XSLT ohjelmointikielenä

Miika Nurminen

14. huhtikuuta 2011

minurmin@jyu.fi

Tiivistelmä

Seminaarityössä käsitellään XSLT-kielen käyttökelpoisuutta ohjelmointikielenä. XSLT:n ja siihen läheisesti liityvän XPath-lausekielen perusominaisuuden käydään läpi esimerkkien kera. Ilmaisuvoimaltaan XSLT on Turing-täydellinen, funktiopohjainen ohjelmointikieli, mutta osittain XML-syntaksista johtuvat työläästi kirjoitettavat kontrollirakenteet ja hankaluus mukauttaa XPath-lausekkeita hankaloittavat kielen käyttöä tavanomaisena ohjelmointikielenä käytännössä. Tästä huolimatta XSLT:llä on paikkansa, jos XML-muotoiselle tiedolle on tarvetta tehdä yksinkertaisia haku- ja muunnosoperaatioita. Laajemmissa järjestelmissä XSLT-skriptit soveltuvat *putket ja suodattimet* -tyylisen arkkitehtuurin osaksi, jolloin muunnoksia voidaan ajaa ketjutetusti, muusta ohjelmakoodista erotettuna.

1 Johdanto

XSLT (XSL Transformations) (Clark, 1999) on XSL (Style Sheets for XML) -kieliperheeseen kuuluva sääntöpohjainen muunnoskieli XML-dokumenttien käsittelyyn. Yleisimmin XSLT:tä käytetään muunnoksiin kahden XML-kielen välillä (esim. DocBookista XHTML:ksi), tai hakuoperaatioihin olemassa olevalle aineistolle. XSLT:n merkitys on kasvanut XML-muodossa olevan datan lisääntymisen myötä – WWW-sivuilla ja -sovelluksissa, käyttöliittymän kuvauskielissä sekä sovellusintegraatiossa yleisesti käytetään XML-pohjaisia formaatteja, joiden käsittely kuuluu sovellusohjelmoijen perusammattitaitoon. Tyypillinen käyttötilanne on datan haku tietyssä formaatissa, datan käsittely, jonka jälkeen tiedot muunnetaan johonkin toiseen formaattiin. Jos sekä tuonti- että vientiformaatit ovat XML-muodossa, XSLT:n käyttö yksinkertaistaa XML-datan käsittelyä esim. DOM-rajapintakutsuihin verrattuna.

Nimestään huolimatta XSLT ei ole tyylikieli samassa mielessä kuin esim. HTML-dokumenttien muotoilussa käytetty CSS (Cascading Style Sheets), koska XSLT itsessään ei määrittele mitään visuaalisia esitystapoja. Yksinkertaisissa tilanteissa (=dokumentin rakennetta ei tarvitse muuttaa) XML-dokumentin ulkoasu selaimessa voidaan määritellä suoraan CSS-kielellä (Bos, 2000), mutta XSL-skriptin avulla voidaan muokata tai generoida uutta koodia visuaalisen esityksen muodostamiseen. Esim. itse määriteltyä XML-kieltä[1] noudattava elementti `<korostus tyyppi="lihavoitu">`

[1] Esimerkeissä on pääosin käytetty IT-tiedekunnan opinto-oppaan julkaisujärjestelmän (Honkaranta et al., 2006) XML-kieltä – `http://opinto-opas.jyu.fi/xoo/`

voitaisiin XSLT-muunnossäännöllä muuttaa HTML-elementiksi `<b>`. Korvausoperaatioiden lisäksi XSLT:tä voidaan yhtä lailla käyttää tietojen käsittelytoimintoihin (järjestäminen, tietojen kokoaminen eri dokumenteista) tai muunnoksiin muihin tekstipohjaisiin formaatteihin (esim. TEX). CSS:n sijaan mielekkäämpiä vertailukohteita XSLT:lle ovatkin tavallisille tekstitiedostoille tai CSV (Comma-separated values) -tyyliselle datalle suunnitellut Unixin komentorivityökalut sed tai awk. Näihin verrattuna XSLT-kielen painopiste on suoran merkkijonokäsittelyn sijaan puuna esitetyn tietorakenteen solmujen käsittelyssä.

Seminaarityössä esitellään XSLT:n ja siihen keskeisesti liittyvän XPath-lausekekielen perusominaisuudet ja arvioidaan esimerkkien kera XSLT:n käyttökelpoisuutta ohjelmointikielenä. XPath- ja XSLT-kehitystyökaluja käydään lyhyesti läpi. Työssä käsitellään pääosin XSLT:n ja XPathin 1.0-versioita, jotka ovat olleet standardiasemassa vuodesta 1999 ja joilla on laaja tuki eri ympäristöissä. Suositusten uudemmat, 2.0-versiot (Kay, 2007) tarjoavat runsaasti uusia ominaisuuksia, mutta niiden tuki ei vielä ole yhtä laajaa. Versioiden keskeisimpien erojen havainnollistamista lukuun ottamatta niiden tarkempi käsittely sivuutetaan. Luku 2 esittelee yleisesti XML-dokumenttien rakennetta ja käsittelytapoja, luvussa 3 esitellään tarkemmin XSLT:n soveltamista. Luvussa 4 arvioidaan XSLT:tä ja verrataan sitä perinteisiin ohjelmointikieliin. Luku 5 on yhteenveto.

2 XML:n käsittelystä

XML (Bray et al., 1998) on W3C:n määrittelemä rakenteinen ja tekstipohjainen metakieli, jonka pohjalta voidaan määritellä samaa yleistä rakennemallia noudattavia kieliä rakenteisen tiedon esittämiseen. XML on yksinkertaistettu versio vanhemmasta SGML-kielestä[2]. Tarkempi syntaksi voidaan kuvata jollakin skeemakielellä (esim. DTD, XML Schema[3], Relax NG[4]), mutta näiden käyttö ei ole välttämätöntä.

XML-dokumentteja kutsutaan joskus itsensä kuvaaviksi, koska dokumentin merkistökoodaus mainitaan dokumentissa itsessään, ja dokumentissa oleva rakenteinen metatieto (elementit ja attribuutit) on erotettu varsinaisesta elementtien sisällä olevasta datasta. Jos dokumentti noudattaa XML:n yleistä rakennemallia, sitä kutsutaan hyvin muodostetuksi (well-formed), mikä on edellytys sen käsittelyyn XML-jäsentimellä. Jos dokumentti noudattaa lisäksi skeemaa (esim. elementtien sisältymisjärjestys, attribuuttien nimeäminen), sitä kutsutaan validiksi. On kuitenkin huomioitavaa, että XML-dokumentin *tulkinta* riippuu täysin sitä käsittelevästä sovelluksesta. Todennäköisesti yleisimmin käytetty XML-kieli on HTML-kielen XML-versio XHTML, jonka käsittelyssä selaimet ovat kuitenkin huomattavasti sallivampia kuin XML-spesifikaatio edellyttäisi. Suuri osa www:ssä olevista XHTML-sivuista on HTML-sivujen tapaan käytännössä epävalidia ”tagisoppaa”, jonka käyttäjät kuitenkin odottavat näkyvän selaimessa.

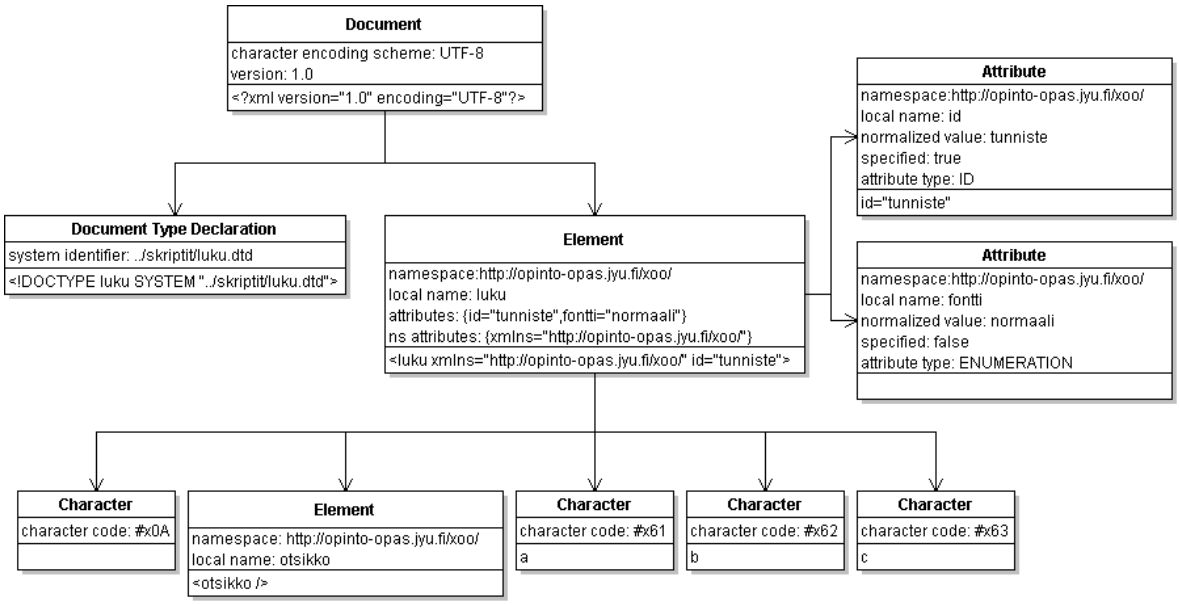
XML:n käsittelyyn on määritelty useita kieliä ja rajapintoja, joista tunnetuin on ehkä oliopohjaisen puumallin tarjoava DOM (Document Object Model)[5], josta on olemassa toteutus yleisimmille ohjelmointikielille. Myös muita kielikohtaisia kirjastoja XML:n käyttöön on olemassa (esim. dom4j[6]). DOM-rajapinnasta poiketen XSLT (Clark, 1999) on itsenäinen muunnoskieli, joka ei ole sidottu muuhun sovellukseen. XSLT:n ytimessä on lausekekieli XPath (Clark & DeRose, 1999), jolla määritellään XML-dokumentin haettavat osat. XSLT:tä lähellä olevia määrityksiä ovat SQL-tyylinen (ja myös XPathia käyttävä) hakukieli XQuery[7], sekä dokumentin ulkoasumäärityksiin

