XML JA SEN MUUNTAMINEN XSL-MUUNNOSTEN AVULLA

Juho Kuoppala
Insinöörityö
Oulun seudun ammattikorkeakoulu
Raahen tietokonealan yksikkö

ALKULAUSE

Tämä opinnäytetyö on tehty Oulun seudun ammattikorkeakoulussa, Raahen tietotekniikan ja liiketalouden yksikössä keväällä 2003, Tampereen yliopiston täydennyskoulutuskeskuksen toimeksiannosta. Insinöörityössä tutkittiin XML:n muuntamista XSL-muunnosten avulla.

Yrityksen puolesta työn ohjaajina ovat toimineet tekniseltä puolelta Pekka Raisio ja insinöörityön liittämisessä kokonaiskehitykseen Kari Virtanen. Koulun puolesta insinöörityön valvojana toimi Eino Niemi.

Kiitokset Pekka Raisiolle teknisen toteutuksen alkuun laittamisesta sekä Kari Virtaselle opastuksesta työn eri vaiheissa. Haluan kiittää myös Eino Niemeä rakentavasta palautteesta, sekä Kaija Heinulaa auttamisesta kieliasuun liittyvissä kysymyksissä.

Raahessa 22.4.03

Juho Kuoppala
Seminaarinkatu 2 as. 310
92100 RAAHE
0503204365
jkuoppal@ratol.fi

TIIVISTELMÄ

Tekijä: Juho Kuoppala
Nimi: XML ja sen muuntaminen XSL-muunnosten avulla
vuosi: 2003
Laajuus: 44 sivua
Saatavuus: Raahen tietokonetekniikan ja liiketalouden yksikön kirjasto

Tämän opinnäytetyön tavoitteena on ollut selvittää, miten XML-dokumenttien muuntaminen tapahtuu XSL-muunnosten avulla. XSL:stä ja sen käytöstä löytyy useita eritasoisia ja -laatuisia oppaita niin kirjallisuudesta kuin Internetistäkin. Kuitenkin riittävän selkeän ja kattavan oppaan puuttuminen loi pohjan tämän opinnäytetyön tekemiselle.

Työn alkuosassa tarkastellaan asioita, joita on huomioitava, sekä millaisia tekniikoita XML:n muuntaminen mahdollisesti vaatii. Työssä esitetään, miten XPathia apuna käyttäen päästään navigoimaan XML:n elementtien välillä ja miten sitä käytetään apuna XSL-muunnoksia tehtäessä. Toteutusosuudessa on esitetty esimerkein, miten XML-dokumentti muunnetaan HTML:ksi tai rakenteeltaan toisenlaiseksi XML-dokumentiksi.

Työn tuloksena on saatu selkeä opas muunnosten käyttämisestä sekä uusi käyttöliittymä OpenMDV-ohjelmaan.

Avainsanat: XML, XSL, XSLT, XPath, Merkkauskielet

ABSTRACT

Author: Juho Kuoppala

Name: Transforming XML using XSL-transformations

Year: 2003

Duration: 44 pages

Availability: Library of the Raahe Institute of Computer Engineering and Business

The aim of this thesis was to find out how XML-documents can be transformed using XSL-transformations. There are many different types of guides about XSL-transformations but the lack of a clear and comprehensive guide has resulted in this thesis.

The beginning of the thesis contains things that must be taken into account when transforming XML-documents. The study contains a guide on how XPath is used to navigate between different elements of XML and how it can be used when executing XSL-transformations. In the study is presented by using examples how to transform XML-documents to HTML or other XML-documents with different structure.

The result of this thesis is a clear guide on using transformations and a new user interface to the OpenMDV program.

Keywords: XML, XSL, XSLT, XPath, markup languages

SISÄLLYSLUETTELO

ALKULAUSE 2
TIIVISTELMÄ 3
ABSTRACT 4
SISÄLLYSLUETTELO 5
KÄSITTEITÄ JA MÄÄRITELMIÄ 7
1. JOHDANTO 8
1.1 Merkkauskielistä 8
1.1.1 SGML 9
1.1.2 HTML 9
1.2 XML 10
1.2.1 DTD 10
1.2.2 Parserit eli jäsentimet 10
1.2.3 Entiteetit 11
1.3 CSS–tyylitiedostot 11
1.4 XSL 11
1.4.1 Muunnoksen suoritustapa 12
1.4.2 XSL-muunnoksen tapahtumat 12
1.5 XPath 13
1.5.1 XPathin akselit 14
1.5.2 Solmutestit 15
1.5.3 Predikaatit 15
2. MÄÄRITELMÄ 19
2.1 Käytetyt ohjelmat 19
3. TOTEUTUS 21
3.1 XSLT-esimerkkejä 21
3.1.1 XSLT - Tiedon näyttäminen ja elementtien paikallistaminen 22
3.1.2 Attribuuttien paikallistaminen 26
3.1.3 Operaattorit 27
3.1.4 Testiarvojen vertaaminen 28
3.2 XML:n rakenteen muuttaminen 33
3.2.1 Elementtien luominen 33
3.2.2 Attribuuttien luominen ja tiedon lajitteleminen 34
3.2.3 Solmujen kopiointi 36
3.3 Muunnosprosessin suorittaminen Cocoonia apuna käyttäen 37

3.4 OpenMDV 40
3.3.1 Käyttöliittymä 41
4. YHTEENVETO JA LOPPUPÄÄTELMÄT 42
LÄHTEET 43
LIITTEET 44

KÄSITTEITÄ JA MÄÄRITELMIÄ

CERN	European Organisation for Nuclear Research
SMGL	Standard Generalized Markup Language; Yleismerkkauskieli
XSP	Extensible Server pages
XML	Extensible Markup language; Dokumenttien kuvaamiseen käytetty metakieli
XSL	Extensible Stylesheet Language
XSLT	Extensible Stylesheet Language Transformation; XML-syntaksia noudattava muunnoskieli.
DTD	Data Type Definition; Tapa kuvata dokumentin kielioppia
Skeema	Microsoftin määrittämä dokumenteille tarkoitettu kielioppi
HTML	Hypertext Markup Language; Webissä käytettävä merkkauskieli
CSS	Cascading Style Sheets; Yksinkertainen tyylikieli
XPath	XML Path Language; XML-dokumentin osien paikantamiseen ja valintaan käytetty kieli
Solmu	Dokumentissa esiintyvä elementti
URI	Uniform Resource Identifier

1. JOHDANTO

Tämä insinöörityö on tehty Tampereen yliopiston täydennyskoulutuskeskukselle, jonka yhtenä toiminta-alueena on projekti- ja konsultointitoiminta, jossa on erikoistuttu tilaajan tarpeista räätälöityyn kehittämis- ja koulutushankkeisiin.

XML-dokumenttien muuntamiseen on useita eri tapoja. Näistä yksi on XSL-muunnokset. XSL-muunnokset ovat suhteellisen uusi tapa muuntaa XML-dokumentteja niin HTML-, XML- kuin muiksikin dokumenteiksi. XSL-muunnoksista on useita ohjeita, mutta selkeän ja XSL:n olennaisista osista koostuvan oppaan puute on luonut pohjan tämän insinöörityön tekemiselle.

Työ liittyy Tampereen yliopiston täydennyskoulutuskeskuksen koordinoimaan Wirlab-tutkimus ja kehitysprojektiin. Wirlab on kiinteiden ja langattomien tietoverkkojen tutkimuskeskus Seinäjoella. Sen tutkimuskohteita ovat kiinteät ja langattomat tietoverkot ja niiden hyödyntäminen ja käyttö erilaisilla päätelaitteilla. Työssäni olen perehtynyt aiemmin yhteistyössä muun muassa CERN:in kanssa tehtyyn OpenMDV-ohjelmaan, joka on tehty tiedon talletusratkaisuksi internetissä.

1.1 Merkkauskielistä

1960-luvulta lähtien on useissa yhteistyöprojekteissa kehitetty tekniikoita painettavan materiaalin käsittelystä. Että tietokoneohjelmat voisivat asetella ja lukea tekstiä oikein, on tekstin rakenne voitava erottaa sen sisällöstä.

Merkkauskielet erottavat dokumenttien fyysisen ja loogisen rakenteen toisistaan. Näin voidaan määritellä esimerkiksi eritasoisia otsikoita, mutta jättää päätös niiden varsinaisesta käytöstä myöhempään vaiheeseen. Dokumentin looginen rakenne eli sen eri osaset ja niiden keskinäiset suhteet määritellään dokumenttityyppimäärittelyssä (DTD). Merkkauskielen tunnisteilla voidaan merkitä tiettyjä alueita tekstistä. Näihin elementteihin liittyy yleensä attribuutteja,

joiden avulla elementeille voidaan antaa tiettyjä, dokumenttityyppimäärittelyssä mainittuja arvoja.

Merkkauskielet eroavat ohjelmointikielistä siinä, etteivät ne sinällään suorita mitään prosessointia. Sen sijaan merkkauskielellä merkattua tekstiä voidaan käsitellä helposti ohjelmointikielten avulla erilaisia tarkoituksia varten. Merkkauskielten avulla voidaan rikastaa dokumentteja eli tuoda niihin lisäinformaatiota, joka puolestaan mahdollistaa dokumenttien tehokkaamman käsittelyn. /2/

1.1.1 SGML

Standard Generalized Markup Language on vuonna 1980 julkistettu kansainvälinen standardi (ISO 8879), ja kuten nimikin sanoo, yleismerkkauskieli. Nimestään huolimatta se on itse asiassa metakieli, jolla voidaan määritellä merkkauskieliä. Myös XML, johon tässä insinöörityössä on keskitytty, on johdettu SGML-kielestä. SGML:lle ominaisia piirteitä ovat sisältöä kuvaava merkkaustapa, dokumenttityypin käsite ja riippumattomuus järjestelmästä, jolla tekstiä kirjoitetaan./2/

1.1.2 HTML

Hypertext Markup Language on tunnetuin ja käytetyin SGML:llä määritelty merkkauskieli. HTML on tarkoitettu nimenomaan verkkojulkaisun rakenteen ja ulkoasun ilmaisemiseen. Sen käyttö on yleistynyt suuresti Internetin yleistymisen myötä. HTML sisältää n. 100 ennalta määrättyä elementtiä, joiden mukaan selaimet tulkitsevat ja näyttävät Internet-sivun sisältämän koodin parhaaksi katsomallaan tavalla. HTML on matalan tason kieli eikä sitä käytetä juuri muualla kuin Internet-sivujen julkaisemisessa.

1.2 XML

XML on World Wide Consortiumin (W3C) määrittelemä kieli. XML perustuu rakenteellisten dokumenttien laajennettavaan ajatusmalliin. XML itsessään ei määrittele kuin dokumentin yleisen rakenteen, jossa esitellään vain kieliversio ja dokumentin juurielementti. Kaiken muun voi tekijä määritellä itse tai käyttää valmiita malleja. Toteutusosuudessa tulen esittämään esimerkin XML-dokumentista, jota tullaan muokkaamaan käyttäen apuna XSL-muunnoksia.

