monikanavaisuus.

2.3 Päätelaitteet ja teknologiat

Tässä kohdassa käydään läpi monikanavajärjestelmiin liittyviä päätelaitteita ja teknologioita. Tarkastelu perustuu kuvan 2 jaotteluun.

2.3.1 SMS ja MMS

GSM (Global System for Mobile Communications)-verkoissa, mobiilit palvelut perustuvat pääasiallisesti *WAP*:iin (Wireless Application Protocol) tai *SMS*:ään (Short Message Service) [2]. SMS on erittäin suosittu ja sen teknologian ympärille kehitellään koko ajan uusia palvelumuotoja rajoituksista (vain 160 merkkiä pitkä tekstimuotoinen viesti) huolimatta. Sen vahvuutena on yksinkertaisuus; se on helppo käyttää ja lähes kaikki päätelaitteet tukevat sitä.

MMS (Multimedia Messaging Service) on SMS:n seuraavan sukupolven versio. Tekstin lisäksi MMS-viestit voivat sisältää videokuvaa, värikuvia ja ääntä. Aikaisemmin MMS-viestien ajateltiin olevan liian raskaita nykyisiin mobiiliverkkoihin ja päätelaitteisiin, mutta laitteiden nopea kehitys ja kuluttajien tarpeet ovat mahdollistaneet palvelumuodon.

2.3.2 WAP ja GPRS

WAP-palvelut syntyivät tarpeesta luoda kommunikointimenetelmä langattomaan maailmaan, joka ymmärtää päätelaitteiden ja kaistanleveyden synnyttämät rajoitukset. WAPin tehtävänä on määritellä yhteys reaaliaikaiseen tietoon langattomissa päätelaitteissa erilaisten verkkoratkaisujen yli. Nykyään kaikki suurimmat matkapuhelinvalmistajat valmistavat WAP:ia tukevia päätelaitteita ja tämä osaltaan aiheuttaa ongelmia sillä ratkaisumalleja on niin monenlaisia. Monien ratkaisumallien takia puhtaasti päätelaitteesta riippumattoman palvelun kehittäminen ei ole mahdollista. Suurimmat WAP:ia tukevien päätelaitteiden rajoituksista ovat pieni näyttö, ahdas näppäimistö ja grafiikan tukeminen (mm. värien näyttäminen).

GPRS (General Packet Radio Service) on GSM-verkossa toimiva pakettikytkentäinen tiedonsiirtopalvelu, jota käytetään pääasiassa langattoman Internet-yhteyden muodostamiseen matkapuhelimen tai GPRS-sovittimen avulla. GPRS:n yleistymisen myötä WAP-päätelaitteet ovat saaneet useita parannuksia. Yksi niistä on pakettiperusteinen laskutus eli käyttäjä ei maksa tiedon siirtämiseen kuluvasta ajasta vaan siirtämästään tiedon määrästä.

2.3.3 i-mode

WAPin suurin haastaja on tähän mennessä ollut NTT DoCoMon i-mode. I-modet ovat toimineet *CDMA* (Code Division Multiple Access) -verkoissa ja tukeneet pakettiperustaista tiedonsiirtoa, mitä WAP on alkanut tukemaan vasta GPRS:n myötä. I-mode on myös tukenut alusta lähtien värinäyttöjä ja grafiikkaa mikä on edesauttanut viihdetarkoitukseen tehtyjen palveluiden menestystä. I-mode perustuu kompaktiin HTML:ään (cHTML), joka on HTML:n kevennetty versio langattomille laitteille. cHTML on tarjonnut kehittäjille tutun syntaksin ja näin nopeuttanut palveluiden kehittämistä.

2.3.4 J2ME/MIDP

J2ME (Java 2 Micro Edition) [22] on pienlaitteille ja sulautetuille järjestelmille tarkoitettu ohjelmistojen kehitys- ja ajoympäristö. J2ME:ssä on erilaisille laitteille tarkoitettuja konfiguraatioita. Konfiguraatio koostuu virtuaalikoneesta, kirjastoista, luokista ja API-rajapinnasta. Tällä hetkellä on olemassa kaksi konfiguraatiota: CLDC (Connected Limited Device Configuration) ja CDC (Connected Device Configuration).

Konfiguraatioiden päällä on vielä erilaisiin käyttökohteisiin tarkoitettuja profiileja. Paljon käytetty profiili on MIDP (Mobile Information Device Profile) [23], joka määrittää mobiililaitteiden ohjelmistojen avoimen sovelluskehitysympäristön. Sovelluskehitysympäristö muodostuu sovellusarkkitehtuurista ja API-rajapinnoista. MIDP määrittää käyttöliittymäkomponentit, syöttötietojen ja tapahtumien käsittelyn, verkkoyhteyksien käsittelyn ja aikaan liittyviä toimintoja ottaen huomioon mobiilien laitteiden rajallisen prosessitehon ja muistin määrän.

2.3.5 Kämmentietokoneet

Kämmentietokoneet on kehitetty muistikirjamikrojen (notebook computer) pohjalta. Matkapuhelimiin verrattuna kämmentietokoneissa on isompi näyttö joka pystyy näyttämään kehittynyttä grafiikkaa. Uusimmat kämmentietokoneet sisältävät www-selaimen joka tukee HTML:ää mikä tekee erillisestä palvelusta, esim. WAP:sta, turhan. Kämmentietokoneissa ongelmia saattavat aiheuttaa kosketusnäytön ja stylus-kynän käyttö verrattuna hiireen sekä näytön pienuus. Yleensä internet-sovelluksissa oletetaan päätelaitteen näytön kooksi 15 tuumaa, joten kämmentietokoneella palvelua käytettäessä palvelun sivua ei pystytä näyttämään kokonaan.

2.3.6 Digitaalitelevisiot

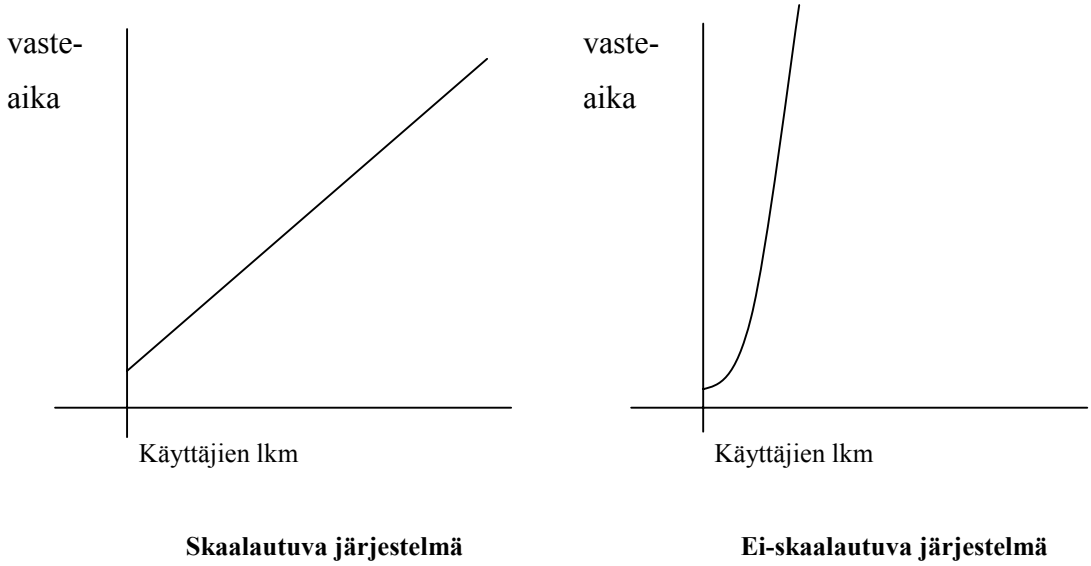
Digitaalitelevisiota voidaan pitää 1950-luvulla markkinoille tulleen väri-tv:n jälkeen merkittävimpänä yksittäisenä uudistuksena televisioalalla. Sillä voidaan näyttää selkeämpää ja selvästi terävämpää kuvaa sekä tarjota monikanavaista ääntä, joka on lähes cd-levyn tasoa. Se mahdollistaa aidosti vuorovaikutteisen palvelun, joka tarjoaa television kautta pääsyn Internetiin ja ohjelmien tilauksen katsojalle sopivaan aikaan. Siirryttäessä digitaalitekniikkaan on parannuksia luvassa vanhaan analogiseen järjestelmään verrattuna. Tärkeimpinä syinä siirtymiselle voidaan mainita muun muassa kuinka paljon tietoa voidaan lähettää, kuinka yhtenäisenä lähetettävä tieto pysyy edetessään ja minkä tyyppistä tietoa lähetettävä signaali voi sisältää. Samaan kaistanleveyteen saadaan mahtumaan paljon enemmän tietoa kuin vanhassa analogisessa järjestelmässä. Digitaalisen signaalin kanssa ei tule läheskään niin paljoa häiriöitä syrjäisessä paikassa sijaitsevan television kuvaan. Niin kauan kun TV vain pystyy ottamaan signaalia vastaan, on kuva digitaalitelevisiossa sama kuin sitä lähetettäessä. Digitaalitelevisio tukee myös datan lähettämistä kuvan ja äänen ohessa. Tämä tarjoaa mahdollisuuden lähettää

kanavan omia tiedotteita (esim. tietoa ohjelmamuutoksista), ohjelmistopäivityksiä televisioon, lisätietoa mainoksissa olevista tuotteista tai mitä tahansa muuta aineistoa samaan aikaan normaali TV-lähetyksen kanssa. Se on myös suunniteltu hyvin joustavaksi tukemaan uudenlaisia palveluja, joille mahdollisesti käyttäjien kiinnostuksen ja kehittyvän tekniikan myötä tulee tarvetta. Tällainen uusi palvelu voisi olla esimerkiksi huomattavasti paremman ja yli neljä kertaa tarkemman kuvanlaadun tajoava HDTV (High Definition TV) [5]. Rohkeimmissa ennusteissa arvoidaan, että digitaalitelevision avulla saadaan Internet joka kotiin ja näin tiedon valtaväylä olisi kohtuullisin kustannuksin jokaisen suomalaisen käytettävissä [4]. Pelkästään jo uusi televisio avaa suuret markkinat eri laitevalmistajille, koska kaikkien on joka tapauksessa digitaaliseen televisioon ajan kuluessa päivitettävä. Mutta jos ennuste TV:n käyttämisestä suuremmissa määrin myös Internet palveluihin pitää paikkaansa, avaisi se jättimarkkinat niin laite-, ohjelmisto- kuin palvelutaloillekin.

2.4 Skaalautuvuus

Internetin suosion kasvaessa verkossa toimivien järjestelmien tulee olla varautuneita suuriinkin käyttäjämääriin. Käyttäjämäärien kasvu saattaa olla nopeaa ja sen takia suoritustehon lisäämisen tulee olla mahdollisimman helppoa. Skaalautuvuudella tarkoitetaan sitä, kuinka hyvin järjestelmä pystyy suoritutumaan tehtävistään käyttäjämäärien kasvaessa. Alla olevat kuvan 3 käyrät havainnollistavat käyttäjien lukumäärän vaikutusta vasteaikaan skaalautuvassa ja ei-skaalautuvassa järjestelmässä. Vasteajalla tarkoitetaan jonkin tehtävän suoritukseen kuluvaa aikaa.

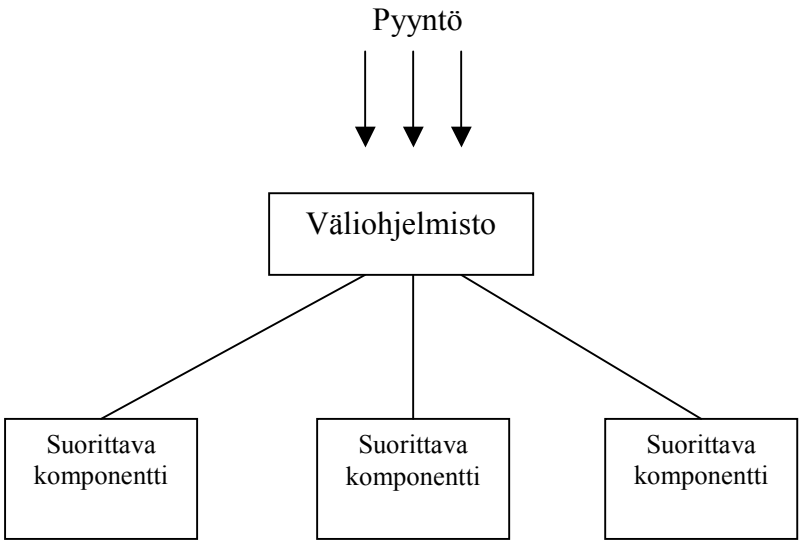
Skaalautuvassa järjestelmässä vasteajat ja virhealttius säilyvät suhteessa samana käyttäjien lukumäärän kasvaessa. Mikäli järjestelmä ei ole skaalautuva, sen tehtävien suorittamiseen kuluttama aika ja virhealttius kasvavat nopeasti.