[2] ISO 8879:1986
[3] `http://www.w3.org/XML/Schema`
[4] `http://relaxng.org/`
[5] `http://www.w3.org/DOM/`
[6] `http://dom4j.sourceforge.net/`
[7] `http://www.w3.org/TR/xquery/`

keskittyvä XSL-FO (XSL Formatting Objects)[8]. XSL-muunnoksia ja muita dokumenttien käsittelyoperaatioita voidaan koota yhteen ns. XML Pipelineen, joita voidaan määritellä XProc-kielellä[9]. XSLT-kielen edeltäjä SGML-dokumenteille oli Lisp-tyylista syntaksia käyttävä DSSSL (Document Style Semantics and Specification Language)[10].

2.1 XML-dokumentin rakenne

XML-dokumentti muodostuu sisäkkäisistä elementeistä, jotka voivat sisältää attribuutteja, tekstiä tai uusia elementtejä. Myös muut XML-spesifiset merkinnät (prosessointiohjeet, kommentit ym.) voidaan tulkita puun solmuiksi. Entiteettiviittausten avulla XML-dokumenttiin voidaan sisällyttää myös muita tiedostoja, jotka XML-prosessorin käsittelyn jälkeen tulkitaan osaksi alkuperäistä dokumenttia. Riippumatta dokumentin fyysisestä esitystavasta (tiedosto, XML-tietokanta, tai esim. DOM-puun tapauksessa muistissa oleva tietorakenne) XML-dokumentin abstraktin tietomallin tulee olla XML Information Set-määrityksen (Cowan & Tobin, 2004) mukainen. Infoset kuvaa, mikä XML-dokumentissa oleva tieto on merkityksellistä ja mitä tietoja XML-jäsentimen tulee antaa käsittelevälle sovellukselle. Infoset määrittää esimerkiksi, että minimoitu elementti (esim. `<br />`) on merkitykseltään sama kuin alku- ja lopputunnisteilla merkitty tyhjä elementti (`<br></br>`), ja että elementin attribuuttien järjestyksellä ei ole merkitystä (Wilde, 2010, Best Practices).



Kuva 1: XML Infoset -puu. Juurielementin `fontti`-attribuutin arvo päätellään DTD:stä (`specified`-ominaisuuden arvo on `false`). `#x0A`-merkki on rivinvaihto.

Infoset-puu esitetään Information item-solmuina, jotka ovat tyypitettyjä (*Document Information Item*, *Element Information Items*, *Attribute Information Items*, *Processing Instruction Information Items*, *Unexpanded Entity Reference Information Items*, *Character Information Items*, *Comment Information Items*, *The Document Type Declaration Information Item*, *Unparsed Entity Information Items*, *Notation Information Items*, *Namespace Information Items*), ja joilla voi olla ominaisuuksia. Dokumentti-, dokumenttityyppi- ja elementtisolmuilla voi olla lisäksi alielementtejä. Jokainen

[8] `http://www.w3.org/TR/xsl11/`
[9] `http://xproc.org/`
[10] `http://www.jclark.com/dsssl/`