Eräs XML:n vahvuus on se, että sopivia työkaluja on saatavilla pilvin pimein. Lähes kaikille ohjelmointikielille on saatavilla valmiit kirjastot, jotka jäsentävät XML-kielistä lähdetietoa. Niinpä XML-muotoa voi yleensä käyttää omissa ohjelmissa hyvin helposti, koska tallennusmuodon käsittelyrutiineja ei tarvitse kirjoittaa itse. /2/

1.2.1 DTD

DTD on yleinen tapa kuvata dokumentin kielioppi. Jokainen hyvin muodostettu XML-dokumentti noudattaa DTD-määrittelyn tai XML skeeman kielioppia. DTD on pohjana useille nykyisille kuvauskielistandardeille. DTD määrittelee *elementit*, niihin kuuluvat *attribuutit* ja elementtien väliset suhteet.

HTML-kielisten dokumenttien DTD-määrittely sisältyy jokaiseen Internet-selaimeen. XML-dokumenttien DTD voidaan joko kirjoittaa dokumentin alkuun tai linkittää XML-elementtiin ulkoisesta tiedostosta. /2/

1.2.2 Parserit eli jäsentimet

XML:ää, kuten kaikkia muitakin tallennusmuotoja, täytyy jollain tavalla tulkita ennen tietojen käyttöä. Rakenteellisen merkkauksen kohdalla se tarkoittaa sitä, että yksiulotteisesta tekstimassasta poimitaan esiin elementit ja muodostetaan niistä puumalli, jossa elementit ovat sijoitettuna sisäkkäin. Tämä tulkitseminen on parserin eli jäsentimen tehtävä. Yksinkertaisimmillaan jäsennin "vain" lukee rakenteen ja muodostaa siitä puun, mutta käytännössä lähes kaikki jäsentimet myös tekevät ainakin jonkinlaisen muototarkistuksen.

Useimmille jäsentimille riittää se, että dokumentti on oikeamuotoinen, kun taas jotkut hallitsevat myös DTD:n ja/tai skeeman mukaan validoinnin. Tarpeesta riippuu, kuinka perusteellista oikeellisuuden tarkastelua kannattaa tehdä. XML-kielten tulkitsemiseen liittyy se HTML:n tekijöille vieras piirre, että ohjelmat hylkäävät XML-dokumentin, jos siinä on yksikin virhe, kun taas HTML:ää yritetään usein, erityisesti selaimissa, tulkita väkisin virheistä huolimatta. Tämän vuoksi XML-kielten jäsentämisessä syntyy vähemmän tulkintavirheitä, mikä edelleen johtaa yhtenäisempiin ja selkeämpiin toteutuksiin./2/

1.2.3 Entiteetit

Entiteetit ovat lyhenteitä jotka otetaan käyttöön &-merkillä ja niiden määrittely päätetään ;-merkkiin. esimerkiksi HTML:ssä käytetyimpiä entiteettejä ovat skandinaaviset merkit (ä-kirjaimen entiteetti ä). Entiteetit voivat joko sijaita ulkoisissa dokumenteissa, josta ne linkitetään käytettävään dokumenttiin tai ne voivat sijaita itse dokumentin kielioppimäärittelyssä. /1/

1.3 CSS–tyylitiedostot

CSS ei ole varsinainen merkkauskieli vaan yksinkertainen tyylikieli, jolla voidaan liittää tyylimääritykset HTML-, XHTML- tai XML-dokumentteihin. Kullekin elementtityypille voidaan määritellä yksilöllinen tyyli, joka kuvaa esimerkiksi fontit, värit, marginaalit ja sijainnit.

1.4 XSL

XSL:ssä on kaksi puolta – transformaatio eli muunnoskieli ja formatointi eli muotoilukieli. Muunnoskielen avulla mahdollistetaan dokumentin muuntaminen eri muotoihin, kun taas muotoilukielellä voidaan muotoilla ja tyylittää dokumenttiä eri tavoin. Nämä kaksi XSL:n puolta voivat toimia jokseenkin itsenäisesti toisistaan riippumatta, joten XSL:ää voidaan pitää kahtena eri kielenä. Käytännössä dokumentti muunnetaan ennen muotoilua, koska

muunnosprosessissa voidaan lisätä ne tunnisteet, joita muotoiluprosessi tarvitsee. Tässä insinöörityössä ei perehdytä varsinaiseen XSL-muotoiluun, vaan kohdedokumenttien muotoilu hoidetaan muilla tavoin, kuten CSS-tyylitiedostojen avulla. /4/

1.4.1 Muunnoksen suoritustapa

XML-dokumentteja voidaan muuntaa, XSL-muunnoksia apuna käyttäen, kolmella tavalla.

Palvelimella

Palvelinohjelma, kuten Javalla toteutettu servlet, käyttää tyylitiedostoa dokumentin automaattiseen muunnokseen ja tarjoaa muunnetun dokumentin asiakkaalle.

Asiakkaan sovelluksess

Asiakasohjelma, kuten selain, suorittaa muunnoksen lukemalla XML-dokumentin alussa olevan prosessointiohjeen avulla määritellyn tyylitiedoston. Tämä tapa on varsinkin selainvalmistajille vaikea, koska määrittelyt muuttuvat jatkuvasti.

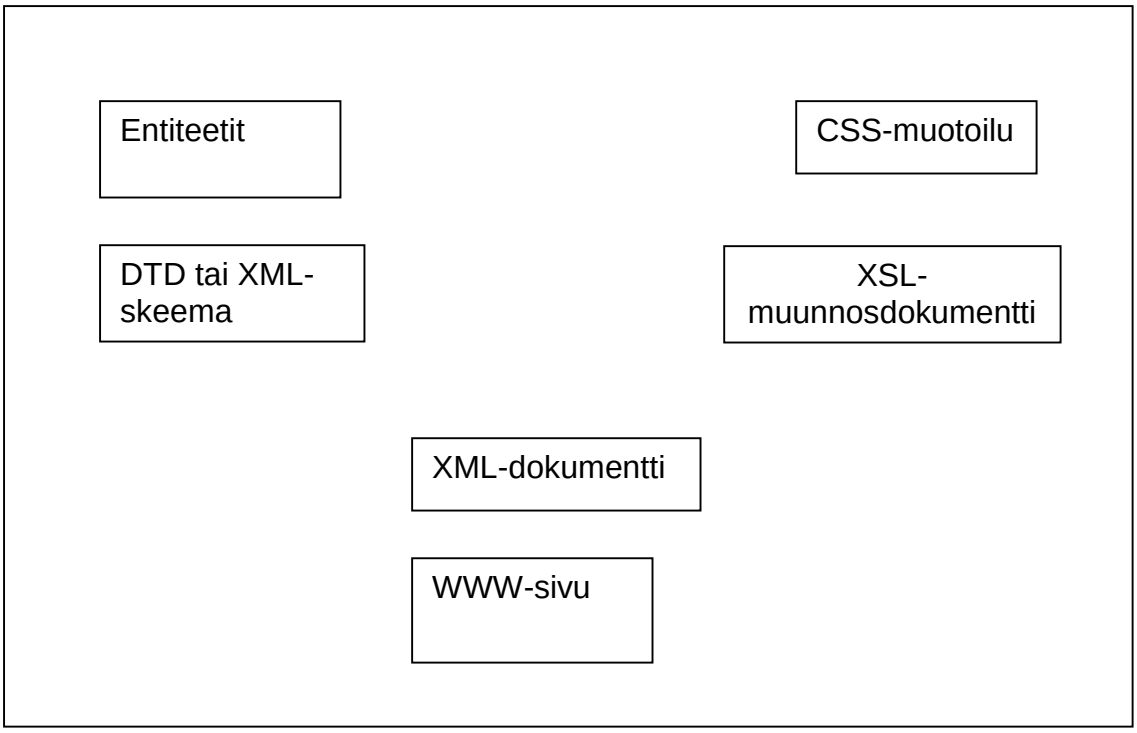
Erillisellä ohjelmalla

Useat itsenäiset ohjelmat, yleensä javaan perustuvat, suorittavat XSL-muunnoksia. Tästä esimerkkinä James Clarkin XT, joka löytyy osoitteesta http://www.jclark.com/xml/ (15.3.2003) /4/

1.4.2 XSL-muunnoksen tapahtumat

XSL-muunnoksilla voidaan siis muokata ja muuntaa XML-dokumentteja. Muunnoksien apuna voidaan käyttää myös useita muita dokumentteja. Kuvassa 1. on esitelty, mitä dokumentteja esimerkiksi WWW-sivun muodostamiseen saatetaan käyttää. XML-dokumenttiin on linkitetty XSL-muunnosdokumentti sekä kieliopin määrittävä DTD-dokumentti. Entiteetit ovat omana tiedostonaan entiteettidokumenttina, johon lyhenteet on kirjattu. XSL-

dokumenttiin on linkitetty CSS-muotoilu, jossa on määritelty tulevalle HTML-sivulle esimerkiksi taustavärit ja taulukoiden reunojen koot. /1/



Kuva 1. WWW-sivu muodostaminen XML-dokumentista

1.5 XPath

XPath on ollut W3C:n suositus marraskuulta 1999. XPath on kieli XML-dokumentin osien paikantamiseen ja valintaan. Sen avulla voidaan paikantaa esimerkiksi kaikki tietyn tyyppiset elementit, yksittäinen määrite tai kaikki tiettyä elementtiä edeltävät elementit. XSL-muunnokset hyödyntävät XPathia suorittaessaan muunnoksia.

”XPathissa käytetään solmun tai solmujoukon määrittämiseksi saantipolkua. Saantipolku muodostuu yhdestä tai useammasta askeleesta, jotka erotellaan /- tai //-merkein. Saantipolkua kutsutaan *absoluuttiseksi saantipoluksi,* mikäli se alkaa /-merkillä. Tällöin polku määritellään juurisolmusta. Muulloin saantipolku

on *relatiivinen* ja alkaa aktiivisesta solmusta, jota kutsutaan *kontekstisolmuksi.* Kontekstisolmua ei koskaan merkitä näkyviin XPath-lausekkeeseen. Syy tähän on se, ettei XPath määrittele mikä, kontekstisolmun pitäisi olla vaan jättää sen XPath-lausekkeita hyödyntäville määrittelyille, kuten XSLT:lle. Askel muodostuu akselista, solmutestistä sekä mahdollisista predikaateista. Esimerkiksi lausekkeessa *child::ELEMENTTI[position() = 5]. child* on akselin nimi, *ELEMENTTI* on solmutesti ja *[position() = 5]* on predikaatti. ” /4/

Mitä nämä akselit, solmutestit ja predikaatit ovat? Niistä seuraavassa.