Kuva 3. Käyttäjien lukumäärän vaikutus skaalautuvan ja ei-skaalautuvan järjestelmän vasteaikaan.

Järjestelmän skaalautuvuus pitää ottaa huomioon jo arkkitehtuuria suunniteltaessa. Koska skaalautuva järjestelmä täytyy suunnitella alusta lähtien todella huolellisesti, siitä tulee automaattisesti varmempi, koska huolellinen suunnittelu ratkaisee yleensä myös muita ongelmakohtia. Ongelmia voivat aiheuttaa käyttäjien määrän kasvaessa mm. synkronointi eli tahdistus. Kun käyttäjämäärä lisääntyy, saattaa joidenkin tehtävien synkronoinnissa esiintyä ongelmia, joita ei pienemmillä käyttäjämäärillä esiintynyt. Liittymät ulkoisiin järjestelmiin voivat myös aiheuttaa ongelmia. Jos skaalautuva järjestelmä pystyy suoriutumaan suuren käyttäjämäärän lähettämästä informaatiosta ja lähettää sitä eteenpäin, saattaa ilmetä, että vastaanottava järjestelmä ei pystykään vastaanottamaan samaa määrää informaatiota samalla vauhdilla. Tähän on ratkaisuna se, että skaalautuvan järjestelmän päähän tehdään puskuri, joka antaa järjestelmän tuottaa informaatiota nopeasti tukkeuttamatta vastaanottavaa järjestelmää.

Eräs tapa ratkaista skaalautuvuus on hajauttaa järjestelmä. Silloin järjestelmän työkuormaa jaetaan useammalle suorittavalle komponentille. Hajautuksessa suoritustehoa voidaan käyttäjämäärien kasvaessa helposti nostaa lisäämällä suorittavia komponentteja.



Kuva 4. Hajautuksen periaate.

Edellä olevassa kuvassa on esitetty hajautuksen periaate. Siinä käyttäjä tai järjestelmä lähettää pyynnön, joka menee väliohjelmistolle. Väliohjelmisto on välitason ohjelma, joka yhdistää kaksi eri järjestelmää toisiinsa [6]. Tämän jälkeen väliohjelmisto ohjaa pyynnön edelleen jollekin suorittavalle komponentille, joita on hajautetussa järjestelmässä useita. Pyynnön palautus menee samaa reittiä.

Jos järjestelmästä halutaan tehdä skaalautuva, aiheuttaa se luonnollisesti ylimääräisiä kustannuksia, mm. testauksesta. Testauksen ajaksi järjestelmä joudutaan ottamaan pois varsinaisesta käytöstä, joka tietenkin tarkoittaa montaa menetettyä euroa. Jos skaalautumistapana käytetään hajautusta, kustannuksia tulee tietysti uusista laitteistoista. Skaalautuvuutta voidaan parantaa myös kompromissiperiaatteella. Tämä tarkoittaa sitä, että parannetaan jotain ominaisuutta jonkin toisen ominaisuuden kustannuksella. Esimerkiksi hidasta prosessoria voidaan kompensoida lisämuistilla.

2.5 Hallinta

Koska kansainvälistyminen on nykyään yksi avainsanoista, saattaa palveluntarjoajalla olla toimipisteitä ympäri maailmaa. Tällöin järjestelmän ylläpitotoimenpiteet pitäisi pystyä suorittamaan sijaintiriippumattomasti. Monikanavajulkaisujärjestelmien, kuten myös muidenkin järjestelmien, hallinta helpottuu, mikäli järjestelmä mahdollistaa etähallinnan. Etähallinnalla

tarkoitetaan sitä, että järjestelmälle voidaan tehdä ylläpitotoimenpiteitä muualtakin kuin itse palvelimelta. Tämä on tärkeää varsinkin silloin, kun järjestelmää käytetään eri paikoista ja etäisyys palvelimelle saattaa olla suuri.

Yleisin malli ympäri maailmaa sijoittuvissa järjestelmissä on se, että yhdessä paikassa on itse palvelin tai joukko palvelimia, joihin sitten muut koneet ottavat *TCP/IP*-yhteyden (Transmission Control Protocol/Internet Protocol) [6]. Tämän yhteyden avulla voidaan hoitaa myös ylläpitotoiminnot. TCP/IP on kuljetuskerroksesta (TCP) ja reitityspalvelusta (IP) koostuva yhteyskäytäntö ja siihen liittyvien palveluiden (FTP, Telnet, WWW ym.) joukko. Se on maailman eniten käytetty lähiverkkojen protokolla.

Etähallinnassa suoritettavat toimenpiteet liittyvät yleensä vain yleisimpiin ja yksinkertaisimpiin operaatioihin. Näitä ovat esimerkiksi järjestelmän toimintojen optimointi sekä tilankäytön tarkkailu ja valvonta. Raskaammat ja vaativammat operaatiot suoritetaan yleensä itse palvelimelta.

WWW-pohjaista hallintajärjestelmää käytetään nykyään kasvavassa määrin uusimmissa ja suurimmissa järjestelmissä. Suosioon vaikuttaa mallin hyvä puoli, eli järjestelmän tilan pystyy näkemään helposti. Hallintajärjestelmän toteutus riippuu toteutustavasta, mutta yleisimmin palvelimeen otetaan HTTP-yhteys, ja näin päästään näkemään järjestelmän tilaa kuvaavia arvoja WWW-sivun kautta. Huonona puolena WWW-pohjaisessa hallintajärjestelmässä on se, että jos jonkun asian *konfigurointi* eli asetusten muuttaminen epäonnistuu, on korjaukset tehtävä kuitenkin suoraan palvelimelle paikan päällä. Tämä takia sitä käytetään lähinnä staattisiin valvonta- ja seurantatoimenpiteisiin.

2.6 Personointi

Nykyaikana ihmisiä uhkaa informaatioähky. Tarjolla on runsaasti erilaista tietoa monen eri kanavan kautta. Kaikki tarjolla oleva tieto ei kiinnosta kaikkia ihmisiä ja jokaisella tulisikin olla mahdollisuus suodattaa tarjotusta informaatiosta vain ne asiat, jotka henkilökohtaisesti kiinnostavat. Eräs ratkaisu tähän ongelmaan on personointi. Personoinnilla tarkoitetaan sitä, että käyttäjät voivat osallistua palveluiden ulkoasun ja jossain tapauksissa myös sisällön muotoilemiseen. He voivat personoida oman aloitussivunsa tai päätelaitteella käyttämänsä palvelut haluaamansa muotoon. Vaikka personointi vaatiikin runsaasti aineistoa, henkilökohtaista

profiilia ja julkaisun jakoa komponentteihin, voidaan sillä lisätä käyttäjätyytyväisyyttä etenkin aktiivisten käyttäjien keskuudessa.

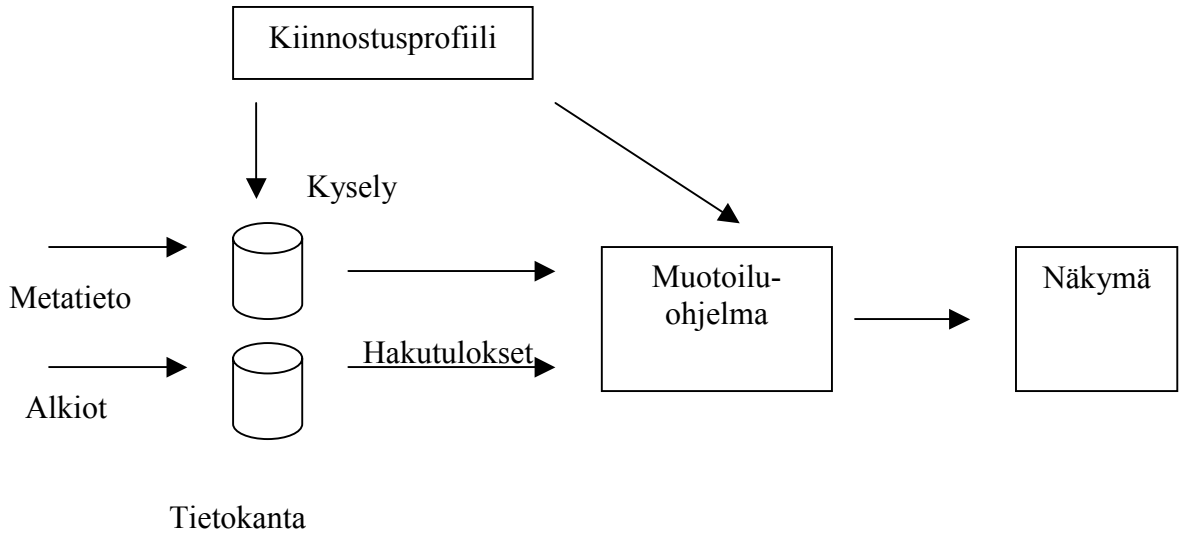
Personointi on suhteellisen uusi asia. Yhä enemmän erilaisissa kanavajulkaisuissa pyritään siihen, että jokaista julkaisun käyttäjää voidaan palvella henkilökohtaisemmin juuri personoinnin avulla. Personointi saattaa jossain määrin korvata verkkomaailmasta puuttuvan kasvokkain tapahtuvan asiakaspalvelun.

Sivustoille on ilmestynyt monia rekisteröintiä vaativia palveluita, joiden avulla yritys voi tarkkailla erilaisiin ihmisryhmiin kuuluvien asiakkaiden kiinnostusta omaan palveluunsa ja toisaalta kohdentaa tarjontaa saadun palautteen avulla. Palveluissa käy päivittäin lukuisiin erilaisiin käyttäjäprofiileihin sopivia asiakkaita. Heillä kaikilla on erilaiset lähtökohdat ja tavoitteet palvelun käyttämiselle. Täytetyn henkilötietolomakkeen ja sivustoilla tehtyjen liikkeiden avulla palvelua voidaan myös muokata juuri tälle asiakkaalle mieleiseksi. Asiakas saa personoitua palvelua ja yritys saa arvokasta tietoa, jota se voi käyttää omassa markkinointitoiminnassaan ja tuotteidensa kehittämisessä. Käyttäjä saa myös lisäarvoa personoinnista nopeutuneen tiedon löytymisen ja palvelun mielekkyyden lisääntymisenä.

Palveluiden omatoiminen personointi on palveluntarjoajan näkökulmasta helppo, mutta tieteellisesti ei kuitenkaan kovin mielenkiintoinen lähtökohta [9]. Käyttäjä voi itse määrätä käyttämänsä sivuston ulkoasun ja sisällön erilaisten ulkoasuun vaikuttavien valintojen avulla. Tällöin käyttäjä joutuu kuitenkin itse muokkaamaan käyttämäänsä palvelua. Kokeneemmat käyttäjät ovat tästä varmasti mielissään, mutta aloittelevalle käyttäjälle tällainen lähestymistapa ei välttämättä sovellu. Sovellus saattaa antaa mahdollisuuden vaikuttaa myös omassa profiilissa näkyvän tiedon sisältöön esimerkiksi valitsemalla, kuinka teknistä tekstiä näytetään. Tämä yhdessä ulkoasun muokkaamisen kanssa tehostaa varmasti erilaisten käyttäjien ja käyttötapausten mielekkyyttä samaa palvelua hyväksikäyttäen.

Käyttäjäprofiilien avulla personoituva palvelu perustuu usein käyttäjälle esitettyihin kysymyksiin. Paras tulos personoinnin kannalta saavutetaan yksilön kiinnostusten kohteiden laajalla ylöskirjaamisella. Tämä lähestymistapa edellyttää palvelun käyttäjältä suurta aktiivisuutta.