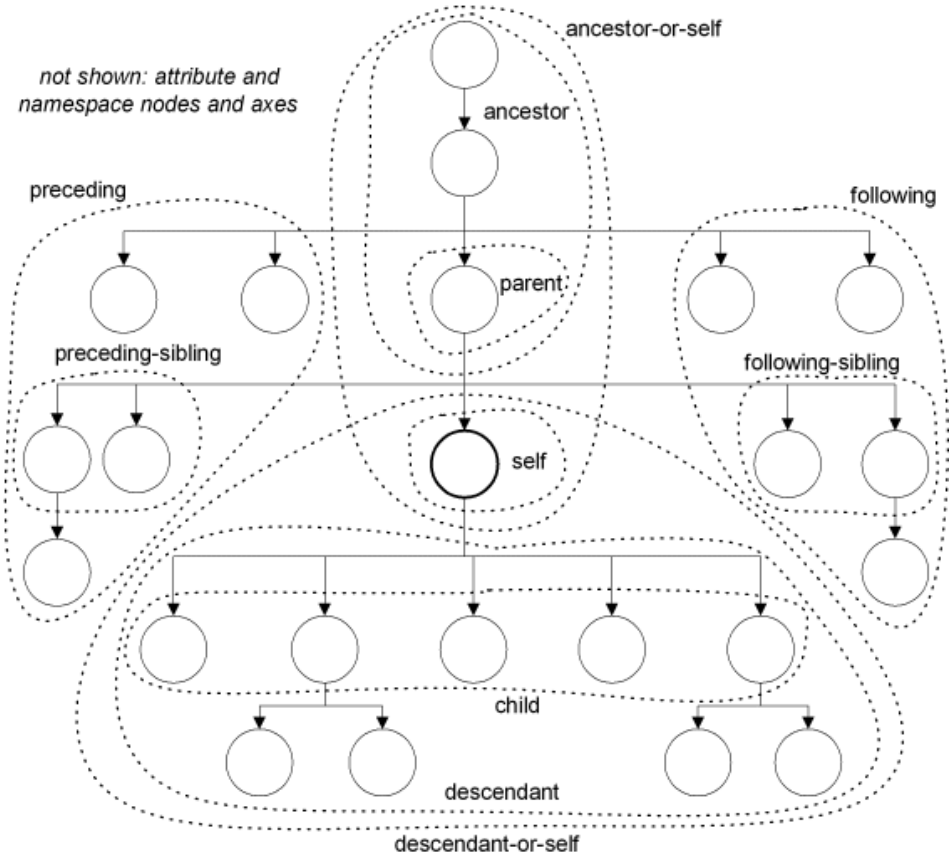
elementin sisältönä oleva merkki tulkitaan erilliseksi solmuksi. Esimerkiksi kuvan 1 Infoset-puu esittää seuraavaa opinto-oppaan minimaalista lukua kuvaavaa XML-dokumenttia:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE luku SYSTEM "../skriptit/luku.dtd">
<luku xmlns="http://opinto-opas.jyu.fi/xoo" id="tunniste">
  <otsikko />
  abc
</luku>
```

Infoset-rakenne muodostaa periaatteessa perustan kaikille XML:ää käsitteleville tietomalleille (DOM-puu, XPath-puu ym.), mutta käytännössä eri tietomallit eivät tällä hetkellä ole identtisiä. Syynä tähän on se, että alkuperäinen XML-spesifikaatio ja ensimmäiset versiot DOM- ja XPath-määrityksistä laadittiin ennen Infosetin standardointia. Infosetiä voidaan kuitenkin pitää referenssimallina, johon XML-dokumentteja käsittelevien tai tuottavien kirjastojen tietomalleja on luontevaa verrata puuttumatta XML-syntaksin yksityiskohtiin.

2.2 XPath

XPath on lausekekieli XML-dokumentin osien hakuun. XPath mallintaa XML-dokumentin Infosetin tapaan solmuiksi, jotka voivat olla elementtejä, attribuutteja, prosessointiohjeita, kommentteja, tekstiä (Infosetissa mainittujen yksittäisten merkkisolmujen koosteita) tai nimiavaruuksia. Hakulausekkeet evaluoidaan suhteessa *kontekstisolmuun*, mikä viittaa XML-prosessorin käsiteltävänä olevaan dokumentin osaan. Evaluointi voi tuottaa *solmujoukon* (node-set), totuusarvon, numeron tai merkkijonon. (Clark & DeRose, 1999)



Kuva 2: XPath-akselit.

XPath-hakulausekkeiden pohjana on tiedostopolkuja muistuttava hakupolku (location path), joka koostuu yhdestä tai useammasta `/`-merkillä erotetusta hakuaskeleesta (location step). Esim. `luku/otsikko` sisältää kaksi hakuaskelta ja hakee kontekstisolmun lapsena olevan `luku`-elementtien lapsielementtinä olevat otsikkoelementit. Yleisemmässä muodossa yksittäinen hakuaskel määritetään muodossa (Hernandez, 2010) **`akseli::NodeTest[predikaatti]`**. Hakuaskeleen osat määritellään seuraavasti:

- *Akseli* (axis) määrittää ”suunnan”, josta solmuja lähdetään etsimään kontekstisolmuun nähden. Oletusakseli on `child`, jolloin haku kohdistuu kontekstisolmun lapsisolmuihin. Useimmat akselit kohdistavat haun solmuihin yleisesti, paitsi `attribute`, joka hakee kontekstisolmun attribuutteja (esim. `korostus/attribute::tyyppi` hakee `korostus`-elementtien `tyyppi`-attribuutit), ja `namespace`, joka kohdistuu nimiavaruuksien määrityksiin. XPath-akseleita on havainnollistettu kuvassa 2[11].
- *NodeTest*:illa voidaan määrätä haettavan solmun (elementin tai attribuutin) nimi tai tyyppi (mikä tahansa solmu, teksti, prosessointiohje ym.).
- *Predikaatti* tarkentaa hakua, esim. `korostus[attribute::tyyppi='ylaindeksi']` hakee kontekstisolmun lapsisolmuista ne `korostus`-elementit, joilla `tyyppi`-attribuutin arvo on `ylaindeksi`.

Lausekkeissa voidaan käyttää laajennettua tai lyhennettyä syntaksia. Lyhennetty syntaksi tuo XPath-kielen entistä lähemmäksi tiedostopolkujen käsittelyyn käytettyjä symboleja. Keskeisimpiä lyhennysmerkintöjä ovat seuraavat (Wilde, 2010, Best Practices):

1. `child::` -akselin voi jättää pois (oletusarvo)
2. `attribute::` -akselin voi korvata merkillä `@` (esim. `korostus/@tyyppi` attribuuttien hakuun tai `korostus[@tyyppi='ylaindeksi']` suodatukseen attribuutin perusteella)
3. `.` on lyhennysmerkintä `self::node()`:lle (vrt. tiedostojärjestelmän nykyinen hakemisto)
4. `..` on lyhennysmerkintä `parent::node()`:lle (vrt. tiedostojärjestelmän ylempi hakemisto)
5. `//` on lyhennysmerkintä `/descendant-or-self::node()/`:lle

XPath-kielessä on määritelty pieni joukko standardifunktioita, jotka käsittelevät perustyyppejä. Esim. `substring-before`-funktiolla voi jakaa merkkijonon kahtia erotinmerkin suhteen. Funktioita voidaan käyttää joko sellaisenaan, jolloin XPath-lausekkeen tulos on funktion arvo, tai hakuaskeleen predikaatin osana. Esim. `following-sibling::kappale[position()=1]` hakee kontekstisolmun jälkeisistä sisarussolmuista seuraavan kappaleen.

Siinä missä XSLT:tä voidaan verrata Unix-tekstityökaluihin, XPath on analoginen säännöllisten lausekkeiden kanssa (joskin varsinaisten merkkijonojen käsittelyn osalta ilmaisuvoimaltaan vähäisempi - XPath 2.0:ssa[12] säännölliset lausekkeet on tuotu kielen osaksi), koska se tarjoaa tehokkaan ja monipuolisen mekanismin XML-dokumentin osien valintaan. CSS:ssä on osittain tietomalliltaan samankaltainen mutta syntaksiltaan erilainen mekanismi dokumentin solmujen valintaan kuin XPathissa, mutta merkittäviä erojakin on: XPathilla ei pysty ottamaan kantaa HTML:n erityispiirteisiin (esim. linkkien `visited`- ja `hover`-arvot) tai sisällön generointiin sivua näytettäessä. Toisaalta CSS:n valintalausekkeet ovat XPathia rajoittuneempia, koska ne perustuvat XPathilla esitettynä `descendant-or-self`-akseliin (`//`). Vapaampi navigointi kontekstisolmun mukaan ei ole CSS:ssä käytettävissä (Pagaltzis, 2006). JavaScript-pohjaisessa jQuery-kirjastossa[13] (2007) on

[11] kuva sivulta `http://www.georgehernandez.com/h/xComputers/XML/XSL/XPath.asp` – alkuperäinen versio sivustolta `http://www.cranesoftwrights.com/training/`. © Crane Softwrights Ltd. 1999.

[12] `http://www.w3.org/TR/xpath20/`

[13] `http://jquery.com/`

DOM-puun käsittelyyn oma lausekekielensä, joka on ottanut vaikutteita sekä CSS:stä että XPathista. Useista kilpailevista hakukielistä huolimatta XPathista on tullut ainakin tutkimuksen puolella *de-facto* -notaatio XML-hakujen esittämiseen (Luk et al., 2002).

XPath-lausekkeiden käsittely sisältyy yleensä XSLT-prosessorin tai muun laajemman kehitystyökalun osaksi, mutta myös erillisiä XPath-työkaluja on saatavilla. Visualisointityökaluja ovat esim. selainpohjainen The XPath Visualizer[14], .NET-pohjainen XPathVisualizer[15], sekä Firefox-plugin XPath Checker[16]. XPath-lausekkeita voi hyödyntää myös DOM-tyylisessä dokumentin ohjelmallisessa käsittelyssä – näin on tehty esim. dom4j-kirjastossa (Rademacher & Strachan, 2001).

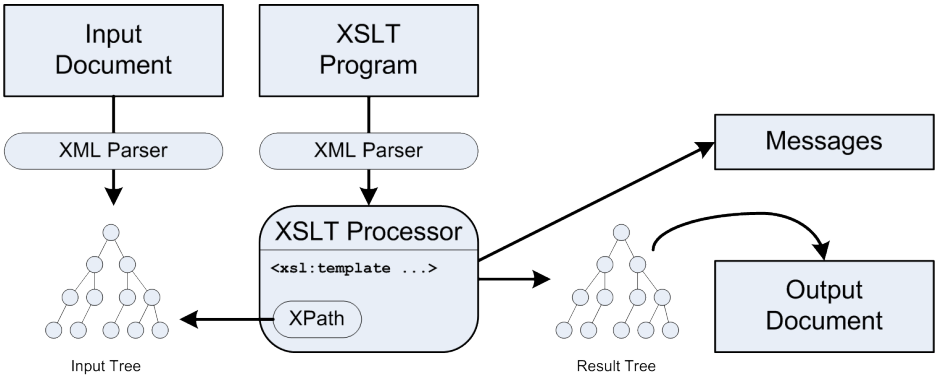
2.3 XSLT

XSLT (Clark, 1999) on deklaratiivinen, XML-syntaksia käyttävä heikosti (Wilde, 2010, XSLT – Part I) ja dynaamisesti (Tittel, 2005) tyypitetty kieli muunnosoperaatioiden määrittelyyn XML-dokumentista tulostiedostoon, joka on tavallisimmin myös XML-muotoa. Muunnos suoritetaan tulkin tapaan toimivalla XSLT-prosessorilla (kuva 3[17]), jota voidaan esim. web-sovelluksen tapauksessa ajaa joko palvelimella tai suoraan selaimessa (selainohjelmasta riippuen – esim. IE:ssä on tuki tyylitiedostolla varustetun staattisen XML-tiedoston käsittelyyn, ja Firefoxissa selaimen sisäistä XSLT-prosessoria voidaan kutsua JavaScriptilla ajonaikaisesti).

Yleisesti käytettyjä komentorivipohjaisia XSLT-prosessoreita ovat Microsoftin MSXML-pakettiin[18] kuuluva msxsl, XSLT- ja XPath 2.0 -suosituksia tukeva Java-pohjainen Saxon[19], Gnome-projektin libxslt-pakettiin[20] kuuluva xsltproc, sekä Apache Xalan[21]. Debuggerin sisältäviä kaupallisia XSLT-kehitystyökaluja ovat mm. Altova XML Spy[22], Exchanger XML Editor[23] ja <oXygen/> XML Editor[24]. Myös yleiskäyttöisiin sovelluskehittimiin on saatavilla XSLT-liitännäisiä (esim. XSLT Debugger for NetBeans[25], Eclipse XSLT Project[26], Emacs XSLT-process[27]).

XSLT-prosessorin suoritusta voidaan ohjata XSL-skriptiin määritellyillä *muunnossäännöillä* (template) tai komentoriviparametreilla ohjelmaa kutsuttaessa. XSLT-prosessori käsittelee sekä syöte- että tulosdokumenttia puumallin avulla lukien syötedokumenttia juuresta alkaen rekursiivisesti, soveltaen muunnossääntöjä dokumentin osiin. Tulospuun fyysiseen muotoiluun voidaan vaikuttaa valitsemalla XSL-skriptin `xsl:output`-elementillä formaatti - XML, HTML tai teksti. XML- ja HTML-formaattien erona on lähinnä elementtien lopputagien käsittely: HTML 4.01:ssä tyhjiä elementtejä (esim. `<br>`) ei suljeta, kun taas hyvin muodostettu XML edellyttää sulkutagia tai minimointia (`<br />`). Jos dokumentin ulostulomuodoksi on valittu teksti, XML-elementtien tagit jätetään huomiotta.

[14] http://www.huttar.net/dimitre/XPV/TopXML-XPV.html
[15] http://xpathvisualizer.codeplex.com/
[16] https://addons.mozilla.org/en-us/firefox/addon/xpath-checker/
[17] Kuva Erik Wilde, lähde http://dret.net/lectures/xml/xslt-2, lisenssi CC BY 3.0.
[18] http://msdn.microsoft.com/en-us/library/ms763742.aspx
[19] http://saxon.sourceforge.net/
[20] http://xmlsoft.org/XSLT/
[21] http://xalan.apache.org/
[22] http://www.altova.com/xmlspy.html
[23] http://www.exchangerxml.com/
[24] http://www.oxygenxml.com/
[25] http://netbeans-xslt-debugger.java.net/
[26] http://wiki.eclipse.org/XSLT_Project
[27] http://xslt-process.sourceforge.net/



Kuva 3: XSLT-prosessorin toiminta.

2.3.1 Perusominaisuudet

XSLT:n tietotyyppejä ovat XPath-lausekkeiden yhteydessä käsitellyt totuusarvot, numerot, merkkijonot ja solmujoukot. XSLT:ssä on lisäksi erityinen tyyppi *puufragmentti* (result tree fragment) (Tittel, 2005), jota käytetään muodostettaessa muuttujia `<xsl:variable>`- tai `<xsl:param>`-elementin alielementeistä (ei siis koske `select`-attribuutilla määriteltyjä muuttuja-arvoja). Puufragmentit muistuttavat solmujoukkoja, mutta sisältävät vain yhden juurielementin ja ovat rajoitettuja sallittujen operaatioiden suhteen. Puufragmenttia voi käsitellä merkkijono-operaatioilla ja kopioida tulosdokumenttiin sellaisenaan, mutta fragmentin osia ei voi valita tai käsitellä hakupolkuja sisältävillä XPath-lausekkeilla. Käytännössä rajoitteet estävät esim. merkkijonosta muodostetun XML-dokumentin osan monivaiheisen käsittelyn saman XSL-skriptin sisällä.

Muunnossäännöillä määritellään käsiteltävät dokumentin kohdat. Muunnossääntö määritellään elementillä `<xsl:template match="pattern">`, missä pattern on XPath-muodossa esitetty hakulauseke. Muunnossäännön sisällä tulosdokumenttia voidaan muotoilla sellaisenaan XML-muodossa, solmu kerrallaan `<xsl:element>` ja `<xsl:attribute>` -elementeillä (jolloin esim. attribuuttiarvoja voidaan kontrolloida `<xsl:attribute>`-elementtiin sijoitetuilla lausekkeilla), tai hakemalla solmun arvo (esim. tekstisisältö) syötedokumentista tai muuttujasta `<xsl:value-of>`-elementillä. Valittava solmu määritetään XPath-lausekkeella, jossa kontekstisolmu on suhteessa nykyiseen muunnossääntöön. `<xsl:text>`-elementillä voidaan luoda eksplisiittisesti tekstisolmuja. Verrattuna XML-koodin joukossa olevaan tekstiin välilyönnit ja rivinvaihdot säilyvät, ja käytettäessä `disable-output-espacing`-määrettä XML-spesifiset erikoismerkit, kuten `<` ja `&` merkitään tekstiin sellaisenaan entiteettimuotojen (`&lt;`, `&amp;`) sijaan.

Muunnossäännön suorituksen yhteydessä dokumentin läpikäyntiä voidaan ohjata edelleen `<xsl:apply-templates select="pattern"/>` -elementillä. Jos `apply-templates`-kutsun hakulauseketta ei vastaa yksikään itse määritelty sääntö käytetään oletuskäsittelyä, eli kopioidaan tekstisolmut hakuun täsmäävistä elementeistä alielementteineen. Ilman `apply-templates`-kutsua syötedokumentin läpikäynti pysähtyy nykyiseen sääntöön, ellei dokumentissa ole muita vielä läpikäymättömiä osia, joihin jokin muu muunnossääntö täsmää. Jos sääntöjä ei ole määritelty ollenkaan, oletuskäsittelyä sovelletaan koko syötedokumenttiin, jolloin tuloksena on dokumentin tekstiesitys (Wilde, 2010, XSLT – Part I).

Varsinaisiin XPath-lausekkeisiin täsmäävien sääntöjen sijaan muunnossääntöjä voi myös nimetä, jolloin niitä kutsutaan manuaalisesti `<xsl:call-template>`-elementillä. Tällöin kontekstisolmu ei muutu, mutta monessa paikassa tarvittavia perusoperaatioita (esim. tietyille elementeille tarvittavaa merkkijonon käsittelyä – katso luku 3.2) voidaan kutsua yhtenäisellä tavalla, oleellisesti aliohjelmien tapaan. XSL-skriptissä on mahdollista määritellä myös muuttujia, joiden arvo määrätään XPath-lausekkeella, ja joita voidaan tämän jälkeen käyttää muiden lausekkeiden osana.

XSLT-prosessorille annetut syöteparametrit näkyvät XSL-skriptissä globaaleina muuttujina. Myös muunnossääntökutsulle voidaan antaa parametreja, joita käytetään muuttujien tapaan. Muiden funktiokielten tapaan muuttujien ja parametrien arvoa ei voi muuttaa jälkikäteen, mutta kutsumalla muunnossääntöä rekursiivisesti voidaan niistä tavallaan luoda ”kopioita” uusilla arvoilla aiemmassa iteraatiossa saatujen arvojen ja muunnossääntöjen paluuarvojen pohjalta.

Muita XSLT:n keskeisiä ominaisuuksia ovat kontrollirakenteet `if` ja `choose`, jotka mahdollistavat ehdollisen käsittelyn muunnossäännön sisällä. `choose` vastaa C-tyylisten kielten switch/case-rakennetta; `if`-elementissä on vain yksi vaihtoehtoinen suorituslohko ilman erillistä else-rakennetta. XSLT sisältää myös for-silmukan vastineen (`for-each`), mutta tämä on ilmaisuvoimaltaan rajoitettu verrattuna yleisempiin rekursiivisiin kutsuihin. XSLT:n kehittyneemmistä ominaisuuksista mainittakoon elementtien numerointi, järjestäminen, ID/IDREF-avainten käsittely, ja tietojen haku ulkoisista XML-dokumenteista `document()`-funktiolla. (Clark, 1999)

2.3.2 Käyttöesimerkki

Esimerkkinä XSLT:n peruskäytöstä määritellään seuraavat muunnossäännöt opinto-oppaan XML:n käsittelyyn (muotoilumääritykset `xsl:stylesheet` ja `xsl:output-element` on jätetty pois):