1.5.1 XPathin akselit

Akseleilla määritetään saantipolku. Akseleita on kaiken kaikkiaan 13 kappaletta ja ne ovat:

- *self,* joka sisältää vain kontekstisolmun itsensä.
- *ancestor,* joka sisältää kontekstisolmun edeltäjät
- *ancestor-or-self,* joka sisältää kontekstisolmun ja sen edeltäjät
- *attribute,* joka sisältää kontekstisolmun attribuutit
- *child*, joka sisältää kaikki kontekstisolmun lapsisolmut.
- *descendant,* joka sisältää kontekstisolmun jälkeläiset
- *descendant-or-self,* joka sisältää kontekstisolmun ja sen jälkeläiset
- *following,* joka sisältää kontekstisolmun kanssa samassa dokumentissa olevat solmut, jotka seuraavat kontekstisolmua
- *following-sibling,* joka sisältää kaikki kontekstisolmua seuraavat samantasioset solmut
- *namespace,* joka sisältää kontekstisolmun nimiavaruussolmut
- *parent,* joka sisältää kontekstisolmun emon.
- *preceding,* joka sisältää kaikki solmut jotka edeltävät kontekstisolmua

- *preceding-sibling,* joka sisältää kaikki kontekstisolmua edeltävät samantasioset solmut. /3/

1.5.2 Solmutestit

Elementtisolmujen valintaan voidaan käyttää niin solmutestinä toimivaa solmun nimeä kuin yleismerkkiä *. Yleismerkin ja solmujen lisäksi voidaan käyttää seuraavia solmutestejä:

- *comment ()* on solmutesti, joka valitsee kommenttisolmun
- *node ()* on solmutesti, joka valitsee minkä tahansa tyyppisen solmun
- *processing-instruction ()* on solmutesti joka, valitsee prosessointiohjesolmun
- *text ()* on solmutesti, joka valitsee tekstisolmun. [4]

1.5.3 Predikaatit

Predikaateissa työskennellään usean tyyppisten lausekkkeiden kanssa. Seuraavassa muutamia predikaattien tyyppejä:

- Solmujoukot . Kuten nimestä voi täätellä, solmujoukot ovat yksinkertaisesti joukko solmuja. Solmun tai solmujen valitsemiseksi solmujoukosta voidaan käyttää useita funktioita joista seuraavassa muutamia:
    - *last ()* on funktio, joka palauttaa solmujen lukumäärän jotka sijaitsevat solmujoukossa.
    - *position ()* on funktio, joka palauttaa kontekstisolmujoukossa olevan sijaitsevan kontekstisolmun sijainnin alkaen 1:stä
    - *id(merkkijono-ID)* on funktio, joka palauttaa solmujoukon jossa on elementti, joka sisältää ID:n avulla funktiolle välitettyn merkkijonon.
    - *name ()* on funktio joka palauttaa solmujoukon ensimmäisen täyden, kvalifioidun nimen. [4]

- Boolen-operaattorit XPathissa voidaan käyttää Boolen arvoja. Mikäli numero on nolla tai tyhjä merkkijono, sitä pidetään epätotena, muulloin se on tosi. Boolen tosi/epätosi-tulokset tuotetaan loogisilla operaatioilla, jotka ovat:
    - *!=* Eri suuri kuin
    - < Pienempi kuin
    - <= Pienempi tai yhtä suuri kuin
    - = Yhtä suuri kuin
    - > Suurempi kuin
    - >= Suurempi tai yhtä suuri kuin.

XPath tukee myös seuraavia funktioita:

- *true ()*-funktio palauttaa aina arvon tosi
- *false()*-funktio palauttaa aina arvon epätosi.
- *not()*-funktio kääntää lausekkeen loogisen merkityksen päinvastaiseksi.
- *lang()*-funktio palauttaa arvon tosi tai epätosi riippuen siinä, onko kontekstisolmun kieli, joka annetaan *xml:lang*-attribuutilla, sama kuin funktiolle välitettävä kieli.

- Numerot XPathissa kaikki numerot, myös kokonaisluvut, on tallennettu liukulukumuodossa. Käytettävät operaattorit ovat:
    - +, eli yhteenlasku
    - –, eli vähennyslasku
    - *, eli kertolasku

- div, eli jakolasku
- mod, joka palauttaa jakojäännöksen.

- Funktiot jotka operoivat numeroilla ovat seuraavia:
  - ceiling (), on funkio, joka palauttaa pienimmän kokonaisluvun, joka on suurempi kuin sille välitetty luku
  - floor (), on funktio, joka palauttaa suurimman kokonaisluvun joka on pienempi kuin sille välitetty luku
  - round (), on funktio, joka pyöristää sille välitetyn luvun kokonaisluvuksi
  - sum (), on funktio, joka paöaittaa sille välitettävien lukujen summan /4/.

- Merkkijonot: XPathissa merkkijonot koostuvat Unicode-merkeistä. merkkijonojen käsittelyyn joukko funktioita joista seuraavassa.
  - starts-with(string jono 1, string jono 2), jos ensimmäisen merkkijonon alkukirjain on sama kuin toisen merkkijonon ensimmäinen kirjain funktio palauttaa arvon tosi
  - contains(string jono 1, string jono 2), mikäli ensimmäinen merkkijono sisältää toisen merkkijonon, funktio palauttaa arvon tosi
  - substring(string jono1, arvo offset, arvo length), funktio palauttaa annetusta merkkijonosta sen osan, joka on määrätty alkamaan offsetillä ja jonka pituus on määrätty lenghtillä
  - substring-before(string jono1, string jono2), funktio palauttaa osan merkkijonon alusta lähtien, kunnes jono2 esiintyy ensimmäisen kerran.

- substring-after(string jono1, string jono2), palauttaa osan jono1:stä jono2:n esiintymisen jälkeen.
- string-length(string jono1), tämä funktio palauttaa merkkijonon jono1 pituuden
- normalize-space(string jono1), on funktio, joka muuttaa useamman tyhjän merkin yhdeksi tyhjämerkiksi ja poistaa kokonaan aloittavan ja lopettavan tyhjämerkin
- concat(string jono1, string jono2, string jono3, ...), concat-funktiolla yhdistetään jono1, jono2, jono3, ... yhdeksi merkkijonoksi
- format-number(number luku1, string jono2), Funtio palauttaa merkkijonon, joka sisältää luku1:n muotoillun merkkijonoversion käyttämällä jono2:ta muotoilumerkkijonona. /4/

tulospuukatkelmat XML-dokumentissa voi olla osia jotka eivät ole täydellisiä solmuja tai solmujoukkoja. Näitä osia kutsutaan tulospuukatkelmiksi XPathissa tulospuukatkelmien käyttäminen on suhteellisen pientä. Tulospuukatkelmat voidaan joko muuntaa merkkijonoiksi string() tai boolen-arvoiksi boolean(). /4/

2. MÄÄRITELMÄ

Tämän insinöörityön tarkoituksena on perehtyä XSL-muunnoksiin, joiden avulla päästään muokkaamaan sekä XML-dokumenttia itseään että muuntamaan XML:ää esimerkiksi HTML-muotoon. Työssä tulen muuttamaan OpenMDV-ohjelman käyttöliittymän ulkoasua selkeämpään ja visuaalisesti näyttävämpään muotoon sekä pyrin selkeyttämään aiemmin tehtyjen XSL-muunnosten koodia.

Lisäksi työssä pyritään selkeiden esimerkkien avulla esittämään, miten XSL-muunnokset toimivat, miten niillä voidaan XPathin avulla operoida XML-solmujen välillä ja miten niitä niillä voidaan muokata XML:ää**.** Kuten tullaan huomaamaan, XSL-muunnokset sisältävät valtavasti erilaisia mahdollisuuksia muokata dokumentteja eri muotoihin. Kaikkia eri mahdollisuuksia ei pystytä tässä insinöörityössä käsittelemään, mutta pyrin kuitenkin kokoamaan mahdollisimman tehokkaan ja laajan oppaan muunnosten käyttämisestä.

2.1 Käytetyt ohjelmat

Insinöörityön OpenMDV:n muokkauksen toteutusosuudessa tulen käyttämään käyttöjärjestelmänä Linuxin Redhattia (versio 7.2), johon on asennettu Apache web-palvelin, MySQL-tietokanta sekä jBuilder (versio 5.0), jossa käytössä Apache Cocoon. Esimerkkien teossa ja niiden esittämisessä tulen käyttämään Windows-käyttöjärjestelmää ja XSL-muunnoksien suoritukseen jBuilderia.

Apache

Apache on nykyään ylivoimaisesti käytetyin web-palvelin. Se hallitsee palvelinmarkkinoita noin 60 prosentin markkinaosuudella. Apache on erittäin nopea ja luotettava palvelin, jonka suuren yleistymisen on mahdollistanut ainakin se, että ohjelman on täysin ilmainen ja kaikkille avoin lähdekoodi. /6/

MySQL

MySQL-tietokanta on yksi käytetyimmistä tietokantaohjelmista. Apachen tapaan se on avoimeen lähdekoodiin perustuva ohjelmisto, joten sen voi imuroida ilmaiseksi Internetistä. /7/

Apache Cocoon

Apache Cocoon on XML-dokumenttien julkaisemiseen käytetty julkaisualusta. Cocoon hoitaa palvelimella tapahtuvan XML-dokumenttien ja sen mahdollisesti sisältämän Java-koodin julkaisun tarjoten mahdollisuuden tiedon erottamiseen sisällön, logiikan ja tyylin välillä. /9/

3. TOTEUTUS

XSLT:tä käytetään käytetään dokumenttien käsittelyyn, niiden merkkauksen muuntamiseen sekä työskentelyyn niiden kanssa halutulla tavalla. Eräs yleisimmistä muunnoksista on XML-dokumenttien muuntaminen HTML-dokumenteiksi. Muunnoksen tekemiseen tarvitaan kaksi dokumenttia, muunnettava dokumentti ja muunnoksen määrittävä XSL-tiedosto.

3.1 XSLT-esimerkkejä

XML-tiedosto oppilaat.xml, josta esitän useita eri muunnoksia käyttäen XSLT:tä, näyttää seuraavalta.

```
<?xml version=”1.0” encoding=”ISO-8859-1”?>
<?xml-stylesheet type=”text/xsl” href=”oppilaat.xsl”?>
<OPPILAAT>

        <OPPILAS KOULU=”LUKIO”>
                <NIMI>Tomi Saari</NIMI>
                <PITUUS YKSIKKÖ=”cm”>195</PITUUS>
                <PAINO YKSIKKÖ=”kg”>90</PAINO>
                <IKA YKSIKKÖ=”vuotta”>17</IKA>
                <SIJAINTI>Raahe</SIJAINTI>
        </OPPILAS>

        <OPPILAS KOULU=”YLIOPISTO”>
                <NIMI>Mika Isometsa</NIMI>
                <PITUUS YKSIKKÖ=”cm”>180</PITUUS>
                <PAINO YKSIKKÖ=” kg”>70</PAINO>
                <IKA YKSIKKÖ=”vuotta”>25</IKA>
                <SIJAINTI>Sompajoki</SIJAINTI>
                <SIJAINTI>Kukkola</SIJAINTI>
        </OPPILAS>

        <OPPILAS KOULU=”AMMATTIKORKEAKOULU”>
```

```
                    <NIMI>Niina Varis</NIMI>
                    <PITUUS YKSIKKÖ=”cm”>160</PITUUS>
                    <PAINO YKSIKKÖ=”kg”>59</PAINO>
                    <IKA YKSIKKÖ=”vuotta”>21</IKA>
                    <SIJAINTI>ylivieska</SIJAINTI>
          </OPPILAS>
</OPPILAAT>
```

<?xml version=”1.0” encoding=”ISO-8859-1”?> on XML:n prosessointikäsky, joka osoittaa, että käytetään XML-versiota 1.0 ja että käytetään ISO-8859-1 –merkkikoodausta, jolloin käytössä ovat myös skandinaaviset merkit (ä,ö ja å).

<?xml-stylesheet type=”text/xsl” href=”oppilaat.xsl”?> -prosessointiohjeella kerrotaan, mitä tehdään. *Xml-stylesheet* ilmoittaa, että käytetään XSLT-tyylitiedostoa. Prosessointiohjeessa oleva type-attribuutti asetetaan text/xsl ja href-attribuutin arvoksi XSLT-tyylitiedoston URI, kuten tässä *oppilaat.xsl.* Kuten huomataan, XSLT-tyylitiedostojen tarkennin on yleensä xsl.