Toisaalta voidaan miettiä, kuinka pitkälle personoinnissa voidaan mennä. Liikaa personoitu palvelu ei anna virikkeitä uusien asioiden parissa, koska käyttäjä saa eteensä vain tiukasti omaan profiiliinsa sopivaa materiaalia. Alla olevassa kuvassa 5 on esitetty esimerkki personoinnin toimintaideasta.



Kuva 5. Personoinnin perusidea.

Kuvan esittämä toimintaidea on seuraavanlainen: kiinnostusprofiilia vastaavat esim. näkymän ulkoasuun liittyvät piirteet haetaan tietokannasta. Samoin haetaan myös haluttu informaatio. Muotoiluohjelmassa yhdistetään sekä informaatio että käyttäjän valitsemat henkilökohtaiset piirteet ja nämä kootaan näkymäksi, joka esitetään käyttäjälle.

3 XML-TEKNOLOGIAT

W3C (World Wide Web Consortium) [11] kehittää yhteisiä ja yhteensopivia WWW:n eli webin pelisääntöjä ja teknologioita. Keskeinen osa W3C:n työstä on erilaisten Webin kehittymistä ohjaavien teknisten spesifikaatioiden määrittely ja suositusten asettaminen. Eräs keskeisimmistä on modernin Webin perusarkkitehtuurin ja yhtenäisen kieliopin määrittelevä XML-suositus. Tässä luvussa käydään läpi eri XML-teknologioita ja niiden käyttötarkoituksia.

3.1 XML

The World Wide Web Consortium (W3C) perustettiin vuonna 1994 johtamaan webin yleisten protokollien kehittämistä webin kehittymisen edistämiseksi ja sen eri osien yhteensopivuuden takaamiseksi [11]. Protokollalla tarkoitetaan joukkoa yhteisesti sovittuja sääntöjä ja standardeja, joita käyttämällä kaksi laitetta voivat kommunikoida helpommin keskenään.

Eräs W3C:n aikaansaannoksista on XML (Extensible Markup Language), helmikuussa 1998 valmistunut kansainvälinen standardi. XML-standardin taustalla on kaksi aikaisemmin määriteltyä metakieltä: yleinen dokumenttirakenteiden määrittelyyn ja rakenteisten dokumenttien merkkaamiseen tarkoitettu *SGML* (Standard Generalized Markup Language), ja verkkodokumenttien esittämiseen tarkoitettu, SGML-kieleen perustuva *HTML* (Hypertext Markup Language).

HTML:n vahvuus on sen sopivuus verkkojakeluun ja dokumenttien esittämiseen verkossa. Internetin laajeneminen tuo verkkoon yhä monimutkaisempia ja suurempia dokumentteja, joiden monipuoliseen käsittelyyn HTML ei tarjoa riittäviä mahdollisuuksia. SGML puolestaan on kehitetty ennen nykyistä verkkotekniikkaa, joten siinä on niin paljon erilaisia ja harvoin käytettäviä ominaisuuksia, ettei yleisten SGML-sovellusten rakentaminen ole helppoa. XML:stä haluttiin verkkokäyttöön sopiva standardi tiedon esittämistä, käsittelyä ja siirtoa varten, joten siihen yhdistettiin ominaisuuksia sekä SGML:stä että HTML:stä.

3.1.1 XML:n vahvuudet ja heikkoudet

Eräänä XML:n hyvänä puolena voidaan pitää informaation vaihdon helpottumista erilaisten järjestelmien välillä. Kielen eräs tärkeimmistä komponenteista on *DTD* (Document Type Definition). Se on sivupohja, joka määrittelee, mitä elementtejä voidaan käyttää dokumentissa, mitkä elementit voivat sisältää muita elementtejä, elementtien lukumäärän ja järjestyksen, elementtien sisältämät attribuutit ja optimaalisessa tilanteessa myös ne arvot, jotka attribuutti saa sisältää. Jos on sovittu saman DTD:n käytöstä, antaa se tiettyyn aiheeseen liittyvälle tiedolle standardin muodon. Monet eri sovellukset omaavat vakiomuotoiset DTD:t [12]. Tämä tarkoittaa sitä, että järjestelmät voivat käyttää näitä yleisiä DTD:itä dokumentteja kirjoitettaessa eikä niiden tarvitse välittää sisäisestä muodosta. Tämä helpottaa järjestelmien välistä kommunikointia. W3C:n XML-skeema on kehitetty mahdollistamaan XML-dokumentille samantyyppisen oikeamuotoisuuden tarkistamisen, minkä myös DTD mahdollistaa [27]. XML-skeema kuitenkin käyttää XML:n kielioppia, ja mahdollistaa lisäksi nimiavaruuden käyttämisen sekä arvojoukkojen tarkemman määrittelyn. Tietyntyyppiset sovellukset, kuten esimerkiksi tietojen siirtäminen tietokantojen välillä ja sähköisen kaupankäynnin sovellukset, tulevat olemaan yksinkertaisia ja yhteensopivampia XML-skeeman ansiosta [28].

Toisena XML:n hyvänä puolena voidaan pitää dokumenteissa olevan tiedon haun helpottumista. Informaatio on XML-dokumenteissa kuvattu niin tarkasti elementtien avulla (kuva 6), että käyttäjä voi etsiä haluamaansa tietoa paljon tehokkaammin kuin perinteisten tekstihakujen avulla. Myös standardit DTD:t mahdollistavat paljon täsmällisempien hakujen tekemisen, tosin erilaisten DTD:iden suuren lukumäärän takia näiden hyödyntäminen globaaleissa hakukoneissa tulee olemaan paljon vaikeampaa.

```
<?xml version=”1.0” encoding=”ISO-8859-1”?>
<!DOCTYPE document SYSTEM “/tmp/example.dtd”>
<document date=”Aug 1, 2005” author=”Hanna Tahvanainen”>
<head>
<title>Esimerkki</title>
</head>
<body>
<header level=”1”>Esimerkki dokumentista</header>
<p>
          Tämä on esimerkki.
</p>
</body>
</document>
```

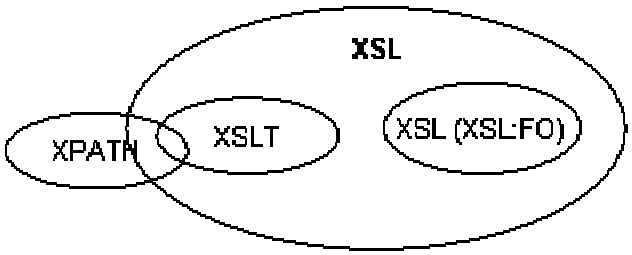
Kuva 6. XML-dokumentti.

Joustavuus on XML:n sekä huono että hyvä puoli. Toisaalta se antaa käyttäjälle vapauksia, koska hänen ei tarvitse pitäytyä ennaltamääritellyissä elementeissä vaan voi määritellä omat elementtinsä itse. Tämä voi kuitenkin johtaa myös joihinkin ongelmiin, sillä jokainen organisaatio tai teollisuusryhmä voi kehittää ja julkistaa oman ”standardinsa”, joka puolestaan saattaa aiheuttaa yhteensopivuusongelmia.

Eräänä XML:n huonona puolena voidaan pitää sen vaatimuksia tiedonsiirtokapasiteetille. Esimerkiksi binäärimuotoisen dokumentin muuntaminen XML-muotoon kasvattaa tiedoston kokoa huomattavasti, joten sen siirtäminen verkossa vaatii enemmän tiedonsiirtokapasiteettia.

3.2 XSL

XSL on tapa muuntaa XML-dokumentti toiseen formaattiin [14]. Sen avulla sisältöä voidaan muuntaa erilaisiin julkaisuympäristöihin sopivaksi kuten WWW-selaimiin, matkapuhelimiin tai tulostamista varten. XSL-standardi on jaettu kolmeen eri standardiin (kuva 7): *XSL-FO*:lla (XSL Formatting Objects) luodaan esitysulkoasu, *XSLT*:llä (XSL Transformations) suoritetaan sanaston muuntaminen ja *XPath*:lla (XML Path language) lisätään osoittimet alkuperäiseen XML-dokumenttiin.

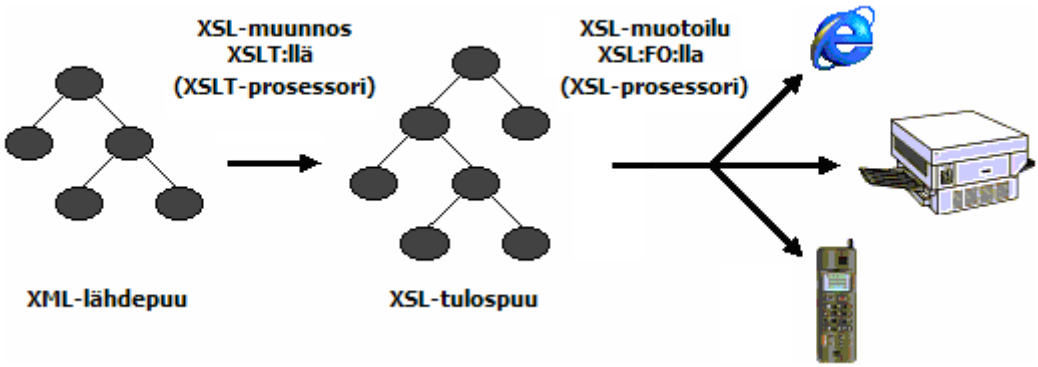


Kuva 7. XSL:n jakautuminen standardeihin.

XSL-prosessissa on pääpiirteittäin kaksi vaihetta:

- XSL-muunnoksella (XSLT) lähdedokumentti muunnetaan sopivaan muotoon.
- XSL-muotoilulla (XSL:FO) määritellään XML-dokumentin ulkoasu.

Kuva 8 havainnollistaa muunnoksen vaiheita. Lopputuloksena on halutunlainen muunnos ja muotoilu halutulle laitteelle.



Kuva 8. XSL-prosessin kaksi päävaihetta ovat muunnos ja muotoilu [13].

3.2.1 XSL-muunnos (XSLT)

XSLT on XML-syntaksia noudattava muunnoskieli [17]. Sen avulla XML-dokumenttien rakenteita voidaan muokata tai dokumentit voidaan muuntaa kokonaan toista rakennemäärittelyä noudattaviksi, esimerkiksi HTML:ksi. XSLT:tä voidaan käyttää myös muuntamaan XML-dokumentit täysin XML:stä eroavaa kieltä noudattavaksi (mm. PDF). Seuraava kuvan 9 esimerkki muuntaa XML-tiedoston osan HTML-muotoon ja määrittää kappaleen:

XML-tiedoston osa:

```
<kappale>
<korostettu>Lihavoitu</korostettu>
termi
</kappale>
```

HTML-kappale

```
<P><B>Lihavoitu</B> ter-
mi</P>
```

XSL-säännöstö:

```
<xsl:template match=”kappale”>
   <html:P>
      <xsl:apply-templates/>
   </html:P>
</xsl:template>
<xsl:template match=”korostettu”>
   <html:B>
      <xsl:apply-templates/>
   </html:B>
</xsl:template>
```

Kuva 9. Esimerkki XML-esityksen muuntamiseksi HTML-muotoon.

Nimiavaruudella määritellään XML-dokumentissa käytettävien elementtien standardityypit. Nimiavaruudet määritellään <xsl:stylesheet> -elementissä xmlns-attribuutissa seuraavasti:

```
<xsl:stylesheet
xmlns:xsl=”http://www.w3.org/1999/XSL/Transform”
xmlns:fo=”http://www.w3.org/1999/XSL/Format”>
```

Nimiavaruuksiin viitataan seuraavasti:

```
<xsl:template>…</xsl:template>
<fo:block>…</fo:block>
```

Oletusnimiavaruudessa viittausta ei suoriteta. Oletusnimiavaruus määritellään seuraavasti:

```
<xsl:stylesheet
xmlns=”http://www.w3.org/1999/XSL/Transform”>
```

Luvussa 4 käsitellään XSL-muunnokseen liittyviä asioita tarkemmin.