```
<xsl:template match="/">
<html>
  <head>
    <title><xsl:value-of select="luku/otsikko" /></title>
  </head>
  <body>
    <xsl:apply-templates select="luku"/>
  </body>
</html>
</xsl:template>

<xsl:template match="luku">
<div>
  <xsl:apply-templates select="luku|kappale" />
</div>
</xsl:template>

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

<xsl:template match="viite[@viitetyyppi='www']">
  <xsl:element name="a">
  <xsl:attribute name="href"><xsl:value-of select="." /></xsl:attribute>
  <xsl:value-of select="." /></xsl:element>
</xsl:template>
```

Dokumentin läpikäynti aloitetaan juurielementistä, mikä täsmää ensimmäiseen muunnossääntöön. Tulosdokumenttiin kirjoitetaan HTML-dokumentin tyypilliset peruselementit. Otsikko haetaan juurielementtinä olevan luvun alla olevasta otsikkoelementistä. `apply-templates`-kutsua sovelletaan vain `luku`-elementteihin, jonka jälkeen suoritus siirtyy seuraavaan muunnossääntöön. Elementit korvataan `div`-elementeiksi, ja suoritusta jatketaan rekursiivisesti alilukuihin ja välittömästi luvun alle sisältyviin `kappale`-elementteihin (jotka muunnetaan omassa säännössään `p`-elementeiksi). Kappale-elementin muunnossäännössä ajettava `apply-templates`-kutsu ajetaan kaikille kappaleen alta löytyville alielementeille. `<viite viitetyyppi="www">`-elementtiä lukuun ottamatta elementtien sisältö kopioidaan tekstimuodossa, koska muita muunnossääntöjä ei ole määritely. Viimeisenä oleva WWW-linkkien käsittely on esimerkki muunnossäännöstä, jossa elementti luodaan eksplisiittisesti XSLT:n omilla luontielementeillä. Tämä mahdollistaa href-arvon asettamisen niin, että XSL-tiedoston rakenne pysyy hyvin muodostettuna (esim. `<a`

`href="<xsl:value-of select="."/>"` -tyylinen suora tulostus ei ole XML-syntaksin takia mahdollista).

Sovellettaessa muunnosskriptiä opinto-oppaan XML-tiedostolle

```
<luku>
  <otsikko>Informaatioteknologian tiedekunta</otsikko>
  <luku>
    <otsikko>Tiedekunta ja sen laitokset</otsikko>
    <kappale><korostus tyyppi="lihavoitu">Informaatioteknologian tiedekunnassa</korostus>
    opiskelee noin 1 500 perustutkinto-opiskelijaa ja
    lähes 200 jatko-opiskelijaa.</kappale>
  </luku>
  <kappale>WWW: <viite viitetyyppi="www">http://www.jyu.fi/it/</viite></kappale>
</luku>
```

saadaan tulosdokumentti