Elementti OPPILAAT on oppilaat.xml-tiedoston juurielementti. Sen lapsia ovat OPPILAS-elementit joiden lapsia taas ovat NIMI, PITUUS, PAINO, IKA ja SIJAINTI, joiden sisältä löytyvät niiden arvot. YKSIKKÖ ja KOULU ovat oppilaat.xml-tiedoston attribuutteja joissa, määritellään arvot cm, kg ja vuotta.

3.1.1 XSLT - Tiedon näyttäminen ja elementtien paikallistaminen

Seuraavissa esimerkeissä olevat muunnokset ovat tiedostoja nimeltä oppilaat.xsl. Seuraavassa yksinkertainen esimerkki millä oppilaat.xml muunnetaan html-muotoon ja siitä näytetään oppilaiden nimet.

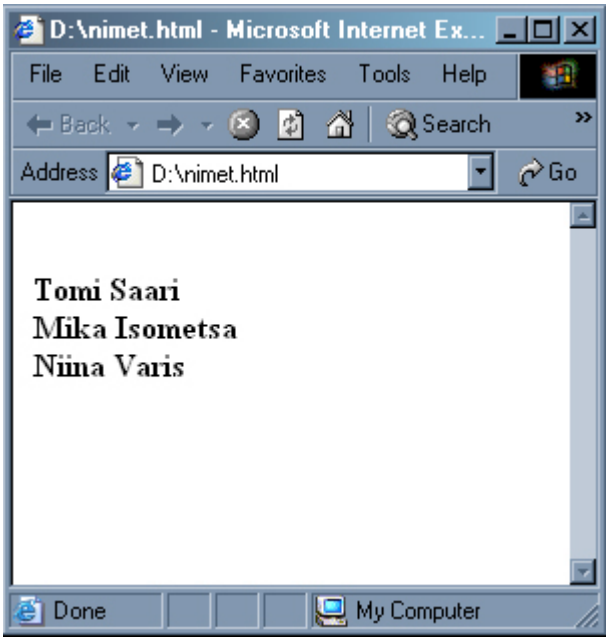
```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="OPPILAAT">
      <HTML>
         <xsl:apply-templates/>
```

```
      </HTML>
   </xsl:template>
   <xsl:template match="OPPILAS">
      <B>
         <br /> <xsl:value-of select="NIMI"/>
      </B>
   </xsl:template>
</xsl:stylesheet>
```

<xsl:apply-templates/>-elementillä suoritetaan muut tulevat elementit <HTML>-tagien sisään. *<xsl:template match="OPPILAS"*>-elementillä paikallistetaan OPPILAAT solmun OPPILAS-lapset. *<xsl:value-of select="NIMI"/>*-elementin avulla haetaan solmuarvot ja tulostetaan ne HTML-tagin <B>, eli asetetaan teksti lihavoiduksi, sisään. oppilaat.xsl avulla muunnettu XML-tiedosto muuntuu HTML-tiedostoksi jonka koodi on:

```
<HTML>
   <B> Tomi Saari </B>
   <B> Mika Isometsä </B>
   <B> Niina Varis </B>
</HTML>
```

Se näyttää selaimessa seuraavalta. Kuva 2.:



Kuva 2. Oppilaat listataan HTML-dokumenttiin

Mikäli haettu elementti olisi ollut <SIJAINTI>, *select*-attribuutin käyttö olisi ollut vajavainen. Se valitsee ainoastaan ensimmäisen solmun, joka vastaa sen valintakriteeriä. Ratkaisuna <SIJAINTI>-elementin tulostukseen voidaan käyttää <*xsl:for-each*>-elementtiä.

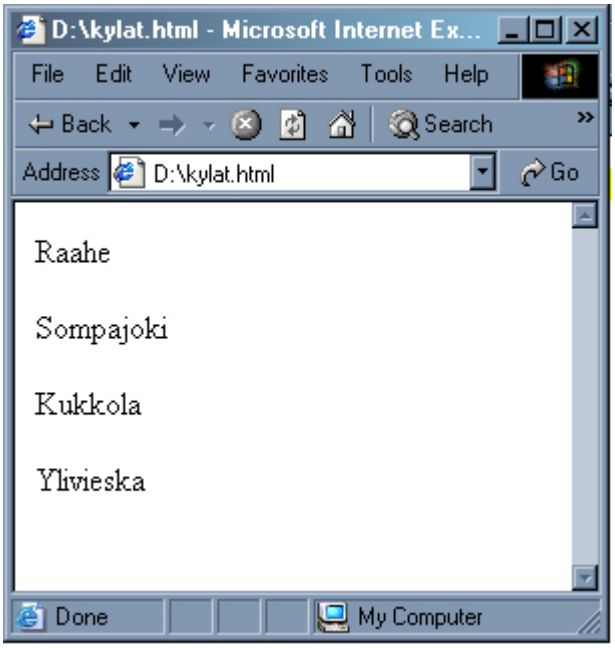
Tällöin käytettävä oppilaat.xsl olisi seuraava:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="OPPILAAT">
      <HTML>
         <xsl:apply-templates/>
      </HTML>
   </xsl:template>
<xsl:template match="OPPILAS">
   <xsl:for-each select="SIJAINTI">
      <p>
         <xsl:value-of select="."/>
       </p>
   </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Tällöin syntyvä html-koodi on:

```
<HTML>
   <p>Raahe</p>
   <p>Sompajoki</p>
   <p>Kukkola</p>
   <p>Ylivieska</p>
</HTML>
```

Koodi näyttää selaimessa seuraavalta (kuva 3.):



Kuva 3. Myös toinen kaupunki listataan HTML-dokumenttiin

<xsl:for-each>-elementillä voidaan siis käyttää kaikkien sopivuuksien silmukoimiseen.

Elementtinimet erotetaan toisistaan /-operaattorilla, jonka avulla voidaan myös tarvittaessa viitata jonkin solmun lapseen. Jos esimerkiksi tarvitaan sääntö, joka pätee ainoastaan <NIMI>-elementteihin, jotka ovat <OPPILAS>-elementtien lapsia, tarvitaan seuraava lauseke ”OPPILAS/NIMI”.

Esimerkiksi

```
<xsl:template match=”OPPILAS/NIMI”>
   <B><xsl:value-of select=”.” /> </B>
</xsl:template>
```

Tämä sääntö ympäröi <NIMI>-elementtien tekstin <B>-elementissä. ”.” lauseketta käytetään select-attribuutin kanssa, että päästään määrittämään kyseinen solmu.

Jälkeläisten paikallistaminen onnistuu samalla tavalla, esim. <NIMI>-elementin perusteella. Lauseke ”OPPILAS/*/NIMI” paikallistaisi kaikki <OPPILAS>-elementin lastenlapset, mikäli niitä olisi, kun taas lauseke ”OPPILAS//NIMI” paikallistaa kaikki <NIMI>-elementit, jotka ovat <OPPILAS>-elementin sisällä.

3.1.2 Attribuuttien paikallistaminen

Oppilaat.xml-tiedostossa painon, pituuden ja iän yksiköt on tallennettu elementtien attribuuteiksi. Näiden paikallistaminen ja näyttäminen tapahtuu @-etuliitteen avulla. Esimerkiksi painon yksikön näyttäminen voidaan tehdä seuraavasti:

```
<xsl:template match="PAINO">
   <xsl:value-of select="."/>
   <xsl:value-of select="@YKSIKKÖ"/>
</xsl:template>
```

Kuten jo edellä käytettiin, *-merkkiä, voidaan sitä käyttää myös attribuuttien paikallistamisessa. @*-yleismerkillä on mahdollista valita kaikki elementin attribuutit. Esimerkiksi ”OPPILAS/@*” valitsee kaikki attribuutit, jotka sijaitsevat OPPILAS-elementtien sisällä.

3.1.2.1 ID:n avulla paikallistaminen

Elementtejä voidaan paikallistaa myös ennalta annetun ID-arvon perusteella. Elementti paikallistetaan id()-hahmon avulla. Että id()-selektoria voitaisiin käyttää, täytyy elementeille antaa ID-attribuutti ja se täytyy esitellä ID-tyyppiseksi DTD-määrittelyssä. Esimerkki

```
<xsl:template match = "id('LisaaID')">
   <H3><xsl:value-of select="."/></H3>
</xsl:template>
```

lisää niiden elementtien tekstisisällön dokumenttiin, joiden ID on LisaaID.

3.1.3 Operaattorit

Kun dokumenteista tulee suurempia ja monimutkaisempia tulee operaattorien käyttäminen tarpeelliseksi. Niiden avulla voidaan tehdä valintoja paikallistettavista elementeistä, etsiä attribuutin arvo annetussa merkkijonossa, testata elementin arvo, testata, sisältääkö jokin elementti tietyn lapsen, attribuutin tai toisen elementin sekä testata solmun sijainti solmupuussa.

TAI-operaattori merkitään pystyviivalla (|). Esimerkiksi elementtien <NIMI> ja <PAINO> näyttämiseksi oppilaat.xml-tiedostosta voidaan käyttää seuraavaa koodia:

```
<xsl:template match="NIMI | MASSA">
   <P>
      <xsl:apply templates />
   </P>
</xsl:template>
```

Ehto-operaattorilla testataan, onko jokin ehto tosi. Se merkitään [ ] ja sen käytöstä seuraavassa muutama esimerkki:

- Lauseke, joka vastaa <OPPILAS>-elementtejä, joilla on lapsena elementti <NIMI>: <xsl:template match = ”OPPILAS[NIMI]”>
- Lauseke, joka vastaa mitä tahansa elementtiä, jonka lapsielementtinä on <NIMI>: <xsl:template match = ”*[NIMI]”>
- Lauseke, joka vastaa mitä tahansa <OPPILAS>-elementtiä, jolla on joko <NIMI>- tai <PAINO>-lapsielementti:
- <xsl:template match = ”OPPILAS[NIMI | PAINO]”>
- <OPPILAS>-elementin KOULU-attribuutti voidaan voidaan valita ehto-operaattorilla seuraavasti: <xsl:template match = ”OPPILAS[@KOULU]”> ja jos halutaan määrittää valittava koulu,
  <xsl:template match = ”OPPILAS[@KOULU=’AMMATTIKORKEAKOULU’]”>

3.1.4 Testiarvojen vertaaminen

<*xsl:choose*>-elementillä päästään vertailemaan testiarvoa usean mahdollisen vastaavuuden välillä. <*xsl:choose*>-elementin testi määritellään test-attribuutilla, jonka jälkeen jokainen etsittävä tapaus on määritettävä <*xsl:when*>-elementillä.