3.2.2 XSL-muotoilu (XSL:FO)

XSL on muotoilukieli, jolla voidaan kuvata XML-dokumentin ulkoasu [13]. XSL:n avulla määritellään fontit, marginaalit, listat, otsikoiden muodot, sivunumeroinnit, jne. XSL:llä voidaan tehdä myös monimutkaisempia ulkoasuun liittyviä asioita kuten alaviitteitä, sarakkeita tai taulukoita. XSL noudattaa XML-syntaksia sekä käyttää epäsuorasti hyödykseen CSS:ssä määriteltyjä ominaisuuksia. Seuraava esimerkki määrittelee säännön vahvennus-elementille muuntamalla elementin sisällön vahvennetuksi:

```
<xsl:template match=”vahvennus”>
  <fo:block font-weight=”bold”>
  <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

Tätä sääntöä voisi soveltaa seuraavaan XML-dokumentin osaan:

```
<vahvennus>Tämä on vahvennettu lause.</vahvennus>
```

3.2.3 Ilmaisukieli Xpath

XPath on kieli XML-dokumentin osien paikantamiseen ja valintaan [16]. Sen avulla voidaan paikantaa esimerkiksi kaikki tietyn tyyppiset elementit, yksittäinen määrite tai kaikki tiettyä elementtiä edeltävät elementit. XPath-määrittely sisältää myös muutaman yksinkertaisen funktion dokumentin sisältämän datan - kuten merkkijonojen - käsittelyyn. XPath on eräänlainen yksinkertainen kyselykieli.

XPath mallintaa XML-dokumentin puurakenteeksi, jossa voidaan liikkua XPath:n ominaisuuksien avulla. Sen luoma puurakenne on abstrakti malli, eikä välttämättä vastaa alkuperäistä rakennetta. Monista muista XML:ään liittyvistä kielistä poiketen XPath ei pohjaudu XML-syntaksiin vaan käyttää omaa, paljon tiiviimpää syntaksia.

Seuraava esimerkki ottaa käsittelyyn elementin luku alielementin kappale, jonka tyyppi-attribuutti on keskeinen:

```
luku//kappale[@tyyppi='keskeinen']
```

Tätä sääntöä voisi soveltaa seuraavaan XML-dokumentin osaan:

```
<luku>
  <kappale tyyppi="keskeinen">
        Tämä kappale on keskeinen.
  </kappale>
</luku>
```

3.2.5 XSL- ja XSLT-prosessorit

XSLT-prosessori on ohjelma, joka muuntaa XML-dokumentin XSL-dokumentiksi. Tämän jälkeen XSL-prosessori tulkitsee dokumentin sisällön ja muotoilee sen esimerkiksi näytölle.

Jotkut XSL- ja XSLT-prosessorit ovat integroituina tietokoneohjelmiin (kuten WWW-selaimiin), joiden täytyy kuitenkin tukea kaikkia kolmea standardia. Tällöin voidaan puhua upotetuista prosessoreista (engl. embedded processors). Microsoftin Internet Explorer on esimerkki upotetusta XSLT-prosessorista.

Kuitenkin jotkut XSL- tai XSLT-prosessorit eivät ole aivan täysin upotettuja ohjelmiin, jolloin voidaan puhua erillisistä prosessoreista (engl. standalone processors). Ne tuottavat erillisiä tiedostoja, joita muut tietokoneohjelmat voivat myöhemmin tulkita. Esimerkiksi Apachen FOP [18] toimii erillisenä XSL-prosessorina.

3.3 DOM

DOM1-määrittely [1] esittelee alustasta ja ohjelmointikielestä riippumattoman, puumaisen (olio)rajapinnan XML- ja HTML- dokumenttien käsittelyyn. Näiden rajapintojen avulla ohjelmat voivat lukea ja muokata dokumenttien sisältöä ja rakennetta sekä liikkua dokumentin rakenteissa. Määrittely koostuu kahdesta päämoduulista:

- Core, joka määrittelee perusrajapinnat XML-rakenteiden käsittelyyn.
- HTML, joka määrittelee rajapinnat HTML- dokumenttien käsittelyyn.

DOM soveltuu käytettäväksi silloin kun dokumenttia ei voida käsitellä suoraviivaisesti solmujen esiintymisjärjestyksessä, vaan rakenteessa voidaan liikkua edestakaisin hyvinkin laajalla alueella. Esimerkiksi editorit ovat sovelluksia, joissa tällainen käsittelytapa on tarpeellinen. DOM-rakenteet ovat usein raskaita ja saattavat - dokumentin koosta riippuen - kuluttaa suuria määriä muistia. Mikäli nämä rajoitukset häiritsevät, kannattaa selvittää riittäisikö kevyempi SAX-rajapinta DOM:n sijasta.

DOM ei määrittele miten rajapinnat pitäisi toteuttaa tai millainen merkitys (semantiikka) dokumentin rakenteilla on - se ainoastaan tarjoaa keinot käsitellä tätä rakennetta ja sisältöä.

DOM:n ensimmäinen taso ei myöskään määrittele esimerkiksi keinoja rakennemäärittelyn käsittelyyn.

DOM2 laajentaa ensimmäisen tason rajapintoja XML-dokumenttien rakenteiden peruskäsittelyn ulkopuolelle. DOM2 on paljon laajempi määrittely kuin DOM1 ja tarjoaa paljon enemmän toimintamahdollisuuksia, jotka ovat tarpeen korkeamman tason sovelluksia - kuten esimerkiksi XML-editoreja - tehtäessä.

DOM2 ei vaadi, että DOM2-yhteensopivan DOM-toteutuksen on toteutettava kaikki kuvatut rajapinnat. Ainoastaan Core-osion perusrajapinnat on toteuttava - kaikki loput osiot ovat vapaaehtoisia. DOM2 tarjoaa myös rajapinnan sovelluksien toteuttamien ominaisuuksien selvittämiseen.

DOM3 [25] täydentää edelleen DOM-spesifikaatiota. Ydinosio määrittää kaksi moduulia (ydin- ja XML-moduuli), jotka DOM-toteutuksen on vähintään toteutettava. DOM-toteutus voisi mahdollistaa kuvan 10 mukaisen jäsentämisen Java-kielellä ilmaistuna.

```
Document doc;
DOMPArser parser = new DOMParser();
String dokumentti="<?xml version='1.0'?>
        <OS nro='12345'><PAIKKA>Paikka</PAIKKA><ASIAKAS>Matti</ASIAKAS>"+
        "<KATU>Kauppakatu</KATU></OS>";
StringReader reader = new StringReader(dokumentti);
InputSource source = new InputSource(reader);
parser.parse(source);
doc = parser.getDocument();
NodeList nodes = doc.getElementsByTagName("PAIKKA");
```

Kuva 10. Esimerkki DOM-jäsennyksestä.

3.4 SAX

SAX [15] eli Simple API for XML on alunperin vain Javalle tarkoitettu XML-ohjelmointirajapinta, joka perustuu jäsennystapahtumien raportointiin funktiokutsujen avulla. Nykyinen versio SAX 2.0.1 tukee jo useaa eri ohjelmointiympäristöä Javan lisäksi. SAX2 rakentuu van-

hemman version SAX1:n päälle ja lisää siihen mm. tuen nimiavaruuksille sekä rajapinnan ominaisuuksien tunnistamiseen. SAX2 on myös laajennettava, joten siihen on (juuri ominaisuuksien tunnistamisen ansiosta) mahdollista lisätä omia rajapintoja rikkomatta alkuperäistä määrittelyä. Tapahtumapohjaisena rajapintana sen etuna on pieni muistintarve: kerrallaan käsiteltävästä dokumentista on muistissa vain juuri käsiteltävänä oleva kohta. Toisaalta suurimpana puutteena on tapahtumapohjaiselle rajapinnalle tyypillinen dokumentissa navigoituvuuden puute: SAX-jäsentäjä käy dokumentin läpi alusta loppuun. Rajapintana SAX on varsin selkeä ja yksinkertainen. Alun perin Javalle tarkoitettuna se myös toteuttaa olio-ohjelmoinnin periaatteita hyvin. Seuraavan kuvan 11 esimerkki havainnollistaa SAX-pohjaista jäsentämistä.

```
public void parse(Reader reader)
        throws SAXException,IOException
{
  XMLReader xmlReader = new SAXParser();
  xmlReader.setContentHandler(this);
  xmlReader.parse(new InputSource(reader));
}
public void startElement(String uri, String localName, String rawName,
                         Attributes attrs)
{
        //käsittele alkuelementti
}
public void characters(char[] characters, int start, int length)
{
        //käsittele data
}
public void endElement(String uri, String localName, String rawName)
{
        //käsittele loppuelementti
}
public void endDocument()
{
        //käsittele dokumentin loppu
}
```

Kuva 11. Esimerkki SAX-jäsennyksestä.

4 JULKAISUJÄRJESTELMÄN TOTEUTTAMINEN

Tässä luvussa käymme läpi XSL-muunnoksen yleisellä tasolla sekä tutustumme Xalaniin, Javan XSL-muunnokseen ja Cocooniin, julkaisujärjestelmän kehykseen.

4.1 XSL-muunnos

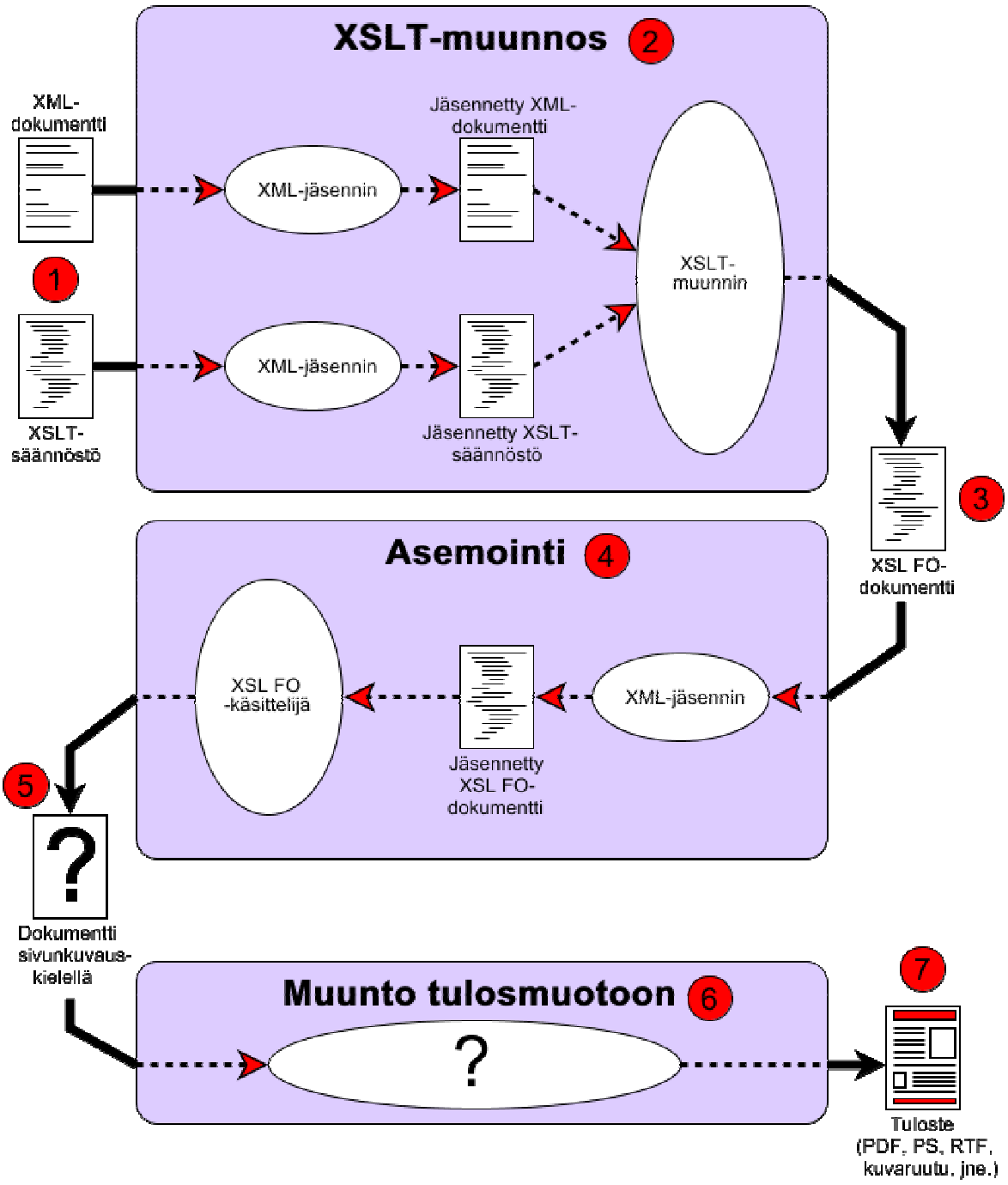
XSL-muunnoksessa XML-dokumentti muutetaan johonkin toiseen dataformaattiin tai erilaiseksi XML-dokumentiksi usean työvaiheen kautta. Työkalut onneksi helpottavat käsittelyprosessia huomattavasti ja käyttäjälle näkyvä järjestelmä vaikuttaa huomattavasti yksinkertaisemmalta kuin mitä se todellisuudessa on. Kuvassa 12 on esitetty XSL-muunnoksen vaiheet.

1 - Alkutilanne

XSL-muunnos perustuu kahteen peruselementtiin: muotoiltavaan XML-dokumenttiin sekä muotoilusäännöt sisältävään XSLT-säännöstöön jotka annetaan syötteenä käsittelyprosessille. Kumpikin dokumentti voi koostua useasta erillisestä osasta, esimerkiksi entiteettien tai xml:includen käytön seurauksena. Tämän lisäksi XSLT-säännöstössä voidaan lukea mukaan syötteenä saatavaan XML-dokumenttiin liitymätöntä materiaalia document-funktion tai erilaisten laajennusten avulla. XML-dokumentti on syötteen muuttuva osa, sillä samaa XSLT-säännöstöä voidaan käyttää usean, samaa rakennetta noudattavan, dokumentin muotoiluun.

2 - XSL-muunnos

Muunnosvaiheen tarkoituksena on muuntaa XML-dokumentti XSLT-sääntöjen avulla XSL FO – dokumentiksi. Muunnosvaiheen alussa XSLT-muunnin lukee molemmat syötedokumentit XML-jäsentimen avulla ja muodostaa niistä itselleen sisäisen rakenteen. Yleensä rakenne on tehokkuussyistä jokin XSLT-sovelluksen tekijöiden kehittämä oma rakenne, mutta se voi olla myös esimerkiksi DOM-puu.



Kuva 12. XSL-muunnoksen vaiheet [24].

Dokumenttien lukemisen jälkeen XSLT-muunnin muuntaa XML-dokumentin annettujen sääntöjen mukaisesti XSL FO – dokumentiksi. Tämä tarkoittaa käytännössä sitä että alkuperäisen syötedokumentin rakenteet korvataan vastaavilla XSL FO – rakenteilla. Muunnoksen tuloksena saatava XSL FO – dokumentti on XML-dokumentti, joten se noudattaa edelleen XML-syntaksia.

Muunnosvaihe on samalla myös muotoiluvaihe [29]. Vaikka syötedokumentti ei vielä ole lopullisessa ulkoasussaan, XSL FO – rakenteet kuvaavat dokumentin osien ulkomuodon: osa tekstistä on lihavoitu, osa kursivoitu, otsikot on kirjoitettu isommalla kirjasinkoolla kuin leipäteksti jne. Varsinainen asemointi suoritetaan kuitenkin vasta myöhemmin.

Muunnosvaihetta ei tarvittaisi mikäli XML-dokumentti kirjoitettaisiin alusta lähtien XSL FO – muotoon tai sovellukset tuottaisivat suoraan XSL FO – muotoisia dokumentteja. XML-dokumenttien muuntaminen XSL FO – muotoon käsityönäkin on mahdollista, mutta se on erittäin työlästä ja virhealtista – silloin tehdään sama muunnosvaiheen työ kuin tässä mutta vaikeammalla tavalla.

3 - XSL FO -dokumentti

Edellä kuvatun XSL-muunnoksen seurauksena, jonkin toisen sovelluksen tuottamana tai käsin kirjoitettuna on muodostettu XSL FO -dokumentti, joka kertoo millaisiksi tekstin eri osat pitää muotoilla. Dokumentti noudattaa edelleen XML-syntaksia mutta alkuperäisessä syötedokumentissa olleita elementtejä ei enää ole jäljellä ja sisältökin voi olla erilainen tai eri järjestyksessä kuin alun perin. Esimerkiksi "<otsikko>XSL-muunnos</otsikko>" elementin paikalla voi olla XSL FO -elementtejä, jotka kertovat miten kyseisen elementin sisältö pitäisi muotoilla, sekä automaattisesti lisättyä tekstiä:

```
<fo:block
    text-align="center"
    color="black"
    font-size="40pt"
    font-family="Helvetica"
    space-before="3cm"
    space-after="1cm" > XSL-muunnos</fo:block>
```

XSL FO -dokumentti ei kerro miltä lopputuloksena saatava dokumentti tarkalleen näyttää. Lopullinen ulkoasu määräytyy vasta asemointivaiheessa.

4 - Asemointi

Asemointivaiheessa muotoiltu dokumentti saa lopullisen ulkoasunsa eli se muunnetaan XSL FO -kielestä jonkin sivunkuvauskielen sääntöjen mukaiseksi. Tässä vaiheessa XSL FO -dokumentti joutuu ensimmäistä (ja oikeastaan viimeistä) kertaa XSL FO -sovelluksen käsiteltäväksi. Koska syötteenä saatava XSL FO -dokumentti noudattaa XML-syntaksia, on se jälleen (kuten muunnosvaiheessakin) luettava sovelluksen sisäisiin tietorakenteisiin jäsentimen välityksellä.

Käsiteltävä dokumentti on tähän mennessä muotoiltu XSL FO -syntaksilla ja se täytyy vielä asemoida. Asemoinnin ja muotoilun välinen ero ei välttämättä ole kovinkaan selkeä. Yksinkertaistaen voidaan sanoa, että muotoilussa päätetään miltä lopputulos näyttää ja asemoinnissa kerrotaan missä kohdassa eri elementit sijaitsevat. Muotoilussa päätetään esimerkiksi minkä kokoisia kirjaimet ovat, millaista kirjaimistoa käytetään, mitä värejä käytetään, paljonko otsikoiden jälkeen on tyhjää tilaa, miten teksti jakautuu kappaleisiin jne. Sen sijaan esimerkiksi kunkin kirjaimen tarkka sijainti sivulla (oli kyseessä sitten paperinen tai elektroninen sivu), tekstin rivitys tai sivutus, sivunumerointi jne. kuuluvat asemointiin.

Muotoilu voi myös olla kohteesta (esimerkiksi päätelaitteesta, jolla lopputulos on tarkoitus näyttää) riippumatonta: otsikko on aina samankokoinen, riippumatta siitä, näkyykö se paperilla vai matkapuhelimen näytöllä. Toisaalta asemoinnissa on aina otettava kohde huomioon: paperin yhdelle riville mahtuva teksti ei varmasti sellaisenaan mahdu multimediakännykän näytölle vaan se on katkaistava useisiin riveihin.

Tulosdokumentissa käytetty sivunkuvauskieli riippuu käytetystä sovelluksesta. Lähes aina se on sovelluksen sisäinen tietorakenne.

5 - Asemoitu dokumentti

Asemoinnin jälkeen dokumentti on kuvattuna sivunkuvauskielelle. Tätä muotoa ei ole määritelty XSL-määrittelyssä vaan se on käytetyn sovelluksen päätettävissä. Käytännössä käyttäjä ei koskaan näe dokumenttia, se on olemassa vain sovelluksen sisällä.

6 - Muunnos tulosteeksi

Tässä viimeisessä työvaiheessa sivunkuvauskielellä kirjoitetusta dokumentista tehdään lopullinen tuloste. Dokumentin ulkoasu on jo määrätty mutta se on vielä kuvattava haluttuun tulosmuotoon.

Useimmiten kaikki aikaisemmat vaiheet ovat samoja riippumatta lopputulosteen muodosta ja ero eri tulosmuotojen välillä tehdään tässä vaiheessa. Näin esimerkiksi uuden tulosmuodon lisääminen sovellukseen aiheuttaa muutoksia vain tässä yhdessä vaiheessa ja muut vaiheet voidaan säilyttää ennallaan. Vaihe on tiukasti sidottu asemointivaiheeseen eikä käyttäjä näe mitään eroa näiden välillä.

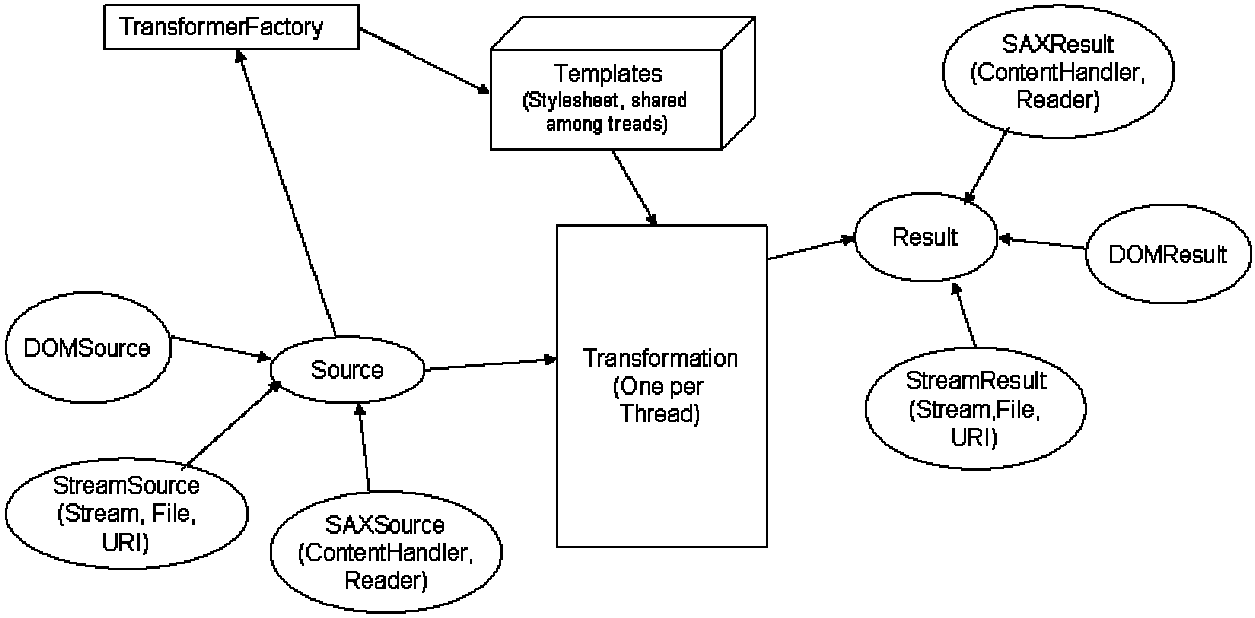
7 - Lopullinen tuloste

Viimeisen työvaiheen jälkeen lopullinen katseltava tai tulostettava dokumentti on valmis. Yleisiä tulosmuotoja ovat mm. Postscript (PS), PDF, RTF tai DVI. Erillinen tiedosto ei kuitenkaan ole ainoa mahdollinen tulos. Muita mahdollisuuksia ovat esimerkiksi dokumentin näyttäminen suoraan ruudulla, sen muuttaminen ohjelmakomponentiksi tai vaikkapa audiomuotoon ääniksi ja puheeksi.

4.2 Xalan-Java

Xalan-Java on XSLT-prosessori XML-dokumenttien muuntamiseksi HTML:ksi, tekstiksi tai toisiksi XML-dokumenteiksi [20]. Se sisältää W3C:n suositukset XSL Transformations (XSLT) – ja XML Path Language (XPath) – tekniikoista. Sitä voidaan kutsua komentoriviltä, käyttää appletissa tai servletissä tai moduulina toisessa ohjelmassa.

Xalan-Java sisältää kuvan 13 mukaisen rajapinnan TraX (Transformation API for XML), joka kuuluu osana rajapintaan JAXP (Java API for XML Processing). TraX takaa modulaariset puitteet ja standardin rajapinnan XML-dokumenttien muuntojen suorittamiseksi, ja se hyödyntää järjestelmän ominaisuuksia määrittäessään, mitä muunninta ja mitä XML-jäsennintä käytetään.



Kuva 13. TraX-muunnosrajapinnan toiminta [20].

Xalan-muunnos toteuttaa kohdassa 4.1 esitellyn muunnoksen.

4.2.1 Esimerkkisovellus

Tutkielmaa varten toteutin yksinkertaisen monikanavajulkaisujärjestelmän, jonka avulla XML-muotoista tietokannan dataa muunnetaan neljään eri muotoon: HTML, PDF, RTF ja WML. RTF- ja WML-muunnos esitellään tarkemmin kohdassa 4.3.2, koska niissä käytetään hyödyksi Cocoon-julkaisuputkea. Liitteenä 6 on esitetty Java-servletti muunnosten toteuttamiseksi.

HTML-muunnokseen tarvitaan seuraavat komponentit:

- xml2html.xsl, tyylitiedosto joka määrittelee minkä näköinen HTML-raportti on (liite 1)
- kirjasto.xml, XML-tiedosto missä on raporttiin kirjoitettava tieto generoituna tietokannasta (liite 2)
- Java-koodi (kuva 14), joka suorittaa varsinaisen muunnoksen ja kirjoittaa raportin (liite 3).