```
<html>
<head>
  <title>Informaatioteknologian tiedekunta</title></head>
<body>
  <div>
    <div>
    <p>Informaatioteknologian tiedekunnassa opiskelee noin 1 500 perustutkinto-
       opiskelijaa ja lähes 200 jatko-opiskelijaa.</p>
    </div>
    <p>WWW: <a href="http://www.jyu.fi/it/">http://www.jyu.fi/it/</a></p>
  <div>
</body>
</html>
```

Lukujen otsikot eivät ole tulosdokumentin `body`-osiossa mukana, koska `luku`-elementin muunnossäännöstä kutsutaan vain muita `luku`- ja `kappale`-elementtejä. Toisaalta `korostus`-elementin alla oleva teksti on mukana, koska `kappale`-elementin muunnossäännössä käsiteltäviä elementtejä ei ole rajattu. XSL-skriptejä kehitettäessä tyypillinen ongelma on valita sopiva käsittelyjärjestys niin, että kaikki tarvittavat elementit käydään läpi mahdollisimman tehokkaasti. Esim. potentiaalisesti monen eri solmun yli hakeva `descendant-or-self`-akseli vaatii enemmän solmujen läpikäyntiä kuin yhdelle tasolle rajattu haku. Yhdistämällä `value-of`-elementtejä, muunnossääntöjen kutsuja tai kopiointitoimintoja varomattomasti on mahdollista luoda tarpeettoman (periaatteessa myös rajattoman) monia kopioita samasta sisällöstä tulosdokumenttiin.

Laajennetaan skriptiä rivinvaihtoa ja korostuksia käsittelevillä muunnossäännöillä, jolloin `korostus`-elementti korvautuu HTML-kielen `b`-elementiksi. Erityyppiset korostustyylit erotellaan `choose`-rakenteella, ja korostuselementtien alla olevat tekstisolmut kopioituvat oletuskäsittelyssä.

```
<xsl:template match="rivinvaihto"><br/></xsl:template>

<xsl:template match="korostus">
  <xsl:choose>
    <xsl:when test="@tyyppi [.='lihavoitu']">
      <b><xsl:apply-templates select="text()|rivinvaihto|korostus"/></b>
    </xsl:when>
    <xsl:when test="@tyyppi [.='tasaleveys']">
      <tt><xsl:apply-templates select="text()|rivinvaihto|korostus"/></tt>
    </xsl:when>
    <xsl:otherwise>
      <i><xsl:apply-templates select="text()|rivinvaihto"/></i>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Tässä tapauksessa `choose`-lauseen käyttö ei ole välttämätöntä: säännön voisi jakaa osiin, esim. lihavoinnin osalta `<xsl:template match="korostus[@tyyppi='lihavoitu']">`. Tämäntyyppisiin yksinkertaisiin muunnoksiin XSLT soveltuu hyvin – korvaamalla `if` ja `choose`-rakenteet omilla muunnossäännöillään koodin ylläpidettävyys pysyy hyvänä.

3 XSLT:n sovelluksia

Käydään läpi lyhyillä esimerkeillä XSLT:n kehittyneempiä ominaisuuksia ja käyttöä eri sovelluksissa.

3.1 Identtinen kuvaus

Monet XSL-skriptit pohjautuvat ns. identtiseen kuvaukseen[28], joka kopioi XML-solmun sisällön sellaisenaan. Identtinen kuvaus on hyödyllinen, jos halutaan säilyttää dokumentin rakenne pieniä poikkeuksia lukuun ottamatta – erityisesti, jos useimpien elementtien nimi pidetään samana. Poikkeukset ilmaistaan muilla muunnossäännöillä.

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

Tyypillinen tilanne identtisen kuvauksen käyttöön on muunnosketju, jossa käytetään rinnakkain XHTML-kieltä (esim. muotoilukielenä kappaleiden sisällä), ja jotain omaa XML-kieltä (dokumentin ”ylärakenteen” kuvaukseen), ja muodostetaan tästä WWW-sivu. XHTML-sisältö voidaan päästää läpi sellaisenaan, ja oman XML-kielen elementit muunnetaan XHTML:ksi.

3.2 Merkkijonon korvaus

XSLT 1.0:n merkkijono-operaatiot ovat hyvin rajalliset ja rajoittuvat lähinnä muutamaan XPathissa määriteltyyn perusfunktioon: `concat`, `contains`, `normalize-space` (whitespacen poisto), `starts-with`, `string-length`, `substring`, `substring-after`, `substring-before` ja `translate` (yksittäisten merkkien korvaus). XSLT:llä itsellään voi kuitenkin määritellä myös uusia funktioita (nimettyjä muunnossääntöjä) merkkijonojen käsittelyyn, joista hyödyllisimpiä on yleinen merkkijonon korvaus (Mangano, 2005, luku 2.7).

```
<xsl:template name="replace-string">
  <xsl:param name="text"/>
  <xsl:param name="from"/>
  <xsl:param name="to"/>
  <xsl:choose>
    <xsl:when test="contains($text, $from)">
      <xsl:variable name="before" select="substring-before($text, $from)"/>
      <xsl:variable name="after" select="substring-after($text, $from)"/>
      <xsl:variable name="prefix" select="concat($before, $to)"/>
      <xsl:value-of select="$before"/>
      <xsl:value-of select="$to"/>
      <xsl:call-template name="replace-string">
        <xsl:with-param name="text" select="$after"/>
        <xsl:with-param name="from" select="$from"/>
        <xsl:with-param name="to" select="$to"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="$text"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Esimerkissä sovelletaan rekursiota: korvausfunktiota kutsutaan niin kauan kuin korvattava merkkijono esiintyy. Merkkijonon katkaisu hoidetaan standardifunktioilla `substring-before` ja `substring-after`.

[28] Esimerkki on Dave Pawsonin XSLT FAQ-sivustolta, `http://www.dpawson.co.uk/xsl/sect2/identity.html`

3.3 Epäsuora viittaus muunnossääntöön

XSLT:n ehkä keskeisin rajoite ohjelmoinnin kannalta on muunnossääntöjen kutsumuodon staattisuus ja uudelleenkäytön vaikeus. Dokumentin osia valitessa muunnossäännöt on sidottava tietyn nimisiin elementteihin, vaikka käsittelylogiikka olisi muuten identtinen (esim. taulukkomaisen rakenteen käsittelyn osalta). Elementtien tai muunnossääntöjen nimiä tai muutakaan akseleihin tai solmutesteihin liittyvää tietoa ei voi merkitä XPath-lausekkeisiin muuttujamuodossa. Funktioiden ja loogisten operaattoreiden yhteydessä muuttujia voi käyttää, ja joissakin tilanteissa eri kielten käsittelylogiikkaa voidaan mukauttaa jakamalla muunnos osiin (katso luku 3.4). Silti yksittäisten lausekkeiden rakenteeseen kokonaisuutena ei voi vaikuttaa dynaamisesti.

Muunnossääntöjen kutsurajoitusta on kuitenkin mahdollista kiertää sijoittamalla niitä omiin nimiavaruuksiinsa (template references) (Novatchev, 2001; Haines, 2000[29]). Tällöin XPathilla voidaan muodostaa erityinen `namespace-uri`-hakulauseke, joka täsmää muunnossääntöön epäsuorasti. Tämän hakulausekkeen voi sijoittaa muuttujaan ja käyttää edelleen toisen säännön kutsuparametrina, jolloin kutsuttavassa muunnossännössä ajettava ”funktio” on siis vaihdettavissa. Koska jokaista näin kutsuttavaa sääntöä varten täytyy määritellä oma nimiavaruutensa, menetelmää voidaan pitää kömpelönä ja on tavallaan XPathin väärinkäyttöä, koska lauseketyyppi on varsinaisesti suunniteltu XSL-skriptissä olevan datan hakuun, ei niinkään XSL-koodin ajamiseen (Clark, 1999). Tästä huolimatta ominaisuus on oleellinen tarkasteltaessa XSLT:n ilmaisuvoimaa ohjelmointikielenä.

Esimerkkinä dynaamisen kutsun toteutuksesta määritellään erilliset summa- ja tulofunktiot, joita kutsutaan `apply`-funktion kautta. Funktio kutsuu parametrina annettua funktiota syötedokumentissa olevalle listalle (juurielementin alla olevien elementtien tekstisolmut – elementtien nimillä ei ole väliä, mutta arvot oletetaan numeerisiksi). Esimerkki perustuu osittain Novatchevin (2001) artikkeliin. Määritellään funktiot tiedostossa `functions.xsl`.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version = "1.0" >
<xsl:template
  name="sum" match="*[namespace-uri()='calc-sum']">
  <xsl:param name = "values" select="/.."/>
  <xsl:param name = "result" select="0" />
  <xsl:choose>
    <xsl:when test="count($values) = 0">
      <xsl:value-of select="$result" />
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="sum">
        <xsl:with-param name="values" select="$values[position() &gt; 1]" />
        <xsl:with-param name="result" select="$values[1]+$result" />
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template
  name="prod" match="*[namespace-uri()='calc-prod']">
  <xsl:param name = "values" select="/.."/>
  <xsl:param name = "result" select="1" />
  <xsl:choose>
    <xsl:when test="count($values) = 0">
      <xsl:value-of select="$result" />
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="prod">
        <xsl:with-param name="values" select="$values[position() &gt; 1]" />
        <xsl:with-param name="result" select="$values[1]*$result" />
      </xsl:call-template>
```