Seuraavassa tulostan oppilaat.xml-dokumentista html-koodin, joka esittää koulut ja oppilaat, sekä esittää ne eri kokoisilla, tekstillä jotka on määritelty KOULU-attribuutin mukaan. Mikäli jotain koulua ei ole mainittu, muunnos tulostaa kaikki oppilaan tiedot <pre>-elementin sisään. <*xsl:text*>-elementtiä käytetään tässä tilanteessa välilyönnin aikaansaamiseksi. XSL-muunnos näyttää seuraavalta.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="OPPILAAT">
  <HTML>
    <HEAD>
      <TITLE>
        Oppilaat kouluittain
      </TITLE>
```

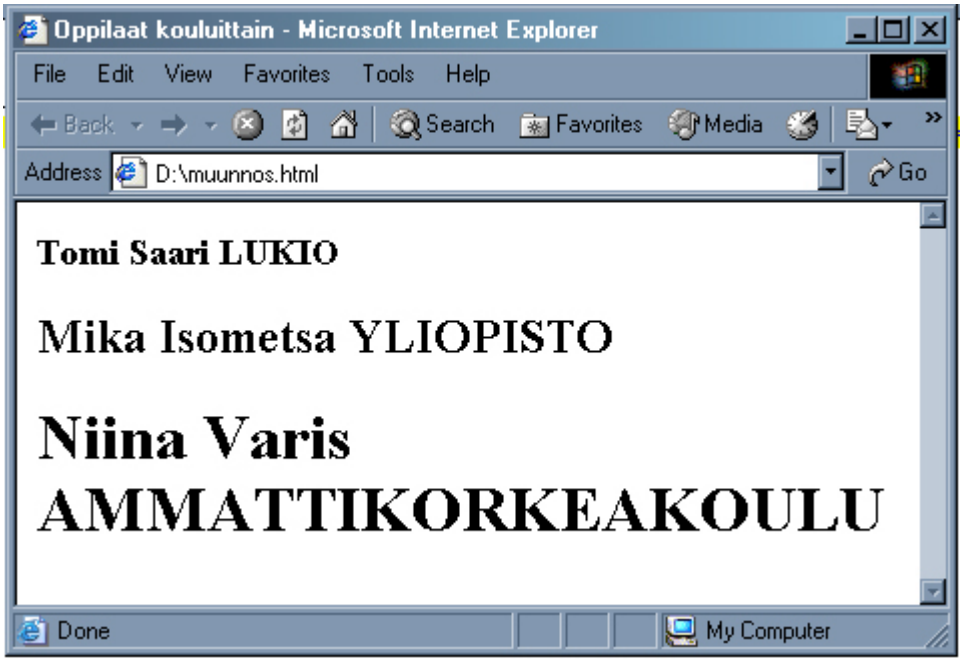
```
        </HEAD>
        <BODY>
           <xsl:apply-templates select="OPPILAS"/>
        </BODY>
     </HTML>
</xsl:template>

<xsl:template match="OPPILAS">
    <xsl:choose>
        <xsl:when test="@KOULU = 'AMMATTIKORKEAKOULU'">
           <h1>
              <xsl:value-of select="NIMI"/>
              <xsl:text> </xsl:text>
              <xsl:value-of select="@KOULU"/>
           </h1>
        </xsl:when>
        <xsl:when test="@KOULU = 'YLIOPISTO'">
           <h2>
              <xsl:value-of select="NIMI"/>
              <xsl:text> </xsl:text>
              <xsl:value-of select="@KOULU"/>
           </h2>
        </xsl:when>
        <xsl:when test="@KOULU = 'LUKIO'">
           <h3>
              <xsl:value-of select="NIMI"/>
              <xsl:text> </xsl:text>
              <xsl:value-of select="@KOULU"/>
           </h3>
        </xsl:when>
        <xsl:otherwise>
            <PRE>
               <xsl:value-of select="."/>
            </PRE>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

```
</xsl:stylesheet>
```

Syntyvä html-koodi on:

```
<?xml version="1.0" encoding="UTF-8"?>
<HTML>
   <HEAD>
      <TITLE> Oppilaat kouluittain</TITLE>
   </HEAD>
   <BODY>
      <h3>Tomi Saari LUKIO</h3>
      <h2>Mika Isometsa YLIOPISTO</h2>
      <h1>Niina Varis AMMATTIKORKEAKOULU</h1>
  </BODY>
</HTML>
```

html-koodi näyttää selamessa seuraavalta (Kuva 4.):



Kuva 4. HTML-dokumenttiin listataan oppilaat sekä heidän koulunsa eri tekstikoolla.

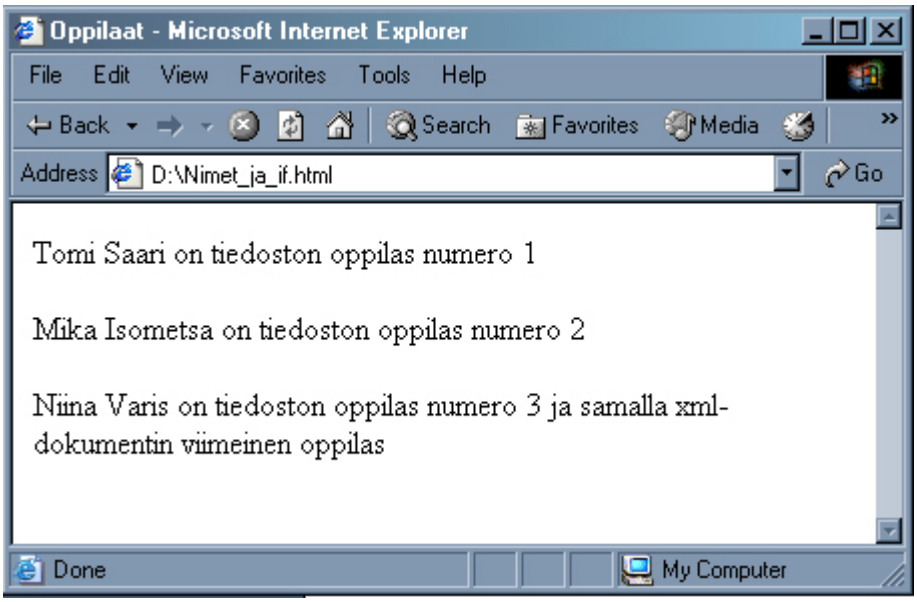
Tehtäessä valintoja, jotka perustuvat lähdedokumenttiin, on *<xsl:if>*-elementti hyödyllinen apuväline. Elementin käyttämiseksi asetetaan test-attribuutin arvoksi lauseke, joka arvioi boolen arvoa. Esimerkissä luodaan HTML-dokumentti, jossa asetetaan näytölle kaikkien oppilaiden nimet ja viimeisen kohdalla ilmoitetaan, että tämä on XML-dokumentin viimeinen oppilas.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="OPPILAAT">
   <HTML>
      <HEAD>
         <TITLE>
            Oppilaat
         </TITLE>
      </HEAD>
      <BODY>
         <xsl:apply-templates select="OPPILAS"/>
      </BODY>
   </HTML>
</xsl:template>

<xsl:template match="OPPILAS">
   <P>
   <xsl:value-of select="NIMI"/>
   on tiedoston oppilas numero <xsl:value-of select="position()"/>
    <xsl:if test="position() = last()">
    <xsl:text> </xsl:text>ja samalla xml-dokumentin viimeinen oppilas
   </xsl:if>
   </P>
</xsl:template>

</xsl:stylesheet>
```

Syntyvä html-koodi on:

```
<?xml version="1.0" encoding="UTF-8"?>
<HTML>
<HEAD>
<TITLE> Oppilaat</TITLE>
</HEAD>
 <BODY>
  <P>Tomi Saari on tiedoston oppilas numero 1</P>
  <P>Mika Isometsa on tiedoston oppilas numero 2</P>
  <P>Niina Varis on tiedoston oppilas numero 3 ja samalla xml-dokumentin viimeinen oppilas</P>
 </BODY>
</HTML>
```

Koodi näyttää selaimessa seuraavalta (kuva 5.):



Kuva 5. Oppilaat listataan ja kerrotaan heidän sijaintinsa lähdedokumentissa, sekä ilmoitetaan, kuka on dokumentin viimeinen oppilas.

3.2 XML:n rakenteen muuttaminen

Tähän mennessä käytetyissä muunnoksissa on keskitytty ainoastaan muuntamaan valmiita dokumentteja määrittäen, missä järjestyksessä ja mitä kohdedokumenttiin tulisi tulla. XSL-muunnokset tarjoavat kuitenkin eri tapoja esimerkiksi uusien solmujen luomiseen tai solmuarvojen vaihtamisen attribuuteiksi kohdedokumenttiin, joka voi olla niin HTML-, XML- tai tekstidokumentti. Kohdedokumentin rakenteen muuttamisessa pätevät samat säännöt dokumenttipuussa navigoitaessa kuin muissakin muunnoksissa.

3.2.1 Elementtien luominen

XML-dokumentin rakennetta saatetaan joutua muuttamaan lennosta. Esimerkiksi uuden elementin luominen tapahtuu *<xsl:element>*-elementin avulla. Seuraavassa esimerkissä luon uuden XML-tiedoston oppilaat.xml dokumentin pohjalta. Tiedosto ottaa alkuperäisestä tiedostosta koulu-attribuutin arvon uudeksi elementiksi ja asettaa oppilaat omiin kouluihinsa.

oppilaat.xml asetetaan kutsumaan muunnos.xsl-tiedostoa, joka näyttää seuraavalta:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="yes"/>

<xsl:template match="OPPILAAT">
<KOULUT>
   <xsl:apply-templates select="OPPILAS"/>
</KOULUT>
</xsl:template>

<xsl:template match="OPPILAS">
   <xsl:element name="{@KOULU}">
      <NIMI><xsl:value-of select="NIMI"/></NIMI>
   </xsl:element>
```

```
</xsl:template>

</xsl:stylesheet>
```

Muunnoksella muodostuva uusi XML-dokumentti näyttää seuraavalta:

```
<?xml version="1.0" encoding="UTF-8"?>
<KOULUT>
   <LUKIO>
      <NIMI>Matti meikäläinen</NIMI>
   </LUKIO>
   <YLIOPISTO>
      <NIMI>Mika Isometsa</NIMI>
   </YLIOPISTO>
   <AMMATTIKORKEAKOULU>
      <NIMI>Niina Saari</NIMI>
   </AMMATTIKORKEAKOULU>
</KOULUT>
```

3.2.2 Attribuuttien luominen ja tiedon lajitteleminen

Uusien attribuuttien luominen tapahtuu *<xsl:attribute>*-elementillä. Jos esimerkiksi haluaisimme näyttää oppilaat.xml-dokumentista ainoastaan oppilaan nimen ja sijainnin joiden arvot on sijoitettu attribuutteihin, voisimme asettaa muunnokselle myös lisävaatimuksen, jonka mukaan nimet tulee näyttää aakkosjärjestyksessä. Tällainen XSL-muunnos ja sen lopputulos näyttäisi seuraavalta.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="yes"/>

<xsl:template match="/">
  <OPPILAAT>
     <xsl:call-template name="jarjesta"/>
  </OPPILAAT>
```

```
</xsl:template>

<xsl:template name="jarjesta">
  <xsl:for-each select="/OPPILAAT/OPPILAS">
    <xsl:sort select="NIMI"/>
    <xsl:call-template name="muodosta_attribuutit"/>
  </xsl:for-each>
</xsl:template>

<xsl:template name="muodosta_attribuutit">
  <OPPILAS>

    <xsl:for-each select="NIMI">
     <NIMI>
       <xsl:attribute name="NIMI_ATT"><xsl:value-of select="text()"/></xsl:attribute>
     </NIMI>
    </xsl:for-each>

     <xsl:for-each select="SIJAINTI">
      <SIJAINTI>
       <xsl:attribute name="SIJAINTI_ATT"><xsl:value-of select="text()"/></xsl:attribute>
      </SIJAINTI>
     </xsl:for-each>