```
try{
     //alustetaan muuntaja
     javax.xml.transform.TransformerFactory tFactory =
     javax.xml.transform.TransformerFactory.newInstance();

     // luodaan varsinainen muuntaja
     javax.xml.transform.Transformer transformer = tFactory.newTransformer
     (new javax.xml.transform.stream.StreamSource("c:\\tmp\\xml2html.xsl"));

     // suoritetaan muunnos, parametrina lähde- ja kohdetiedosto
     transformer.transform
    (new javax.xml.transform.stream.StreamSource("c:\\tmp\\kirjasto.xml"),
     new javax.xml.transform.stream.StreamResult
    (new java.io.FileOutputStream("c:\\tmp\\kirjasto.html")));

}catch (TransformerException te){};
```

Kuva 14. HTML-muunnoksen suorittava Java-koodi.

Kuvan 14 Java-koodissa alustetaan ensin Transformer-muuntaja johon sidotaan xsl-tiedosto. Tämän jälkeen tehdään varsinainen muunnos, parametreina ovat lähdetiedosto joka sisältää tiedon xml-muodossa ja kohdetiedosto joka sisältää muunnoksen lopputuloksen html-muodossa.

Esimerkkisovelluksessa toteutettu PDF-muunnos poikkeaa hieman HTML-muunnoksesta koska siinä käytetään muunnokseen *FOP*:ia. FOP (Formatting Object Processor) on Java-ohjelma joka käyttää apunaan XML-parseria ja muuntaa XSL-tiedostossa annettujen ohjeiden avulla XML-dokumentin haluttuun muotoon. Tuettuja muotoja ovat mm. PDF, PCL, PS, SVG, AWT, MIF ja TXT. FOP ei sisällä omaa XSLT-muunninta tai XML-jäsennintä vaan käyttää ulkoisia sovelluksia, kuten Apachen muiden projektien tuottamia Xalania ja Xercesiä [18]. Kuvassa 15 on esitetty tarvittava Java-koodi muunnoksen suorittamiseksi. Vastaava PDF-dokumentti on esitetty liitteessä 5.

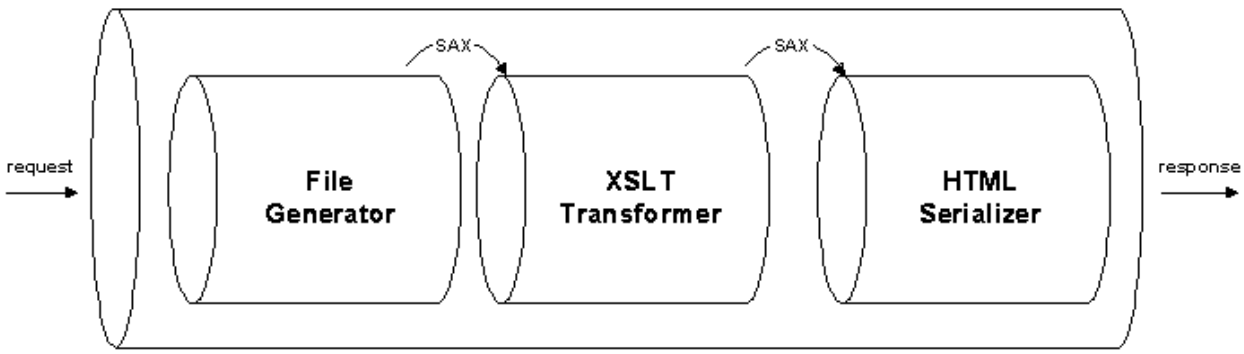
Kuvan 15 muunnos eroaa kuvan 14 HTML-muunnoksesta siten että alussa asetetaan FOP-ajuri ja sille määritellään muuntajaksi PDF. Tämän jälkeen ohjataan muunnos kulkemaan FOP:n kautta. Muunnosprosessi kulkee tästä eteenpäin samalla tavalla kuin edellä kuvattu HTML-prosessi.

```
try{
        Driver driver = new Driver();
        driver.setRenderer(Driver.RENDER_PDF);

        //Asetetaan kohdetiedosto FOP:lle
        driver.setOutputStream(new
        java.io.FileOutputStream("c:\\tmp\\kirjasto.pdf"));

        //Varmistetaan että XSL-muunnoksen tulos menee FOP:n kautta
        Result res2 = new SAXResult(driver.getContentHandler());

        //Asetetaan lähdetiedosto
        Source src = new StreamSource("c:\\tmp\\kirjasto.xml");

        //Asetetaan muuntaja
        Source xsltSrc = new StreamSource("c:\\tmp\\hanna.xsl");
        TransformerFactory transformerFactory = TransformerFac-
        tory.newInstance();
        Transformer transformer = transformerFac-
        tory.newTransformer(xsltSrc);

        //Aloitetaan muunnosprosessi
        transformer.transform(src, res2);

}catch(TransformerException te) {};
```

Kuva 15. PDF-muunnoksen suorittava Java-koodi.

4.3 Cocoon – kehys

Cocoon on Apache Cocoon -projektin toteuttama avoimeen lähdekoodiin perustuva julkaisujärjestelmän kehys [19]. Cocoonin toiminta perustuu XML-kieleen, XSL-muunnoksiin ja joukkoon valikoituja yleiskäyttöisiä komponentteja, joista kootaan putkilinjoja (pipeline) kuvan 16 mukaisesti.



Kuva 16. Putkilinjan idea [26].

Putkilinjat rakennetaan sivukarttojen (sitemap) avulla; sivukartat ovat myös XML-dokumentteja. Kukin linja alkaa syöttäjällä, jatkuu vapaavalintaisella määrällä käsittelijöitä ja päättyy tulostajaan. Data kulkee putkilinjojen sisällä XML-muodossa SAX-tapahtumina. Kuvassa 17 on esimerkki sivukartasta.

```
<map:match pattern="hello.html">
        <map:generate src="docs/samples/hello-page.xml"/>
        <map:transform src="stylesheets/page/simple-page2html.xsl"/>
        <map:serialize type="html"/>
</map:match>
```

Kuva 17. Esimerkki sivukartasta.

4.3.1 Muunnoksen vaiheet

Tutustutaan seuraavaksi lyhyesti putkilinjan komponentteihin ja muunnoksen vaiheisiin.

Syöttäjät: tuottajat, lukijat ja koostajat

Tuottaja (generator) syöttää tiedon putkilinjaan SAX-tapahtumina. Lukijan (reader) avulla Cocoonilla voidaan julkaista staattisia resursseja (kuvat, CSS-tiedostot) sellaisenaan. Lukija on putkilinjan erikoistapaus; sen perään ei voi kytkeä muita komponentteja. Koostajan (aggregator) avulla putkilinjaan voidaan syöttää useammasta XML-dokumentista koostettu dokumentti.

Käsittelijät: muuntimet ja toiminnot

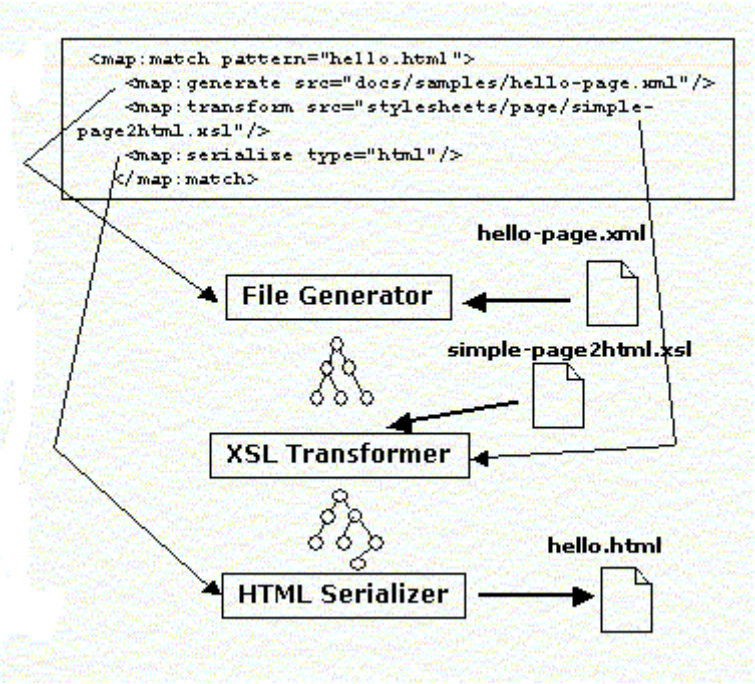
Putkilinjan keskeisin komponentti on muunnin (transformer). Tärkein muunnin on XSLT-muunnin. Muita muuntimia esim. XInclude- ja SQL-muuntimet. Muuntimilla toteutetaan dokumenttien ulkoasun määrittelyn lisäksi myös suuri osa putkilinjojen toimintalogiikasta. Toimintojen (action) avulla voidaan osana putkilinjan toiminnallisuutta esimerkiksi lähettää sähköpostia tai tallentaa tietoa tietokantaan.

Tulostajat: sarjallistajat

Putkilinjat päätetään (lukijalla aloitettuja lukuunottamatta) sarjallistajaan (serializer). Cocoonin mukana tulevien sarjallistajien avulla voidaan esimerkiksi tuottaa:

- HTML- tai XHTML-muotoisia dokumentteja XHTML-sanaston perusteella,
- SVG-, PNG- tai JPEG-muotoisia kuvia SVG-sanaston perusteella tai
- erityisesti tulostamiseen soveltuvia dokumentteja XSL-FO – sanaston perusteella

Putkilinjan prosessointia on havainnollistettu kuvassa 18.



Kuva 18. Putkilinjan prosessointi [26].

Putkilinjan valinta: sovittimet ja valitsimet

Sovittimen (matcher) avulla valitaan putkilinja, johon asiakkaan (selaimen) kutsu välitetään. Valinta tehdään yleensä URI:n rakenteen perusteella:

```
<map:match pattern="hello.html">
        <map:generate src="docs/samples/hello-page.xml"/>
        <map:transform src="stylesheets/page/simple-page2html.xsl"/>
        <map:serialize type="html"/>
</map:match>
```

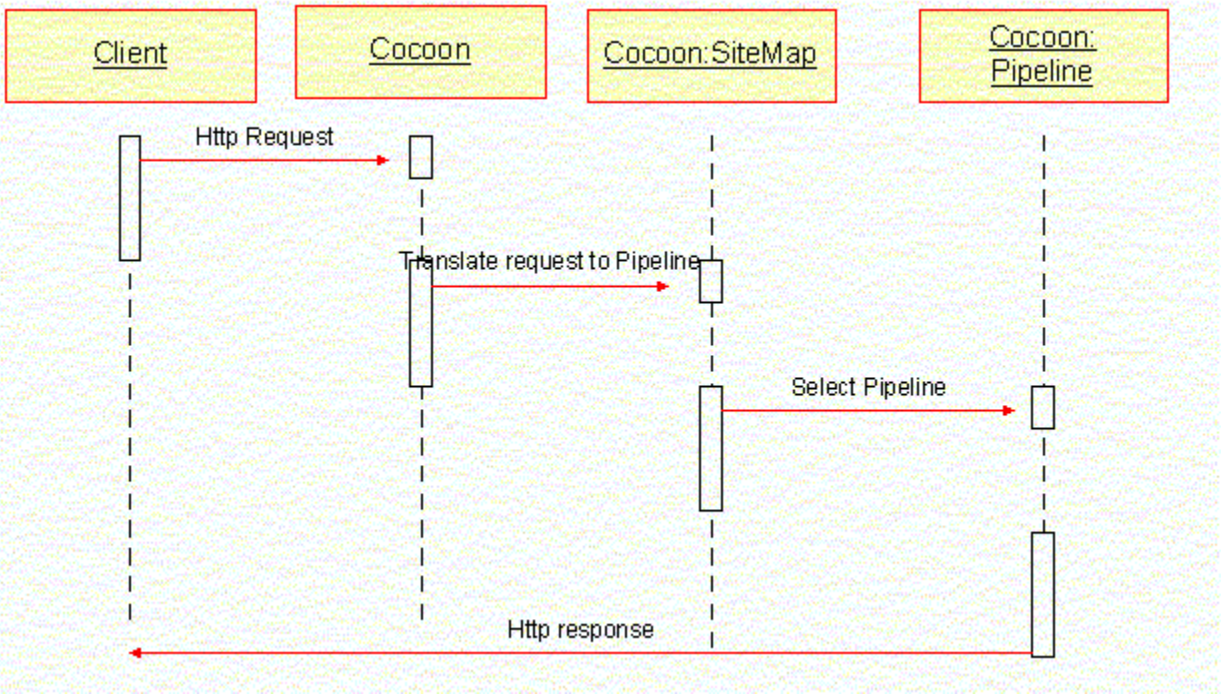
Valitsimen (selector) avulla voidaan parametrisoida putkilinjoja esimerkiksi pyynnön otsikkotietojen (selaimen nimi, jne.) avulla.

4.3.2 Esimerkkisovellus

Esimerkkisovelluksessa WML- ja RTF – muunnoksissa on käytetty Cocoonin julkaisuputkea. Varsinainen java-luokan koodi on yksinkertainen:

```
String cmd = "rundll32 url.dll, FileProtocolHandler
http://localhost:8080/cocoon/kirjasto.wml";
Process p = Runtime.getRuntime().exec(cmd);
```

Lähetetty http-pyyntö Cocoonille etenee kuvan 19 mukaan. Ensin lähetetään http-pyyntö Cocoonille, joka lähettää pyynnön edelleen SiteMapille. SiteMapissa määritellään käytettävä putkilinja (kuva 20), jonka suorituksen jälkeen lähetetään http-vastaus takaisin asiakassovellukselle.



Kuva 19. Http-pyynnön eteneminen Cocoon-muunnosprosessissa [26].

Esimerkkisovelluksessa Sitemap.xmap-tiedostoon on määritelty seuraavanlainen putkilinja:

```
<map:match pattern="kirjasto.wml">
        <map:generate src="c:/tmp/kirjasto.xml"/>
        <map:transform src="examples/style/xml2wml.xsl"/>
        <map:serialize type="wml"/>
</map:match>
```

Kuva 20. Putkilinjan määrittely.

Kuva 20 määrittelee tulostiedoston nimen, käytettävän lähdetiedoston ja muunnokseen käytettävän tyylitiedoston. Liitteissä 4 ja 7 on esitetty vastaavat rtf- ja wml-muotoiset dokumentit.

5 YHTEENVETO

Opinnäytetyön tuloksena valmistui julkaisuympäristö, jolla voidaan tuottaa sisältöä erimuotoisiksi kohdetiedostoiksi. Periaatteena on, että XML-muotoinen sisältö voidaan muuntaa halutuksi sivunkuvauskieleksi luomalla ainoastaan tarvittava ulkoasu XSL-tyylitiedostolla. Rakenteen ja sisällön erottamisella helpotetaan palveluiden ylläpitoa ja päivittämistä.

Julkaisuympäristöön valitsin Apachen tuottaman Cocoon-rajapinnan sekä Xalan-Javan XML-muotoisten dokumenttien muuntamiseen. Sovellustoiminnallisuuden ohjelmoimiseen käytettiin Javaa, jolloin tuotanto voi käyttää palveluissaan mahdollisimman yleisesti tunnettua ohjelmointikieltä sitoutumatta mihinkään yhden ohjelmiston vaatimaan kieleen, jota ei voida hyödyntää muualla.

Javan käyttäminen ohjelmointikielenä tehostaa ja nopeuttaa omalta osaltaan mobiilipalveluita nykyisestään. Lisäksi Java tarjoaa hyvät edellytykset uusien palveluiden kehittämiseen. Koska Javaa käytetään yleisesti kaikessa sovellustuotannossa, se mahdollistaa joustavan työskentelyn yhteistyökumppaneiden kanssa. Sovelluksien ja palveluiden integroiminen toisiinsa helpottuu.

VIITELUETTELO

[1] Bradley, N. (2002) *The XML companion, third edition.* Addison-Wesley, London

[2] Korpiaho, M., Korhonen, M., Inkinen, S., Södergård, C., Heinonen, A., Villi, M., Karvonen, K., Parkkinen, J., Paukkunen, M., Jääskeläinen, M. (2004) *Multi-Channel Solutions.* IT Press, Suomi

[3] Heikniemi, J. (2001) *Mikä on XML?* WWW-sivusto, http://www.heikniemi.net/kirj/moxml.html (05.09.2005)

[4] Silvo, I. *Tulevaisuuden televisio – Tietoyhteiskunnan väylä joka kotiin.* WWW-sivusto, http://www.nettiradio.fi/foorumi/sisalto/297/tulevaisuudentelevisio.html (01.10.2005)

[5] Kuhn, K. J. *HDTV Television - An Introduction*. WWW-sivusto, http://www.ee.washington.edu/conselec/CE/kuhn/hdtv/95x5.htm (01.10.2005)

[6] Järvinen, P. (1996) *Tietotekniikan termit*. WSOY, Juva.

[7] Kalliola, J. *Monikanavajulkaiseminen.* WWW-sivusto, http://www.media.hut.fi/~as75108/2004/2-monikanavajulkaiseminen.pdf (01.10.2005)

[8] Giant Steps (2003) *Create Once, Produce Many, or Obey the Content Master.* WWW-sivusto, http://www.giantstepsmts.com/crossmedia.htm (01.10.2005)

[9] Yli-Koivisto, J. (2000) *WWW-sivujen personointi.* http://www.helsinki.fi/~ylikoivi/documents/www_personointi.pdf (01.10.2005)

[10] Nykänen, O. (2002) *Katsaus W3C:n XML-suosituksiin sekä suositus XML-strategiaksi.* WWW-sivusto, http://www.w3c.tut.fi/reports/2002/1101xml-strategy/index.html (01.10.2005)

[11] W3C (2003) *About the World Wide Web Consortium (W3C).* WWW-sivusto, http://www.w3.org/Consortium/ (01.10.2005)

[12] Garshol, L. M. (1999) *An Introduction to XML.* WWW-sivusto, http://www.garshol.priv.no/download/text/xml-intro/index-en.html (01.10.2005)

[13] W3C (2001) *Extensible Stylesheet Language (XSL) Version 1.0.* WWW-sivusto, http://www.w3.org/TR/xsl/ (01.10.2005)

[14] Juonoja, T. *XSL ja sen hyödyntäminen XML-dokumenteissa.* http://www.mit.jyu.fi/luk/toteutettuja/XSL/ (01.10.2005)

[15] SourceForge (2004) *SAX.* WWW-sivusto, http://www.saxproject.org/ (01.10.2005)

[16] W3C (1999) *XML Path Language (XPath) Version 1.0.* WWW-sivusto, http://www.w3.org/TR/xpath (01.10.2005)

[17] W3C (1999) *XSL Transformations (XSLT) Version 1.0.* WWW-sivusto, http://www.w3.org/TR/xslt (01.10.2005)

[18] The Apache XML Graphics Project (2005) *FOP.* WWW-sivusto, http://xmlgraphics.apache.org/fop/ (01.10.2005)

[19] The Apache Cocoon Project (2005) *The Apache Cocoon Project.* WWW-sivusto, http://cocoon.apache.org/2.1/index.html (08.10.2005)

[20] The Apache XML Project (2005) *Xalan-Java Version 2.7.0.* WWW-sivusto, http://xml.apache.org/xalan-j/ (09.10.2005)

[21] Lajos Moczar (2002), *Transform data into Web applications with Cocoon*. WWW-sivusto, http://www.javaworld.com/javaworld/jw-09-2002/jw-0920-cocoon.html (09.10.2005)

[22] Sun Microsystems (2001), *Java 2 Platform, Micro Edition.* WWW-sivusto, http://java.sun.com/j2me/j2me-ds-0201.pdf (20.11.2005)

[23] Sun Microsystems (2002), *MobileInformation Device Profile.* WWW-sivusto, http://java.sun.com/products/midp/midp-ds.pdf (20.11.2005)

[24] Ruini, H. (2001), *XSL-sovellusten toimintamalli: teoria ja käytäntö*.WWW-sivusto, http://www.cs.helsinki.fi/u/ruini/structure/xsl/xsl/XSL-toimintamallit.html (18.12.2005)

[25] Le Hégaret, P., Wood, L., Robie, J. (2004), *What is the Document Object Model?*. WWW-sivusto, http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/introduction.html (18.12.2005)

[26] The Apache Cocoon Project (2005), *Understanding Apache Cocoon.* WWW-sivusto, http://cocoon.apache.org/2.1/userdocs/concepts/index.html (18.12.2005)

[27] Perttula, K. (2003), *XML-skeemat.* WWW-sivusto, http://www.mit.jyu.fi/luk/toteutettuja/XML-skeemat/ (18.12.2005)

[28] Walsh, N. (1999), *Understanding XML schemas.* WWW-sivusto, http://www.xml.com/pub/a/1999/07/schemas/index.html (18.12.2005)

[29] Stylus Studio (2004), *Processing a Stylesheet.* WWW-sivusto, http://www.stylusstudio.com/w3c/xslfo/N1018E.htm#N1018E (20.12.2005)

Liite 1. xml2html.xsl

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

<xsl:output method="html"/>

<xsl:template match="/">
        <HTML>
          <HEAD>
            <TITLE>HTML-Raportti</TITLE>
          </HEAD>
          <BODY>
            <xsl:apply-templates/>
          </BODY>
        </HTML>
</xsl:template>

 <xsl:template match="data">
       <xsl:apply-templates/>
 </xsl:template>

 <xsl:template match="customer-details-title">
       <h2>
         <xsl:apply-templates/>
       </h2>
 </xsl:template>

 <xsl:template match="customer-name">
       <p>
         <xsl:apply-templates/>
       </p>
 </xsl:template>

<xsl:template match="i">
        <i>
           <xsl:apply-templates/>
        </i>
</xsl:template>

<xsl:template match="customer-title">
        <xsl:apply-templates/>
</xsl:template>

<xsl:template match="p">
        <p>
           <xsl:apply-templates/>
        </p>
</xsl:template>

<xsl:template match="loan-details-title">
       <h2>
         <xsl:apply-templates/>
       </h2>
</xsl:template>
<xsl:template match="customer-description">
       <xsl:apply-templates/>
```

```
    </xsl:template>

    <xsl:template match="loan-description">
            <b>
              <xsl:apply-templates/>
            </b>
    </xsl:template>

    <xsl:template match="table">
      <table border="1">
        <tr>
          <th>Kirjailija</th>
          <th>Kirjan nimi</th>
          <th>Kirjan kuvaus</th>
           <th>Sivumäärä</th>
         </tr>
        <xsl:apply-templates/>
      </table>
    </xsl:template>

    <xsl:template match="row">
      <tr>
        <xsl:apply-templates/>
      </tr>
    </xsl:template>

    <xsl:template match="cell">
      <td>
        <xsl:apply-templates/>
      </td>
    </xsl:template>

</xsl:stylesheet>
```

Liite 2. kirjasto.xml

```
<?xml version="1.0"?>
  <data>
    <customer-details-title>Asiakastiedot</customer-details-title>
    <customer-description>
      <customer-title>Nimi: </customer-title>
      <customer-name>Hanna Testityyppi</customer-name>
      <customer-title>Sosiaaliturvatunnus:  </customer-title>
      <customer-name>123456-123</customer-name>
      <customer-title>Osoite:   </customer-title>
      <customer-name>Gradutie 3, 00100 Helsinki</customer-name>
      <customer-title>Puhelinnumero:   </customer-title>
      <customer-name>09-1234567</customer-name>
    </customer-description>

    <loan-details-title>Lainassa olevat kirjat</loan-details-title>
      <table>
        <row>
          <cell>
            <customer-title>A.A.Milne</customer-title>
          </cell>
          <cell>
            <customer-title>Nalle Puh</customer-title>
          </cell>
          <cell>
            <customer-title>Nalle Puhin seikkailuja.</customer-title>
          </cell>
          <cell>
             <customer-title>87</customer-title>
          </cell>
        </row>
        <row>
          <cell>
            <customer-title>Walt Disney</customer-title>
          </cell>
          <cell>
             <customer-title>Aku Ankan taskukirja</customer-title>
          </cell>
          <cell>
            <customer-title>Aku Ankka ja veljenpojat kaivavat     aar-
        retta.</customer-title>
          </cell>
          <cell>
            <customer-title>23</customer-title>
          </cell>
        </row>
      </table>
    </data>
```