[29] XSLTalk-listan keskusteluja. `http://web.archive.org/web/20060510011133/http://www.coreyhaines.com/coreysramblings/PermaLink.aspx?guid=9c2c2733-ac62-4d99-bd0f-a057a01a2098`

```
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
</xsl:stylesheet>
```

tiedosto `apply.xsl` sisältää funktiokutsut ja muotoilee tulostusta:

```
<xsl:stylesheet version = "1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:sum="calc-sum"
  xmlns:prod="calc-prod">
<xsl:import href = "functions.xsl" />
<xsl:output method = "text" />

<sum:p />   <!-- dynaaminen kutsu täsmää näihin elementteihin -->
<prod:p />

<xsl:template match = "/" >
Sum:<xsl:call-template name = "apply" >
    <xsl:with-param name = "mvalues" select = "child::*/*" />
    <xsl:with-param name = "func" select = "document('')/*/sum:*[1]" />
  </xsl:call-template>

Prod:<xsl:call-template name = "apply" >
    <xsl:with-param name = "mvalues" select = "child::*/*" />
    <xsl:with-param name = "func" select = "document('')/*/prod:*[1]" />
  </xsl:call-template>
</xsl:template>

<xsl:template name = "apply" >
    <xsl:param name = "mvalues" select = "/.." />
    <xsl:param name = "func" select = "0" />
Input [<xsl:for-each select="$mvalues"><xsl:if test="position() &gt; 1">,</xsl:if>
<xsl:value-of select="." /></xsl:for-each>]
Result <xsl:apply-templates select = "$func" >
      <xsl:with-param name = "values" select = "$mvalues" />
    </xsl:apply-templates>
</xsl:template>
</xsl:stylesheet>
```

Viittaustekniikan ideana on määrittää kutakin viitattavaa funktiota varten oma nimiavaruutensa (esim. `xmlns:sum="calc-sum"`), ja sijoittaa XSL-skriptiin tyhjä nimiavaruuteen sijoittuva elementti (`<sum:p />`). XPath-lauseke `document('')/*/sum:*[1]` täsmää XSL-dokumentissa itsessään (koska `document`-funktio viittaa tyhjään dokumenttiin) oleviin `sum`-nimiavaruuden elementteihin, jolloin ajo siirtyy muunnossääntöön, joka on määritelty käsittelemään niitä (`match`-lauseke `*[namespace-uri()='calc-sum']`). Ajettaessa `apply.xsl` tiedostolle

```
<numbers>
  <num>1</num>
  <num>2</num>
  <num>3</num>
  <num>4</num>
  <num>5</num>
</numbers>
```

saadaan tulos

```
$xsltproc map.xsl input.xml

Sum:
Input [1,2,3,4,5]
Result 15

Prod:
Input [1,2,3,4,5]
Result 120
```

Idean pohjalta on mahdollista määritellä myös korkeamman asteen funktioita. Novatchevin esimerkeissä on toteutettu XSLT:llä klassisessa Hughesin (1989) funktio-ohjelmoinnin kuvauksessa luetellut esimerkit numeerista integrointia myöten. Tuloksena on kokonainen funktio-ohjelmointikirjasto FXSL[30], jossa on määritelty vastineet useimmille esim. Haskell-kielen (Jones, 2002) Prelude-moduulista löytyville funktioille (esim. `map`, `sum`, `foldr`, `curry`). Haskell-tyylistä laiskaa laskentaa XSLT:llä ei kuitenkaan ole mahdollista tehdä, ja on korostettava, että esimerkit ovat todella työläitä toteuttaa. Koodirivien määrän puolesta vertailu perinteisiin funktiokieliin ei ole kovin mielekästä, mutta kielen ilmaisuvoimasta sinänsä ei ole epäselvyyttä.

Opinto-oppaassa periaatetta on sovellettu yksinkertaisemmassa muodossa. Taulukoiden generoinnissa erityyppisten taulukoiden kehysten ja sarakemääritysten käsittelyyn liittyvät koodit on erotettu dynaamisesti kutsuttaviin sääntöihin, joita kutsutaan yksittäisten rivien ja solujen käsittelyyn liittyvästä (näille yhteisestä) muunnossäännöstä. Tämä vähentää yhteisessä muunnossäännössä tarvittavia `choose`-rakenteita ja koodin toisteisuutta (taulukon tyyppi tarkastetaan vain kerran – dynaamisesti kutsuttavia sääntöjä määritettäessä), mutta ei varsinaisesti käytä korkeamman asteen funktioita.

3.4 Skriptien ketjutus ja dokumenttien koostaminen

Yksittäisiä XSL-skriptejä voidaan yksinkertaistaa oleellisesti jakamalla muunnokset useisiin eri vaiheisiin ja käyttämällä yhden XSL-muunnoksen tulostiedostoa syötteenä seuraavalle muunnokselle. Periaate on sama kuin komentorivityökalujen yhdistäminen standardisyötteen ja -tulostuksen kautta *putket ja suodattimet* (pipes and filters, tietovuoarkkitehtuuri, piiput ja filtterit) -arkkitehtuurin mukaisesti (Garlan & Shaw, 1994). Tällöin välivaiheena olevien XML-tiedostojen validius ei ole oleellista. Skriptien ketjutusta on käytetty esimerkiksi opinto-oppaan julkaisujärjestelmässä seuraavalla periaatteella:

1. Kokoa koostetiedostossa määritellyt lähdedokumentit yhteen tiedostoon suodatettuna varianttien mukaan (variantit ovat samaa pohjasisältöä käyttäviä erillisiä julkaisuja, joissa vaihtelua sisällytettävien lukujen tai dokumenttien suhteen – esim. oppaat opintopiste- ja opintoviikko-opiskelijoille).
2. Lisää lukuihin, kuviin ja taulukoihin numerointitieto.
3. Yksikäsitteistä tunnisteet.
4. Muunna julkaisumuotoon (esim. HTML-sivu + HTML-sisällysluettelo).

Tarkastellaan tarkemmin muunnoksen ensimmäistä vaihetta, jossa tiedostot kootaan yhteen. osa `koostaja.xsl`-skriptistä:

```
<xsl:output method="xml" version="1.0" encoding="iso-8859-1" omit-xml-declaration="no"
  indent="no" doctype-system="../skriptit/opas.dtd" />
<xsl:param name="variantti" select="/opas/@opasvariantti" />

<xsl:template name="myfile">
  <xsl:param name="parent" select="tiedosto/following-sibling::*" />
  <xsl:param name="this" select="." />
  <xsl:param name="tied" select="tiedosto" />
  <xsl:choose>
    <xsl:when test="document($tied)/luku/otsikko">
      <!-- tiedostot, joilla on otsikko -->
      <xsl:for-each select="document($tied)/luku"></xsl:for-each>
    </xsl:when>
    <xsl:otherwise>
      <!--muut tiedostot -->
      <xsl:for-each select="document($tied)/luku/*"></xsl:for-each>
```