  </OPPILAS>
</xsl:template>
```

<xsl:template match="/">-elementti luo uudelle XML-tiedostolle rungon jonka päälle uutta dokumenttia aletaan rakentaa. *<xsl:call-template name="jarjesta"/>*-elementillä kutsutaan järjestä-elementtiä jossa *<xsl:sort select="NIMI"/>*-elementti järjestää tulevat elementit aakkosjärjestykseen nimen mukaan. *<xsl:attribute name="NIMI_ATT">*-elementillä muodostetaan uusi attribuutti NIMI-elementtiin ja *<xsl:value-of select="text()"/>*-elementillä kopioidaan NIMI-solmun teksti attribuutin arvoksi. Kun sama toimenpide on suoritettu, SIJAINTI-solmulle muodostuu seuraava XML-dokumentti:

```
<?xml version="1.0" encoding="UTF-8"?>
<OPPILAAT>
   <OPPILAS>
      <NIMI NIMI_ATT="Mika Isometsa"/>
      <SIJAINTI SIJAINTI_ATT="Sompajoki"/>
      <SIJAINTI SIJAINTI_ATT="Kukkola"/>
   </OPPILAS>
   <OPPILAS>
      <NIMI NIMI_ATT="Niina Saari"/>
      <SIJAINTI SIJAINTI_ATT="ylivieska"/>
   </OPPILAS>
   <OPPILAS>
      <NIMI NIMI_ATT="Tomi Saari"/>
      <SIJAINTI SIJAINTI_ATT="raahe"/>
   </OPPILAS>
</OPPILAAT>
```

Kuten huomataan, järjestys on muuttunut nimien mukaiseen aakkosjärjestykseen.

3.2.3 Solmujen kopiointi

Solmujen kopiointi tapahtuu *<xsl:copy>*-elementillä, jolla määrätään tarkasti, mitkä osat halutaan kopioida. Ilman lisämäärityksiä elementtien oletussääntönä on, että vain elementin teksti kopioidaan. *<xsl:copy>*-elementilla voidaan kuitenkin muuttaa tämä ja täten mahdollistaa esimerkiksi elementtien, attribuuttien, tekstisolmujen ja prosessointiohjeiden kopiointi. Esimerkkitapauksessa kopioidaan ainoastaan tekstit ja elementit kohdedokumentin näkyvään osaan ja jätetään esimerkiksi attribuutit kokonaan näyttämättä.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="yes"/>

<xsl:template match="* | text()">
```

```
    <xsl:copy>
      <xsl:apply-templates select="* | text()"/>
    </xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

Muunnoksen tulosteeksi saadaan esimerkiksi oppilas Mika Isometsän kohdalla seuraava:

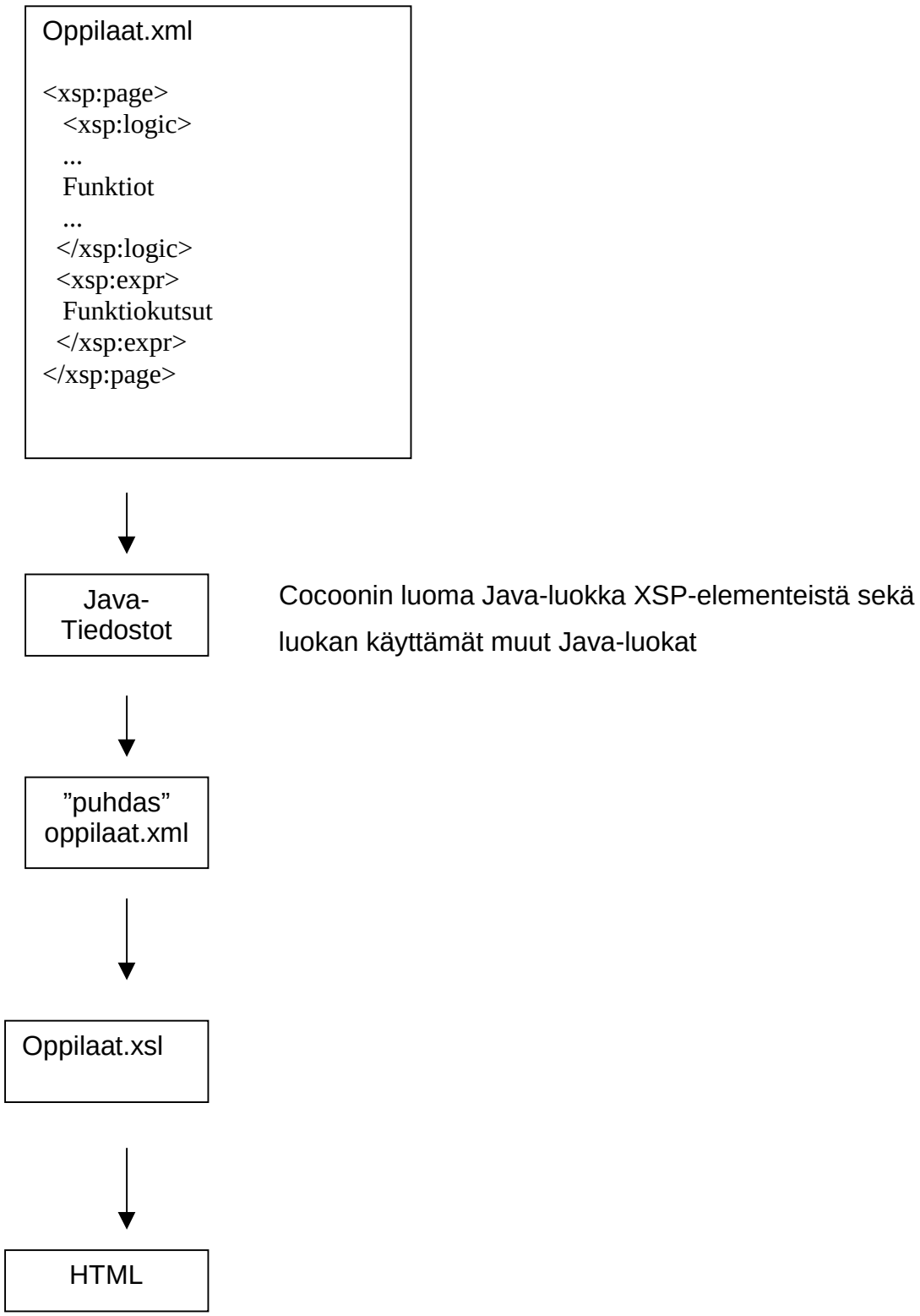
```
<OPPILAS>
    <NIMI>Mika Isometsa</NIMI>
    <PITUUS>180</PITUUS>
    <PAINO>70</PAINO>
    <IKA>25</IKA>
    <SIJAINTI>Sompajoki</SIJAINTI>
    <SIJAINTI>Kukkola</SIJAINTI>
</OPPILAS>
```

3.3 Muunnosprosessin suorittaminen Cocoonia apuna käyttäen

Tässä esimerkissä olen kirjoittanut yksinkertaisen sovelluksen, joka pyrkii jäljittelemään OpenMDV:tä. Esimerkki koostuu kolmesta osasta, joista ensimmäinen on OppilasTietokanta-Java-luokka. Luokka pyrkii simuloimaan tietokantaa, josta oppilaiden tiedot voi hakea, ja se sisältää samat tiedot, kuin aiemmin esitetyssä oppilaat.xml esimerkissä. Toinen osa on uudelleenkirjoitettu oppilaat.xml, jossa käytetään XSP:tä ja sen sisältämää Java-koodia hakemaan oppilaiden tiedot tietokannasta. Kolmas osa on XSL-muunnoksen dokumentti.

Oppilaat.xml-tiedoston juurielementtinä on <xsp:page>-elementti, jossa määritellään käytettävä kieliohjelmointikieli. Cocoon parsii tämän ja muiden XSP-elementtien sisällöt ja rakentaa niiden sisältämien tietojen perusteella väliaikaisen ohjelmakoodin, tässä tapauksessa Java-luokan. Cocoon suorittaa luomansa koodin ja lisää sen tuottaman tuloksen takaisin oppilaat.xml-tiedostoon luoden väliaikaisen XML-tiedoston. Koska oppilaat.xml-tiedostossa

on määritelty myös xslt-dokumentti, Cocoon suorittaa XSL-muunnoksen väliaikaiselle XML-dokumentille. Prosessi on esitetty kuvassa 6.



Kuva 6. Cocoonin suorittama prosessi

Esitän muodostuvan XML:n kopioimalla Cocoonin väliaikaiseen tiedostoon palauttamat puhtaan XML:n, elementit ja attribuutit seuraavan XSL-muunnoksen avulla:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/ | * | @*">
  <xsl:copy>
    <xsl:apply-templates select="* | @* | text()"/>
  </xsl:copy>
</xsl:template>
</xsl:stylesheet>
```

Tulostuva dokumentti näyttää seuraavalta:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-html40/strict.dtd">
<docroot>
<oppilaat>
<oppilas koulu="Lukio">
    <nimi>Tomi Saari</nimi>
    <pituus yksikko="cm">195</pituus>
    <paino yksikko="kg">90</paino>
    <ika yksikko="vuotta">17</ika>
    <sijainti>Raahe</sijainti>
</oppilas>
<oppilas koulu="Yliopisto">
    <nimi>Mika Isometsä</nimi>
    <pituus yksikko="cm">180</pituus>
    <paino yksikko="kg">70</paino>
    <ika yksikko="vuotta">25</ika>
    <sijainti>Sompajoki</sijainti>
    <sijainti>Kukkola</sijainti>
```

```
</oppilas>
<oppilas koulu="AMK">
   <nimi>Niina Varis</nimi>
   <pituus yksikko="cm">1160</pituus>
   <paino yksikko="kg">59</paino>
   <ika yksikko="vuotta">21</ika>
   <sijainti>Ylivieska</sijainti>
</oppilas>
</oppilaat>
</docroot>