Liite 3. Tulostiedosto HTML-muodossa

Asiakastiedot

Nimi:

Hanna Testityyppi

Sosiaaliturvatunnus:

123456-123

Osoite:

Gradutie 3, 00100 Helsinki

Puhelinnumero:

09-1234567

Lainassa olevat kirjat

Kirjailija	Kirjan nimi	Kirjan kuvaus	Sivumäärä
A.A.Milne	Nalle Puh	Nalle Puhin seikkailuja.	87
Walt Disney	Aku Ankan taskukirja	Aku Ankka ja veljenpojat kaivavat aarretta.	23

Liite 4. Kirjasto.rtf

Asiakastiedot

Nimi:
Hanna Testityyppi
Sosiaaliturvatunnus:
123456-123
Osoite:
Gradutie 3, 00100 Helsinki
Puhelinnumero:
09-1234567

Lainassa olevat kirjat

A.A.Milne	Nalle Puh	Nalle Puhin seikkailuja.	87
Walt Disney	Aku Ankan taskukirja	Aku Ankka ja veljenpojat kaivavat aarretta.	23

Liite 5. Kirjasto.pdf

Asiakastiedot

Nimi:
Hanna Testityyppi
Sosiaaliturvatunnus:
123456-123
Osoite:
Gradutie 3, 00100 Helsinki
Puhelinnumero:
09-1234567

Lainassa olevat kirjat

A.A.Milne	Nalle Puh	Nalle Puhin seikkailuja.	87
Walt Disney	Aku Ankan taskukirja	Aku Ankka ja veljenpojat kaivavat aarretta.	23

Liite 6. Raportti.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.sql.*;
import java.lang.*;
import org.apache.fop.apps.Driver;
import org.apache.avalon.framework.logger.Logger;
import org.apache.avalon.framework.logger.ConsoleLogger;
import org.apache.fop.apps.XSLTInputHandler;
import org.apache.fop.render.Renderer;
import java.util.logging.Level;
import javax.xml.transform.stream.StreamSource;
import org.xml.sax.InputSource;
import javax.xml.transform.sax.SAXResult;
import javax.xml.transform.*;
import java.net.*;

public class raportti extends HttpServlet {

private Connection con;
private String kirjailija="";
private String kirjan_nimi="";
private String kuvaus="";
private String sivumaara, nimi, hetu, osoite, puh="";
private ResultSet rs,rs2,rs3=null;
private PreparedStatement stmt=null;
private Statement stmt2,stmt3=null;
private boolean pdf, rtf, wml, html=false;

public void doGet(HttpServletRequest req, HttpServletResponse res)
                  throws IOException, ServletException
    {
        PrintWriter out = res.getWriter();

        if ((pdf==false)&&(rtf==false)&&(wml==false)&&(html==false))
        {
                   res.setContentType("text/html");
        }

        if (pdf!=false){
        try{
                   Driver driver = new Driver();
                   driver.setRenderer(Driver.RENDER_PDF);

                   //Setup the OutputStream for FOP
                   Driver.setOutputStream(new
                   java.io.FileOutputStream("c:\\tmp\\kirjasto.pdf"));

                   //Make sure the XSL transformation's result is piped
                   through to FOP
                   Result res2 = new SAXResult(driver.getContentHandler());

                   //Setup XML input
                   Source src = new StreamSource("c:\\tmp\\kirjasto.xml");
```

```
            //Setup Transformer
            Source xsltSrc = new StreamSource("c:\\tmp\\hanna.xsl");
            TransformerFactory transformerFactory = TransformerFac-
            tory.newInstance();
            Transformer transformer = transformerFac-
            tory.newTransformer(xsltSrc);

            //Start the transformation and rendering process
            transformer.transform(src, res2);
            }catch(TransformerException te) {};
}
else if(html!=false){
try{
            javax.xml.transform.TransformerFactory tFactory =
            javax.xml.transform.TransformerFactory.newInstance();
            javax.xml.transform.Transformer transformer = tFac-
            tory.newTransformer (new
            javax.xml.transform.stream.StreamSource("c:\\tmp\\xml2htm
            l.xsl"));
            transformer.transform (new
            javax.xml.transform.stream.StreamSource("c:\\tmp\\kirjast
            o.xml"), new javax.xml.transform.stream.StreamResult( new
            java.io.FileOutputStream("c:\\tmp\\kirjasto.html")));
}
catch (TransformerException te){};

else if(rtf!=false){

            String cmd = "rundll32 url.dll,FileProtocolHandler
            http://localhost:8080/cocoon/kirjasto.rtf";
            Process p = Runtime.getRuntime().exec(cmd);
}

else if(wml){
            String cmd = "rundll32 url.dll,FileProtocolHandler
            http://localhost:8080/cocoon/kirjasto.wml";
            Process p = Runtime.getRuntime().exec(cmd);
}

try
{
            // rekisteröidään ajuri
            Class.forName("org.gjt.mm.mysql.Driver");

            // muodostetaan yhteys kantaan
            String url = "jdbc:mysql://localhost/kirjasto";
            con = DriverManager.getConnection(url,"hanna","hanna");
}
catch (ClassNotFoundException cnfe)
{
            out.println("JDBC-ajurin rekisteröinti epäonnistui.");
}

catch(SQLException se)
{
            out.println("Kantaan ei saatu yhteyttä.");
}

if ((pdf==false)&&(rtf==false)&&(wml==false)&&(html==false)){
try{
```

```
String hakuparametri =
req.getParameter("Asiakastunnus2");
if (hakuparametri.length() == 0){
          out.println("Et syöttänyt asiakastunnusta.");
}
else{
          int hakuInt = Integer.parseInt(hakuparametri);
          stmt2 = con.createStatement();
          rs = stmt2.executeQuery("SELECT k.*, a.* FROM
          kirja AS k, asiakastiedot AS a WHERE
          k.lainaaja = a.asiakastunnus");

          if (!rs.first()){ out.println("Asiakkaalla ei
          ole kirjoja lainassa.");}
          else{
                //tulostetaan sivun kiinteä osuus
                out.println("<html>");
                out.println("<head>");
                out.println("<title>Raportti</title>");
                out.println("</head>");
                out.println("<body>");

          out.println("<h2 align=center> Raportti - Asi-
          akkaalla lainassa olevat kirjat </h2>");
          out.println("<form action =\" http:// local-
          host:8080/examples/servlet/raportti\" method
          =\"post\" >");
          out.println("<table border=1 width=\"30%\"
          align=\"center\" valign=\"middle\">");
          rs.first();
          nimi = rs.getString("nimi");
          hetu = rs.getString("hetu");
          osoite = rs.getString("osoite");
          puh = rs.getString("puh");
          out.println("<tr> <td>Kirjailija</td>
          <td>Kirjan nimi</td> <td>Kirjan kuvaus</td>
          <td>Kirjan sivumäärä</td></tr>");
          FileWriter out2 = new File-
          Writer("c:\\tmp\\kirjasto.xml", false);
          PrintWriter tiedosto = new PrintWriter (out2);

          tiedosto.write("<?xml version=\"1.0\"?>");
          tiedosto.write("<data>");
          tiedosto.write("<customer-details-
          title>Asiakastiedot</customer-details-
          title>");

          tiedosto.write("<customer-description>");
          tiedosto.write("<customer-title>Nimi:
          </customer-title>");
          tiedosto.write("<customer-name>");
          tiedosto.write(""+nimi);
          tiedosto.write("</customer-name>");
          tiedosto.write("<customer-
          title>Sosiaaliturvatunnus:  </customer-
          title>");
          tiedosto.write("<customer-name>");
          tiedosto.write("" + hetu);
          tiedosto.write("</customer-name>");
          tiedosto.write("<customer-title>Osoite:
          </customer-title>");
```

```
                        tiedosto.write("<customer-name>");
                        tiedosto.write("" + osoite);
                        tiedosto.write("</customer-name>");
                        tiedosto.write("<customer-title>Puhelinnumero:
                        </customer-title>");
                        tiedosto.write("<customer-name>");
                        tiedosto.write("" + puh);
                        tiedosto.write("</customer-name>");
                        tiedosto.write("</customer-description>");
                        tiedosto.write("<loan-details-title>Lainassa
                        olevat kirjat</loan-details-title>");
                        tiedosto.write("<table>");
                do{
                        if (rs.getInt("lainaaja") == hakuInt) {
                        kirjailija = rs.getString("kirjailija");
                        kirjan_nimi = rs.getString("kirjan_nimi");
                        kuvaus = rs.getString("kuvaus");
                        sivumaara = rs.getString("sivumaara");

                        tiedosto.write("<row>");
                        tiedosto.write("<cell>");
                        tiedosto.write("<customer-title>");
                        tiedosto.write(""+kirjailija);
                        tiedosto.write("</customer-title>");
                        tiedosto.write("</cell>");
                        tiedosto.write("<cell>");
                        tiedosto.write("<customer-title>");
                        tiedosto.write(""+kirjan_nimi);
                        tiedosto.write("</customer-title>");
                        tiedosto.write("</cell>");
                        tiedosto.write("<cell>");
                        tiedosto.write("<customer-title>");
                        tiedosto.write(""+kuvaus);
                        tiedosto.write("</customer-title>");
                        tiedosto.write("</cell>");
                        tiedosto.write("<cell>");
                        tiedosto.write("<customer-title>");
                        tiedosto.write(""+sivumaara);
                        tiedosto.write("</customer-title>");
                        tiedosto.write("</cell>");
                        tiedosto.write("</row>");

                        out.println("<tr><td><input name=\"Nimi\"
                        type=text value=\""+kirjailija+"\"
                        size=30></td><td><input name=\"Nimi\"
                        type=text value=\""+kirjan_nimi+"\"
                        size=30></td><td><textarea name=\"Nimi\"
                        rows=\"4\">\""+kuvaus+"\"</textarea></td><td><
                        input name=\"Nimi\" type=text
                        value=\""+sivumaara+"\" size=30></td></tr>");
                }
                rs.next();
                }while (!rs.isAfterLast());

                tiedosto.write("</table>");
                tiedosto.write("</data>");
                tiedosto.flush();
                tiedosto.close();
}
}
}catch(SQLException se)
```

```
        {
                out.println("Ei toimi.");
        }

        out.println("</table>");
        out.println("<table align=\"center\"><tr><td colspan=\"4\">Valitse
        hakutietojen tiedostomuoto:</td></tr><tr><td align=\"center\"><input
        type=\"submit\" name=\"PDF\" value=\"PDF\"></td><td
        align=\"center\"><input type=\"submit\" name=\"RTF\"
        value=\"RTF\"></td><td align=\"center\"><input type=\"submit\"
        name=\"WML\" value=\"WML\"></td><td align=\"center\"><input
        type=\"submit\" name=\"HTML\" value=\"HTML\"></td></tr></table>");

        out.println("</form>");
        out.println("</body>");
        out.println("</html>");
        }
        pdf=false;rtf=false;wml=false;html=false;
}

public void doPost (HttpServletRequest req, HttpServletResponse res)
throws ServletException
{
        res.setContentType("text/html");

        //tutkitaan mitä painiketta on painettu ja toimitaan sen mukaan
        try{
        PrintWriter out = res.getWriter();
        if (req.getParameter("PDF") != null)
        {
                pdf=true;
                out.println("Generoitu kirjasto.pdf tallennettiin hake-
                mistoon c:\\tmp");
                doGet(req,res);
        }

        else if (req.getParameter("RTF") != null)
        {
                rtf=true;
                out.println("Odota hetki... Avataan tallennus-ikkunaa.");
                doGet(req,res);
        }

        else if (req.getParameter("WML") != null)
        {
                wml=true;
                out.println("Odota hetki... Avataan tallennus-ikkunaa.");
                doGet(req,res);
        }
        else if (req.getParameter("HTML") != null)
        {
                html=true;
                out.println("Generoitu kirjasto.html tallennettiin hake-
                mistoon c:\\tmp");
                doGet(req,res);
        }
        else out.println("Ei mitään!");
}
catch (IOException io){}
}
}
```

Liite 7. Kirjasto.wml