[30] http://fxsl.sourceforge.net/

```
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="luku|otsikko|kuvio|taulukko|lista|kappale">
  <xsl:choose>
    <xsl:when test="(string-length($variantti) &gt; 0)
                    and (string-length(@opasvariantti) &gt; 0)
                    and ($variantti!=@opasvariantti)" />
    <xsl:otherwise>
      <xsl:choose>
        <xsl:when test="child::tiedosto" >
          <xsl:call-template name="myfile" >
            <xsl:with-param name="parent" select="tiedosto/following-sibling::*" />
            <xsl:with-param name="this" select="." />
            <xsl:with-param name="tied" select="tiedosto" />
          </xsl:call-template>
        </xsl:when>
        <xsl:otherwise>
          <xsl:copy>
            <xsl:apply-templates select="@*|node()"/>
          </xsl:copy>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Tyylitiedoston alussa oleva `variantti`-parametri luetaan oletuksena syötetiedoston `opas`-juurielementin attribuutista, tai voidaan antaa XSLT-prosessorille komentoriviparametrina. Nimettyä `myfile`-sääntöä kutsutaan uuden lähdedokumentin lukemiseen tulostiedostoon – suurin osa muunnossäännön yksityiskohdista on jätetty tässä pois. Jälkimmäistä sääntöä kutsutaan kaikille ”lohkotasoisille” (esim. kappale, kuvio, taulukko – vrt. lohkotason elementit HTML:ssä) oppaan elementeille. Jos elementti kuuluu ”väärään” varianttiin, se suodatetaan pois. Muussa tapauksessa tutkitaan, onko nykyisen elementin alla tiedostoa. Jos on, kutsutaan jälleen `myfile`-sääntöä, joka lukee nykyisen elementin tilalle tiedoston sisällön ja käsittelee sitä edelleen kutsuen lopulta rekursiivisesti alkuperäistä lohkotason sääntöä (nyt toisen tiedoston sisällölle). Viimeisessä `otherwise`-lohkossa muut elementit kopioidaan identtisellä kuvauksella.

4 XSLT:n arviointia

Luvussa arvioidaan XSLT-kieltä käytännön sovellusten ja kielen ominaisuuksien kannalta.

4.1 Käyttökohteita

XSLT:n merkittävin etu on yhteensopivuus XML:n kanssa. Kieli on alusta alkaen suunniteltu XML-datan lukemiseen, ja varsinaisten XSL-skriptien kirjoittaminen ja validointi onnistuu millä tahansa XML:ää tukevalla editorilla. Toinen etu on XPath-kieli, joka on ilmaisuvoimainen mutkikkaisiinkin hakuoperaatioihin ja käytössä muissakin XML-sovelluksissa. Skriptien parametrisointi ja muunnosten ketjuttaminen sallivat dokumentin rakentamisen löyhästi kytketyillä komponenteilla putki-suodatin-arkkitehtuurimallilla. Muunnosten jakaminen osiin mahdollistaa tietynasteisen uudelleenkäytön muunnoksia kootessa (muunnetaan dokumentti tietylle ”standardikielelle” ennen jatkokäsittelyä), vaikka kieli itsessään ei tue uudelleenkäyttöä kovin hyvin.

Jos sovellus käyttää XML:ää viestintään tai tietojen esittämiseen, XSL-skriptien integrointi sovellukseen on suoraviivaista riippumatta siitä, millä kielillä ja kirjastoilla muu osa sovellusta on tehty. XSLT ei millään tavalla sulje pois muiden XML-työkalujen kuten DOM-rajapinnan käyttöä – esim. jos sovelluksen tulee lukea tietoa suoraan XML-tiedostoista, erillisen jäsentimen käyttö

on lähes välttämätöntä. XSLT vähentää kuitenkin tarvetta XML-datan manuaaliseen läpikäyntiin ja muokkaukseen muussa ohjelmakoodissa. Esimerkiksi opinto-oppaan julkaisujärjestelmä (luku 3.4; Honkaranta et al., 2006) käyttää XSLT:n lisäksi palvelinpuolella PHP, Python ja sed-skriptejä – ja web-käyttöliittymä JavaScriptia. XML-dokumenttien koostamisen, esikäsittelyn ja HTML-ulostulon lisäksi XSLT:tä käytetään myös LaTeX-koodin generointiin sekä Joose-käyttöliittymän osalta (Hautakangas et al., 2009) selaimen oman XSLT-prosessorin kutsuissa.

Esimerkkinä yksinkertaisemmasta XSLT-sovelluksesta Tverskov (2010) on määritellyt seitsemästä vaiheesta koostuvan mukautettavan muunnosketjun XML-datan muuntamiseksi CSV:ksi. Toisessa ääripäässä on ohjelmistojen dokumentointiin suunniteltu DocBook-kieli[31], joka on paitsi itsessään erittäin laaja, lähes 400 elementtiä sisältävä XML-kieli, myös sisältää XSLT:llä toteutetut muunnossäännöt HTML- ja PDF-muotoon ja muihin yleisimpiin julkaisuformaatteihin (Stayton, 2007). Apache Cocoon[32] on monipuolinen web-sovelluskehys, joka käyttää keskeisesti XSLT:tä julkaisutoimintojensa osana. Myös monet OpenOffice-tekstinkäsittelyohjelman muunnossuotimet on toteutettu XSLT:llä (Eisenberg, 2005, luku 9).

4.2 XML-syntaksista

XSLT:n omat kontrollirakenteet ovat periaatteessa yksinkertaisia, mutta imperatiivisiin kieliin tottuneille rekursion käyttö muunnossääntöjen arvoja laskettaessa voi olla hankalaa hahmottaa. XSLT-kehitystyökalut (katso luku 2.3) auttavat tässä, mutta monet saatavilla olevat debuggerit ovat kaupallisia. Myös XPath-lausekkeiden muotoilu voi olla haastavaa, koska lausekkeet muodostavat alikielen XSLT:n sisälle selvästi XML:stä poikkeavalla (joskin sinänsä tehokkaalla ja monipuolisella) syntaksilla. Toisaalta XPath-visualisointityökalut (luku 2.2) helpottavat huomattavasti kielen omaksumista. XML-syntaksi tekee varsinkin ehtolauseiden rakentamisesta tarpeettoman työlästä. Tätä voidaan helpottaa käyttämällä jotain esiprosessoria XSLT-koodin generointiin. Useita vaihtoehtoisia syntakseja on ehdotettu, kuten libslax[33], XSLTXT[34], XSPP[35] ja Xl/t (Love, 2007). Valitettavasti nämä eivät ole kovin laajasti käytettyjä eivätkä W3C:n tukemia.

Yleinen skriptejä jonkin verran mutkistava ja esimerkeissä pääosin sivuutettu tekijä on nimiavaruuksien huomiointi. Jos syöte- tai tulosdokumentti käyttää XML-nimiavaruuksia, ne tulee merkitä kaikkiin muunnossääntöihin. Esim. opinto-oppaan tapauksessa skripteissä tulisi määritellä syötedokumenttia varten viittaus nimiavaruuteen `xmlns:o="http://opinto-opas.jyu.fi/xoo/"` ja tulosdokumenttia varten html-nimiavaruus `http://www.w3.org/1999/xhtml/`. Syötedokumentin nimiavaruuden viite on merkittävä kaikkiin XPath-lausekkeisiin, tai haku ei onnistu (esim. luvussa 2.3.2 mainittu otsikon haku voitaisiin merkitä nimiavaruuden kera muodossa `<xsl:value-of select="o:luku/o:otsikko"/>`). Tilanne mutkistuu entisestään, jos tulosdokumentissa on useita nimiavaruuksia. Tällöin nimiavaruus on merkittävä eksplisiittisesti myös osaan tulostettavista elementeistä – muuten voidaan käyttää oletusnimiavaruutta.

Vaikka useimmat ohjelmoijat pitävät kielen XML-syntaksina rasitteena, se on myös mahdollistava tekijä ”metatyylisivuille”. Rutiininomaisia XSL-muunnoksia (esim. elementtien nimien korvaus) voidaan automatisoida tekemällä XSL-skriptejä generoivia muunnossivuja, jolloin varsinaisen muunnoksen määritys muuttuu koodin kirjoituksesta enemmän konfiguroinnin tyyliseksi. Perusidea on sama kuin sovellusaluepohjaisissa kielissä (DSL, Domain Specific Languages) (Kelly & Tolvanen, 2008). Tekniikkaa on sovellettu esim. opinto-oppaan Joose-editorissa ajettavassa 2-suuntaisessa muunnoksessa XHTML:sta oppaan XML:ksi (Hautakangas et al., 2009; Lehtonen,

[31] `http://www.docbook.org/`
[32] `http://cocoon.apache.org/`
[33] `http://code.google.com/p/libslax/`
[34] `http://savannah.nongnu.org/projects/xsltxt`
[35] `http://ostatic.com/xspp/home/1`

2008[36]). Esimerkiksi seuraavassa on osa editorin aiemman version muunnosmäärityksestä, jossa kuvataan elementtien vastaavuuksia:

```
<classreplace mode="juuri" from="luku" to="div" toclass="luku" newmode="juuri"/>
<korostus mode="juuri" tyyppi="kursivoitu" to="em" newmode="juuri" />
<korostus mode="juuri" tyyppi="lihavoitu" to="strong" newmode="juuri" />
<korostus mode="juuri" tyyppi="pieni" to="small" newmode="juuri" />
<korostus mode="juuri" tyyppi="ylaindeksi" to="sup" newmode="juuri" />
```

Metatyylisivujen ongelmana on, että generaattoriskriptien toteutus on huomattavasti tavanomaisia XSL-skriptejä työläämpää, ja monimutkaisuus kasvaa syötetietojen ilmaisuvoiman kasvaessa. Joose-editorin tapauksessa konfigurointitiedostossa pidetään lähinnä yksinkertaiset muunnokset, joissa tietyt elementti- ja attribuuttinimet korvataan suoraviivaisesti toiselle nimelle. Enemmän käsittelylogiikkaa vaativat muunnosoperaatiot on kirjoitettu manuaalisesti generaattoriskriptin osaksi, ja ne kopioituvat sellaisenaan generoituun skriptiin. Pidemmälle vietynä metatyylisivujen avulla voisi periaatteessa käsitellä konfigurointitiedostojen lisäksi myös kokonaisia XSL-skriptejä, mitä voisi pitää XML-maailman vastineena itseään muokkaavalle koodille.

4.3 Kielen ilmaisuvoimasta

Ilmaisuvoiman kannalta XSLT:n keskeisin puute on muunnossääntöjen kutsumuodon (ja yleisemmin XPath-lausekkeiden rakenteen) staattisuus, mikä on käsitelty luvussa 3.3. Yksittäisten funktioiden osalta ongelmaa voidaan kiertää nimiavaruuksia käyttämällä, mutta laajemmilla skripteillä tai useille eri XML-kielille suunnatuilla muunnoksilla tämä on perustavanlaatuinen ongelma. XSLT 2.0:ssa (Kay, 2007) ongelma on kuitenkin korjattu määrittelemällä uusi rakenne `<xsl:function>`, jota voidaan kutsua suoraan mistä tahansa XPath-lausekkeesta.

Toinen XSLT:n ongelma on puufragmenttien rajoitettu käsittely (katso luku 2.3.1). Tätä voidaan kiertää laajennusfunktiolla `exsl:node-set`[37], joka muuttaa puufragmentin solmujoukoksi, mahdollistaen esim. muuttujan arvon jatkokäsittelyn (samaan tapaan kuin muuttuja olisi alkuperäisestä dokumentista XPath-lausekkeella haettu solmujoukko). Kaikki tärkeimmät XSLT-prosessorit tukevat laajennusta, mutta kutsukäytännössä on pientä vaihtelua eri prosessoreilla, mikä monimutkaistaa XSL-koodia. Tämäkin ongelma on korjattu XSLT 2.0:ssa: erillinen puufragmenttityyppi on poistettu kokonaan, jolloin muuttujien arvot ovat suoraan solmujoukkomuodossa (Kay, 2007).

Novatchevin FXSL-kirjasto osoittaa XSLT:n olevan ”aito”, Turing-täydellinen ohjelmointikieli (Turingin koneesta on jopa XSLT-toteutus[38]), vaikka kieltä ei sellaiseksi ole suunniteltukaan – jo syntaksinsa takia sitä ei varmastikaan kannata käyttää kovin monimutkaisiin laskentatehtäviin, ja on selvää, ettei XSLT:tä kannata edes harkita, ellei suoritettava tehtävä vaadi nimenomaan XML-muotoisen datan käsittelyä. Kielen ilmaisuvoima antaa kuitenkin mahdollisuuden uusien funktiokirjastojen kehittämiseen ja käyttöön. Kielen ilmaisuvoimaa lisäävät myös exsl-yhteisön laajennukset ja erityisesti XSLT 2.0, jonka käyttöönottoa on hidastanut lähinnä puutteellinen tuki XSLT-prosessoreissa Saxonia lukuun ottamatta, ja kielen lisääntynyt monimutkaisuus. Tulevaisuudessa on kuitenkin syytä olettaa XSLT 2.0:n yleistyvän.

5 Yhteenveto

Seminaarityössä käsiteltiin XSLT-kieltä, sen käyttöä ja sovelluksia erityisesti ohjelmoinnin kannalta. XSLT:n käyttö edellyttää XML:n perusteiden ja XPath-lausekkeiden tuntemusta, jonka päälle kieli lisää puurakenteiden käsittelymallin ja omia kontrollirakenteitaan. XSLT on periaatteessa

[36] Opinto-oppaan jatkokehitysprojekti 2008

[37] `http://www.exslt.org/exsl/functions/node-set/`

[38] `http://www.unidex.com/turing/utm.htm`

Turing-täydellinen, funktiopohjainen ohjelmointikieli, mutta Turing-koneiden tapaan sen käyttäminen yleiskäyttöiseen ohjelmointiin ei ole mielekästä. Sen sijaan kevyeen (ja tässä käytyjä esimerkkejä raskaampaankin) XML-datan käsittelyyn kieli soveltuu hyvin – ei pelkästään kielen nimen viittaamaan ”tyylitykseen”. Jos on odotettavissa, että kehitettävä sovellus käyttää tai tuottaa XML:ää, tai sovelluksen tietojen käsittelylogiikasta on erotettavissa XML-muotoisen tiedon haku- tai muunnosketjuja, kannattaa XSLT pitää kandidaattina käytettäviä tekniikoita valitessa (luku 2). Työn esimerkeissä sivuttiin opinto-oppaan julkaisujärjestelmää, jonka keskeisimmät toiminnot perustuvat juuri XSL-muunnoksiin.

XSLT tarjoaa muihin XML-rajapintoihin nähden löyhästi muuhun sovellukseen kytketyn käsittelymekanismin, jolla määriteltyjä sääntöjä voidaan helposti siirtää eri ympäristöihin – esim. muunnettaessa XML:ää HTML-sivuiksi kieltä voidaan yksinkertaisimmillaan käyttää suoraan selaimessa ilman muita komponentteja. XSLT:n haasteita ovat työläs XML-syntaksi (jota korjaamaan on tosin esitetty useita yksinkertaistettuja syntakseja) ja varsinkin kielen 1.0-versiossa useat uudelleenkäyttöä rajoittavat tekijät (muunnossääntöjen kutsun staattisuus, puufragmenttien käsittelyn rajoitteet). Näistä rajoituksista on kuitenkin päästy eroon XSLT 2.0:ssa, ja jos tekniikkaa otetaan käyttöön, kannattaa ilman muuta käyttää suoraan uudempaa versiota – jyrkemmästä oppimiskäyrästä huolimatta tämä kannattaa skriptien parantuneen ylläpidettävyyden muodossa. XSLT:n funktiopohjaisuus voi vaatia orientoitumista, jos kehittäjät ovat tottuneet imperatiiviseen ohjelmointiin. Toisaalta funktio-ohjelmoinnin ideoiden ymmärtäminen on hyödyllistä kenelle tahansa kehittäjälle, ja on odotettavissa, että funktio-ohjelmoinnin merkitys kasvaa tulevaisuudessa entisestään (Kaijanaho, 2003, luku 1).

Lähteet

Bos, B. (2000). *How to add style to XML* (Tech. Rep.). W3C. http://www.w3.org/Style/styling-XML

Bray, T., Paoli, J., & Sperberg-McQueen, C. M. (1998). *Extensible markup language (XML) 1.0* (W3C Recommendation). W3C. http://www.w3.org/TR/1998/REC-xml-19980210

Clark, J. (1999). *XSL transformations (XSLT) version 1.0* (W3C Recommendation). W3C. http://www.w3.org/TR/1999/REC-xslt-19991116

Clark, J., & DeRose, S. (1999). *XML Path language (XPath) version 1.0* (W3C Recommendation). W3C. http://www.w3.org/TR/1999/REC-xpath-19991116

Cowan, J., & Tobin, R. (2004). *XML Information Set (second edition)* (W3C Recommendation). W3C. http://www.w3.org/TR/2004/REC-xml-infoset-20040204/

Eisenberg, J. D. (2005). *OASIS OpenDocument essentials – Using OASIS OpenDocument XML.* Friends of OpenDocument Inc.

Garlan, D., & Shaw, M. (1994). *An introduction to software architecture* (Tech. Rep. No. CMU-CS-94-166). http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf

Hautakangas, H., Koudelia, N., Lehtonen, J., & Nysten, J. (2009). *Joose-sovellusprojekti – sovellusraportti* (Tekninen raportti). Jyväskylän yliopisto. http://sovellusprojektit.it.jyu.fi/joose/dokumentit/sovellusraportti/sovellusraportti.pdf

Hernandez, G. (2010). *XPath.* Web page. http://plasmasturm.org/log/444/

Honkaranta, A., Rinne, P., & Nurminen, M. (2006). *Opinto-oppaan monikanavajulkaisu* (Tekninen raportti). Jyväskylän yliopisto. http://opinto-opas.jyu.fi/dokumentit/Xoo_jatko_raportti.pdf

Hughes, J. (1989). Why functional programming matters. *Comput. J.*, *32*(2), 98-107.

Jones, S. P. (Ed.). (2002). *Haskell 98 language and libraries – the revised report* (Tech. Rep.). haskell.org. http://www.haskell.org/onlinereport/

jQuery – API/1.1.2/DOM/Traversing/Selectors (jQuery documentation). (2007). The jQuery Project. http://docs.jquery.com/DOM/Traversing/Selectors

Kaijanaho, A.-J. (2003). *Funktio-ohjelmointi* (Tech. Rep.). Luentomoniste. http://users.jyu.fi/~antkaij/opetus/fo/2003/moniste/

Kay, M. (2007). *XSL transformations (XSLT) version 2.0* (W3C Recommendation). W3C. http://www.w3.org/TR/2007/REC-xslt20-20070123/

Kelly, S., & Tolvanen, J.-P. (2008). *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley Interscience.

Love, D. (2007). *Alternate syntax for XSLT* (Tech. Rep.). Rochester Institute of Technology. http://hdl.handle.net/1850/5292

Luk, R. W., Leong, H., Dillon, T. S., Chan, A. T., Croft, W. B., & Allan, J. (2002). A survey in indexing and searching XML documents. *Journal of the American Society for Information Science and Technology*, *53*(6), 415-437.

Mangano, S. (2005). *XSLT cookbook*. O'Reilly.

Novatchev, D. (2001). *The functional programming language xslt – a proof through examples* (Tech. Rep.). http://fxsl.sourceforge.net/articles/FuncProg/FunctionalProgramming.html

Pagaltzis, A. (2006). *How to map CSS selectors to XPath queries.* Web page. http://plasmasturm.org/log/444/

Rademacher, T., & Strachan, J. (2001). *dom4j cookbook* (Tech. Rep.). MetaStuff Ltd. http://dom4j.sourceforge.net/dom4j-1.6.1/cookbook.html

Stayton, B. (2007). *DocBook XSL: The complete guide – fourth edition*. Sagehill Enterprises.

Tittel, E. (2005). *XSLT expression variables and data types* (tutorial). searchSOA.com. http://searchsoa.techtarget.com/tip/XSLT-expression-variables-and-data-types

Tverskov, J. (2010). *Transform XML to CSV with XSLT pipeline.* Web page. http://www.xmlplease.com/pipeline_xml2csv

Wilde, E. (2010). *XML foundations.* Course material. http://dret.net/lectures/xml/