<!-- This page was served in 567 milliseconds by Cocoon 1.8 -->
```

Java-koodit LIITE 1:ssä

3.4 OpenMDV

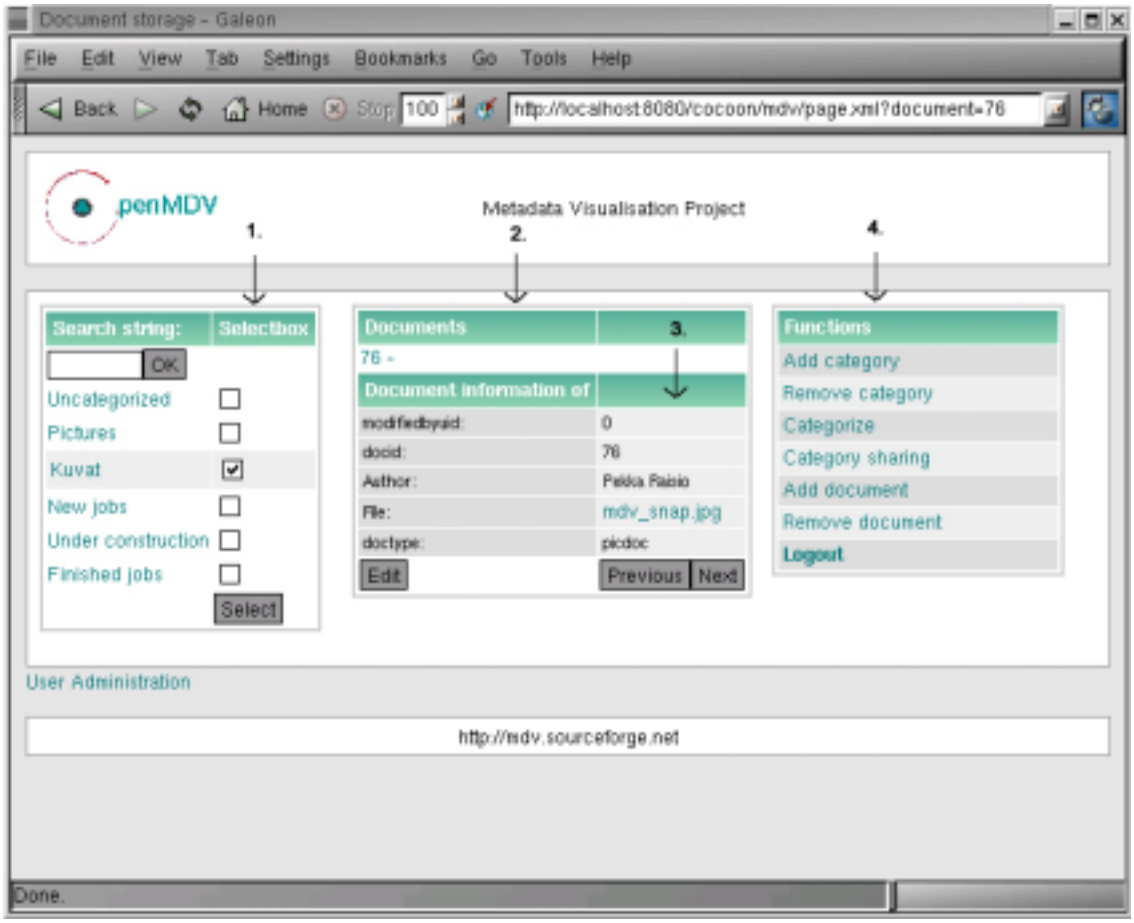
OpenMDV on webissä toimiva ohjelma, joka on rakennettu tiedon talletus- ja hallintaratkaisuksi. Tieto ja tiedostot jaetaan eri kategorioihin, jotka talletetaan MySQL-tietokantaan. OpenMDV:n varsinaiset palvelut on toteutettu Javalla, joiden tekemiseen allekirjoittanut ei tässä insinöörityössä ole keskittynyt.

Ohjelman suoritus lähtee XML-tiedostosta, johon on kirjoitettu sekä XML:ää että Javaa. Apache Cocoonin avulla suoritetaan Java-koodi ja tehdään mahdolliset tietokantakyselyt. Tästä saatujen tietojen perusteella Cocoon muodostaa lopullisen XML-tiedoston, joka kutsuu XSL-muunnosta, jonka avulla suoritetaan muuntaminen HTML:ksi ja lopputulos tulostetaan selaimelle. OpenMDV:n koodeista löytyy esimerkki liittestä 2.

3.3.1 Käyttöliittymä

Käyttöliittymän pääsivu on jaettu kolmeen osaan (kuva 7.):

1. Osa johon listataan tietokannasta löytyvät kategoriat. Mikäli kategorialista on laajempi, voidaan käyttää samasta osasta löytyvää hakua.
2. Kun jokin kategoria on valittu, OpenMDV listaa valitusta kategoriasta löytyvät dokumentit.
3. Kun jokin dokumenteista valitaan, ohjelma listaa dokumentille annetut tiedot ja antaa mahdollisuuden esim. kuvatiedoston katselemiseen tai tietojen muuttamiseen.
4. Tässä osassa voidaan lisätä ja poistaa niin kategorioita kuin niiden dokumenttejakin. Samoin voidaan jakaa kategorioita muiden käyttäjien nähtäville ja yhdistää dokumentteja muihin kategorioihin.



Kuva 6. OpenMDV:n pääsivu

4. YHTEENVETO JA LOPPUPÄÄTELMÄT

Tässä työssä olen esittänyt XML:n muuntamista ja muokkaamista, useilla eri tavoilla, käyttäen XSL-muunnoksia. Esimerkit on tehty opiskelemalla mahdollisimman useasta lähteestä saatuja tietoja ja pyritty painottamaan olennaisimpia seikkoja, mitä XML:n muuntamiseen liittyy.

XML:n rakenteen muuntaminen onnistuu muillakin ohjelmointikielellä, kuten esimerkiksi Javalla. Kuitenkin tähänastisten kokemusten perusteella XSL-muunnokset XPathia apuna käyttäen ovat allekirjoittaneen mielestä ylivoimaisesti kätevin tapa muuntaa suorittaa muunnokset.

Työn tekijän pohjatiedot ennen tämän insinöörityön tekemistä olivat suhteellisen pienet. Uuden oppimisen kannalta työn tekeminen todella antoisaa. Tekijän osaamisen taso kasvoi niin työvälineiden kuin ohjelmointikielien kannalta.

Työn puutteena voidaan pitää ohjeiden riittämättömyyttä. Tämän työn pohjalta ei voi alkaa toteuttaa valmista järjestelmää tutustumatta laajemmin esimerkiksi Javaan, mutta ainakin työ antaa tukevan pohjan XSL-muunnoksiin ja niiden peruskäyttöön. Toivottavasti esimerkeissä annetut ohjeet ja vinkit tarpeeseen.

LÄHTEET

/1/ Linjama, T.: XHTML, docendo Finland Oy, Jyväskylä, 2001

/2/ <URL: http://www.w3.org> (4.4.2003)

/3/ <URL:http://www.cs.helsinki.fi/u/ruini/structure/xsl/> (12.3.2003)

/4/ Holzner, S.: INSIDE XML, Edita Oyj, Jyväskylä, 2001

/5/ Kay, M.: XSLT 2nd edition, Wrox Press Ltd, UK, 2001

/6/ Hunter, D.: Beginning XML, Wrox Press Ltd, UK, 2000

/7/ <URL :http://www.apache.org> (4.4.2003)

/8/ <URL:http://www.mysql.com/products/mysql/index.html>(4.4.2003)

/9/ <URL :http://xml.apache.org/cocoon/> (4.4.2003)

/10/ <URL :http://www.cs.uta.fi/reports/pdf/A-2002-12.pdf> (4.4.2003)

LIITTEET

LIITE 1 Cocoonin käyttäminen ja tietojen hakeminen.

LIITE 2 OpenMDV Esimerkki

LIITE 1

oppilaat.xml

```
<?xml version="1.0"?>
<?cocoon-process type="xsp"?>
<?cocoon-process type="xslt"?>

<?xml-stylesheet href="page-test.xsl" type="text/xsl"?>

<xsp:page language="java" xmlns:util="http://www.apache.org/1999/XSP/Util"
        xmlns:xsp="http://www.apache.org/1999/XSP/Core">

  <xsp:structure>
    <xsp:include>fi.ratol.jkuoppal.cocoon_esim.OppilasTietokanta</xsp:include>
    <xsp:include>fi.ratol.jkuoppal.cocoon_esim.Oppilas</xsp:include>
  </xsp:structure>

  <docroot>
    <xsp:logic><![CDATA[

      //Luodaan tietokanta-olio
      OppilasTietokanta tietokanta = new OppilasTietokanta();

      //Haetaan oppilaat tietokannasta
      Oppilas[] oppilaat = tietokanta.haeOppilaat();

      //Kirjoitetaan oppilaiden tiedot StringBufferiin XML-muodossa
      StringBuffer output = new StringBuffer();
      output.append("<oppilaat>\n");
      for (int i = 0; i < oppilaat.length; i++) {
        output.append("<oppilas koulu=\"" + oppilaat[i].getKoulu() + "\">\n");
        output.append("    <nimi>" + oppilaat[i].getNimi() + "</nimi>\n");
        output.append("    <pituus yksikko=\"cm\">" + oppilaat[i].getPituus() + "</pituus>\n");
        output.append("    <paino yksikko=\"kg\">" + oppilaat[i].getPaino() + "</paino>\n");
        output.append("    <ika yksikko=\"vuotta\">" + oppilaat[i].getIka() + "</ika>\n");

        String[] sijainnit = oppilaat[i].getSijainnit();
        for (int s = 0; s < sijainnit.length; s++) {
```

```
        output.append("    <sijainti>" + sijainnit[s] + "</sijainti>\n");
      }
      output.append("</oppilas>\n");
    }
    output.append("</oppilaat>\n");

    //Tulostetaan konsoliin testausta varten
    System.out.println("########################################");
    System.out.println("OUTPUT: "+output.toString());
    System.out.println("########################################");
    ]]>
  </xsp:logic>

  <xsp:expr>
    this.xspParser.parse(
      new InputSource(
        new StringReader(
          output.toString()
        )
      )
    ).getDocumentElement()
  </xsp:expr>
 </docroot>
</xsp:page>
```

OppilasTietokanta.java

```
package fi.ratol.jkuoppal.cocoon_esim;


public class OppilasTietokanta {

 //MUISTA! Kopioi käännetyt hakemistoon
 // /home/jkuoppal/mdv/openmdv/tomcat/inst/webapps/openmdv/WEB-INF/classes

 public OppilasTietokanta() {

 }

 public Oppilas[] haeOppilaat() {
  Oppilas[] oppilaat = new Oppilas[3];
  oppilaat[0] = new Oppilas("Lukio", "Tomi Saari", "195", "90", "17", new String[] {"Raahe"});
  oppilaat[1] = new Oppilas("Yliopisto", "Mika Isometsä", "180", "70", "25", new String[] {"Sompajoki", "Kukkola"});
  oppilaat[2] = new Oppilas("AMK", "Niina Varis", "160", "59", "21", new String[] {"Ylivieska"});

  return oppilaat;
```

```
  }

}
```

Oppilas.java

```
package fi.ratol.jkuoppal.cocoon_esim;

/**
 * @author Juho Kuoppala
 * @version 1.0
 */

public class Oppilas {

  private String koulu;
  private String nimi;
  private String pituus;
  private String paino;
  private String ika;
  private String[] sijainnit;

  public Oppilas(String koulu, String nimi, String pituus, String paino, String ika, String[] sijainnit) {
    this.koulu = koulu;
    this.nimi = nimi;
    this.pituus = pituus;
    this.paino = paino;
    this.ika = ika;
    this.sijainnit = sijainnit;
  }

  public String getKoulu() {
    return koulu;
  }

  public String getNimi() {
    return nimi;
  }

  public String getPituus() {
    return pituus;
  }

  public String getPaino() {
    return paino;
  }

  public String getIka() {
    return ika;
  }

  public String[] getSijainnit() {
    return sijainnit;
  }

  public String toString() {
    return "Oppilaan tiedot: " + koulu + ", " + nimi + ", " + pituus + ", " + paino + ", " + ika;
  }}
```

LIITE 2 OpenMDV Esimerkki

Seuraavat koodit ovat OpenMDV:n Personize sivusta. Ensimmäinen Personize.xml on koodi jota Cocoon lähtee suorittamaan ja seuraava koodi on Cocoonin suorittamien oimenpiteiden jälkeen syntynyt väliaikainen tiedosto, josta muunnos tehdään en ole keskittynyt OpenMDV:n Java-puoleen joten XML-dokumentit Personize.xml:t ovat täysin muiden tekemiä.

OpenMDV ohjelmasta esimerkkikoodi.

Personize.xml

```
<?xml version="1.0"?>
<?cocoon-process type="xsp"?>
<?cocoon-process type="xslt"?>

<!--<?xml-stylesheet href="personize-html_test.xsl" type="text/xsl"?>-->
<?xml-stylesheet href="page-test.xsl" type="text/xsl"?>
<!-- Copyright (c) 2001, Helsinki Institute of Physics
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.  Redistributions
in binary form must reproduce the above copyright notice, this list of
conditions and the following disclaimer in the documentation and/or
other materials provided with the distribution.  Neither the name of
the Helsinki Institute of Physics nor the names of its contributors
may be used to endorse or promote products derived from this software
without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. -->


<!--Add a new document to multiple categories-->

<xsp:page language="java" xmlns:util="http://www.apache.org/1999/XSP/Util"
      xmlns:xsp="http://www.apache.org/1999/XSP/Core">
```

```
<xsp:structure>
  <xsp:include>fi.hip.mdv.servlets.PersonHandlingTranslator</xsp:include>
  <xsp:include>fi.hip.mdv.servlets.CategoryHandlingTranslator</xsp:include>
  <xsp:include>fi.hip.mdv.servlets.HttpSessionValueKeys</xsp:include>
  <xsp:include>fi.hip.mdv.servlets.RefererStack</xsp:include>
  <xsp:include>java.util.Hashtable</xsp:include>
  <xsp:include>java.util.Stack</xsp:include>
  <xsp:include>java.util.ArrayList</xsp:include>
  <xsp:include>java.lang.StringBuffer</xsp:include>
  <xsp:include>fi.hip.mdv.dataaccess.DataAccess</xsp:include>
  <xsp:include>fi.hip.mdv.dataaccess.Person</xsp:include>
  <xsp:include>fi.hip.mdv.dataaccess.Category</xsp:include>
</xsp:structure>

<page>
  <xsp:logic><![CDATA[

    Stack refererStack = RefererStack.getInstance(request);
    StringBuffer output = new StringBuffer();
    output.append("<result>");

    //see how we were called
    if ("true".equals(request.getParameter("confirmed"))) {
      output.append("<refresh url=\"");
      output.append("page.xml");
      output.append("\"/>");
    }
    DataAccess mydataaccess = ((DataAccess)
  request.getSession().getAttribute(HttpSessionValueKeys.DATAACCESS_OBJECT));
    if (mydataaccess != null && mydataaccess.isLoggedIn()) {
      //get the category sharing information from
      //the current category
      CategoryHandlingTranslator myCatTrans =
                    CategoryHandlingTranslator.getInstance(request);
      String catid = myCatTrans.getCategoryID();
      Category cat = mydataaccess.getCategory(catid);

      PersonHandlingTranslator myPersonTrans = PersonHandlingTranslator.getInstance(request);
      if (myPersonTrans != null && cat != null)  {

myPersonTrans.togglePersonSelection(request.getParameterValues(PersonHandlingTranslator.PERSON_PARAMETER));
        Person[] mypersons = myPersonTrans.getPersons(request.getSession());
        //System.out.println("This many persons :" +mypersons.length);
        output.append(cat.toXMLString());
        output.append("<personlist>");
        for (int i=0;i<mypersons.length;i++) {
          Person perSon = mypersons[i];
          if (perSon != null) {
           //System.out.println("PERSON: "+perSon.toXMLString());
           //now see if this person shares the category in question
           String perid = perSon.getID();
           if (cat.containsSharedWith(perid)) {
               System.out.println("selecting "+perid);
               perSon.setSelected(true);
           }
           output.append(perSon.toXMLString());
          }

        }
        output.append("</personlist>");
```

```
        output.append("<currentuser id=\"");
        output.append(mydataaccess.getCurrentUser().getID());
        output.append("\"/>");
      } else {
        System.out.println("PersonTranslator is null");
      }

    } else {
      //login not ok
      output.append("<refresh url=\"login.xml?action=login\"/>");
    }
    output.append("</result>");
    System.out.println("PERSONIZE-->");
    System.out.println(output.toString());
    System.out.println("<--PERSONIZE");
    ]]>
  </xsp:logic>

  <title>Sharing</title>

  <referer><xsp:expr>(String) refererStack.peek()</xsp:expr></referer>

  <xsp:expr>
    this.xspParser.parse(
      new InputSource(
        new StringReader(
          output.toString()
        )
      )
    ).getDocumentElement()
  </xsp:expr>

 </page>

</xsp:page>
```

Personize.xml cocoonin suoritettua Javan.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-html40/strict.dtd">
<page>
<title>Sharing</title>
<referer>page.xml</referer>
<result>
<category id="75" label="Kuvat" parent selected="0" type="normal">
  <metaitem name="owner" value="1"></metaitem>
</category>
<personlist>
  <person id="1" selected="0">
    <metaitem name="uid" value="1"></metaitem>
    <metaitem name="loginName" value="15"></metaitem>
    <metaitem name="sn" value="Raisio"></metaitem>
    <metaitem name="gn" value="Pekka"></metaitem>
    <metaitem name="email" value></metaitem>
    <metaitem name="dateOfBirth" value="null"></metaitem>
    <metaitem name="role" value="x"></metaitem>
  </person>
  <person id="34" selected="0">
```

```
    <metaitem name="uid" value="34"></metaitem>
    <metaitem name="loginName" value="jkuoppal"></metaitem>
    <metaitem name="sn" value="Kuoppala"></metaitem>
    <metaitem name="gn" value="Juho"></metaitem>
    <metaitem name="email" value="jkuoppal@ratol.fi"></metaitem>
    <metaitem name="dateOfBirth" value="null"></metaitem>
    <metaitem name="role" value="user"></metaitem>
  </person>
  <person id="1" selected="0">
    <metaitem name="uid" value="1"></metaitem>
    <metaitem name="loginName" value="jkuoppal"></metaitem>
    <metaitem name="sn" value="Kuoppala"></metaitem>
    <metaitem name="gn" value="Juho"></metaitem>
    <metaitem name="email" value="jkuoppal@hotmail.com"></metaitem>
    <metaitem name="dateOfBirth" value="null"></metaitem>
    <metaitem name="role" value="user"></metaitem>
  </person>
  <person id="1" selected="0">
    <metaitem name="uid" value="1"></metaitem>
    <metaitem name="loginName" value="15"></metaitem>
    <metaitem name="sn" value="Raisio"></metaitem>
    <metaitem name="gn" value="Pekka"></metaitem>
    <metaitem name="email" value></metaitem>
    <metaitem name="dateOfBirth" value="null"></metaitem>
    <metaitem name="role" value="user"></metaitem>
  </person>
  <person id="78" selected="0">
    <metaitem name="uid" value="78"></metaitem>
    <metaitem name="loginName" value="Panu"></metaitem>
    <metaitem name="sn" value="Testimies"></metaitem>
    <metaitem name="gn" value="Panu"></metaitem>
    <metaitem name="email" value="panu@testi.net"></metaitem>
    <metaitem name="dateOfBirth" value="null"></metaitem>
    <metaitem name="role" value="user"></metaitem>
  </person>
</personlist>
<currentuser id="1"></currentuser>
</result>
</page>
<!-- This page was served in 1688 milliseconds by Cocoon 1.8 -->
```

personize-html_test.xsl jolla xsl-muunnos suoritetaan

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="page">
    <xsl:processing-instruction name="cocoon-format">type="text/html"</xsl:processing-instruction>
    <html>
      <head>
        <xsl:if test="boolean(//refresh)">
          <meta http-equiv="refresh">
            <xsl:attribute name="content">
              <xsl:text>0; URL=</xsl:text>
              <xsl:value-of select="//refresh/@url"/>
            </xsl:attribute>
          </meta>
        </xsl:if>
        <title><xsl:value-of select="title"/></title>
```

```
    <link rel="stylesheet" type="text/css" href="resources/main.css"/>
   </head>
   <body>

  <div class="capsule">
   <table class="capsuletable" cellspacing="1">
    <tr>
     <td><img src="resources/mdvlogo.gif" widht="121" height="57" alt="OpenMDV" /> </td>
     <td> Metadata Visualisation Project </td>
    </tr>
   </table>
  </div>
  <br />
  <div class="capsule">
     <xsl:apply-templates select="//personlist"/>
  </div>
  <br />
  <div class="capsule" align="center">http://mdv.sourceforge.net</div>
   </body>
  </html>
</xsl:template>

<xsl:template match="personlist">
  <form action="servlet/PersonListHandlingServlet" method="post">
   <table class="maintable" cellpadding="4" cellspacing="0" border="0">
    <tr>
     <td colspan="2" class="headinglila">
      <xsl:value-of select="//title"/>
     </td>
    </tr>
    <tr>
     <td colspan="2"> Share with all users:
      <xsl:choose>
       <xsl:when test="//result/category/metaitem[@name='sharedwith']/@value ='0'">
        <input TYPE="checkbox" NAME="allshare" checked="true"/>
       </xsl:when>
       <xsl:otherwise>
         <input TYPE="checkbox" NAME="allshare"/>
       </xsl:otherwise>
      </xsl:choose>
     </td>
    </tr>
    <tr>
     <td>
      <!-- Dialog content -->
      <table cellpadding="0" cellspacing="0" border="0" width="100%">
       <tr>
        <td style="vertical-align:top; padding-top:10;">Select users to
            <br/>share with:
        </td>
        <td>
         <img src="resources/empty.gif" width="10"/>
        </td>
        <td>
         <select name="person" multiple="true" size="12">
          <!--Insert the list of categories -->
          <xsl:apply-templates select="person"/>
         </select>
        </td>
        <td>
```

```
            <img src="resources/empty.gif" width="10"/>
          </td>
        </tr>
      </table>
    </td>
    <td valign="top" align="right">
      <!-- Dialog controls -->
      <table cellpadding="0" cellspacing="0" border="0" width="100%">
        <tr>
          <td></td>
          <td>
            <center>
              <input type="submit" value=" OK "/>
            </center>
          </td>
        </tr>
        <tr>
          <td>
            <img src="resources/empty.gif" height="8"/>
          </td>
        </tr>
        <tr>
          <td></td>
          <td>
            <center>
              <input type="button" value="Cancel"
                  onclick="javascript:window.location.href='page.xml'"/>
            </center>
          </td>
        </tr>
        <tr>
          <td>
            <img src="resources/empty.gif" width="10"/>
          </td>
          <td></td>
          <td>
            <img src="resources/empty.gif" width="10"/>
          </td>
        </tr>
      </table>
    </td>
  </tr>
 </table>
</form>
</xsl:template>

<xsl:template match="person">
 <xsl:if test="//currentuser/@id != @id">
  <xsl:if test="@selected=1">
   <option value="{@id}" selected="true">
    <xsl:value-of select="metaitem[@name='gn']/@value"/> 
    <xsl:value-of select="metaitem[@name='sn']/@value"/>
   </option>
  </xsl:if>
  <xsl:if test="@selected=0">
   <option value="{@id}">
    <xsl:value-of select="metaitem[@name='gn']/@value"/> 
    <xsl:value-of select="metaitem[@name='sn']/@value"/>
   </option>
  </xsl:if>
```

```
  </xsl:if>
 </xsl:template>

</xsl:stylesheet>
```