Tampereen yliopisto

Simo Voutilainen

SANAKIRJATIETOMALLIN TOTEUTUS XML-MUODOSSA

Informaatioteknologian ja viestinnän tiedekunta
Pro gradu -tutkielma
Huhtikuu 2021

TIIVISTELMÄ

Simo Voutilainen: Sanakirjatietomallin toteutus XML-muodossa
Pro gradu -tutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Huhtikuu 2021

Piirrerakenne on yksinkertainen tiedonkuvausmenetelmä, jolla on mahdollista esittää hierarkkisia rakenteita. XML on merkintäkieli, jota käytetään laajasti monenlaisten dokumenttien tuottamiseen. Tässä tutkielmassa pohditaan XML-kielen soveltuvuutta piirrerakenteiden esittämiseen.

Tutkielmassa esitellään FSXML-säännöt, joilla mikä tahansa piirrerakenne voidaan muuntaa XML-muotoon riippumatta piirrerakenteen syvyydestä tai laajuudesta. FSXML-sääntöjä sovelletaan piirrerakenteeseen, joka on tarkoitettu sanakirjan sisällön kuvaamiseen. Tuloksena syntyvän DTD-tiedoston eli dokumenttityypin määrittelyn käyttökelpoisuutta testataan todellisella sanakirjalla. Sanakirjan sisältö muunnetaan DTD:n mukaiseen muotoon sekä siitä edelleen HTML-muotoon. Muunnokset tehdään XSL-kielisillä muunnossäännöillä, joiden toimintaa kuvataan esimerkkien avulla rivi riviltä.

Sanakirjasisällön kuvaamiseen on olemassa useita XML-standardeja ja muita XML-pohjaisia ratkaisuja. Tutkielmassa esitellään niistä muutama ja verrataan niitä tutkielmassa kehitettyyn dokumenttityyppiin. Vertailussa todetaan dokumenttityypin samankaltaisuus TEI-standardin sanakirjaosuuden kanssa, mikä viittaa FSXML-sääntöjen käyttökelpoisuuteen.

Avainsanat: XML, XSL, XSLT, DTD, FSXML, sanakirja, piirrerakenne.

Sisältö

1 Johdanto

Nancy Ide ja kumppanit hahmottelivat vuonna 1993 artikkelissaan Outline of a model for lexical databases [Ide *et al*. 1993] hierarkkisen mallin sanakirjasisällön esittämiseen. Malli perustuu Iden ja Véronisin aiempaan artikkeliin [Véronis & Ide 1992]. Tekijät esittävät mallinsa piirrerakenteena (*feature structure*). Tutkimus on tehty tietokantatoteutuksen näkökulmasta, mutta itse malli on riippumaton kuvausmenetelmästä. Tässä pro gradu -työssä toteutan mallin XML-muodossa mahdollisimman alkuperäisenä, mitä ei taustakartoituksen perusteella ole aiemmin tehty. Tarkoitukseni on näin esittää, miten piirrerakennetietomalli toteutetaan XML-muotoisena. Osana tätä toteutusta määrittelen säännöt, joilla mikä tahansa piirrerakennemalli voidaan muuntaa XML-muotoon. Olen antanut säännöille nimen FSXML (*Feature Structure as XML*). Taustakartoituksessa en ole löytänyt tällaisia yleisiä sääntöjä, joten oletan että sellaisia ei ole aiemmin määritelty.

Iden ja kumppanien piirrerakennemallissa tietoalkiot voivat esiintyä rekursiivisesti sisäkkäin. Tämän rekursion avulla malli mahdollistaa tiedon esittämisen hierarkiana. Lähtöoletukseni on, että puumaisen rakenteensa ansiosta XML soveltuu hierarkkisen rakenteen esittämiseen. Piirrerakennemalli mahdollistaa myös faktoroinnin. Faktoroinnissa tietoalkion sisältö voidaan määritellä muutenkin kuin atomisena arvona, esimerkiksi listana, taulukkona tai relaationa, mikä vähentää tiedon redundanssia. Tähänkin tarkoitukseen oletan XML:n soveltuvan hierarkkisuutensa ansiosta.

Määrittelen piirrerakennemalliin perustuvan XML-dokumenttityypin DTD-tiedostona. Elementtien ja attribuuttien nimet noudattavat pääsääntöisesti Iden ja kumppanien käyttämiä termejä, mutta lisään myös muutamia termejä elementeille, joita Ide ja kumppanit eivät mainitse artikkelissaan – artikkelin ei ole tarkoituskaan esittää täydellistä rakennetta sanakirjasisällölle, vaan ennemminkin periaatteet sopivan rakenteen luomiseen.

Osoitan myös, että näin luotua dokumenttityyppiä voidaan käyttää todellisen sanakirjasisällön esittämiseen. Tähän tarkoitukseen käytän esimerkkisisältönä Juhani Norrin sanakirjaa *Dictionary of Medical Vocabulary in English, 1375-1550* [Norri 2016]. Sanakirjan sisältö on haettu tietokannasta ja tallennettu XML-muotoiseksi Jukka Ala-Fossin [2021] kehittämällä ohjelmistolla. Kehitän XSLT-muunnoksen, jolla muunnan tämän raakamuotoisen XML-sisällön piirrerakennemallin mukaiseksi XML:ksi. Lisäksi osoitan, että näin aikaansaatua sisältöä voidaan myös muuntaa edelleen käytettävämpään muotoon. Tätä varten kehitän toisen XSLT-muunnoksen, jolla muunnan piirrerakennemallin mukaisen XML-sisällön HTML:ksi.

Lopuksi vertailen omaa XML-toteutustani kolmeen vastaavaan toteutukseen: toiseen piirrerakennemallia hyödyntävään toteutukseen, yleiseen piirrerakennetoteutukseen sekä

Text Encoding Initiative -standardin osiossa 9 esiteltyyn XML-muotoiseen sanakirjatoteutukseen.

2 Aiempi aiheeseen liittyvä tutkimus

Sanakirjojen esittämiseen digitaalisessa muodossa on käytetty monenlaisia menetelmiä. Esittelen niitä lyhyesti tässä luvussa.

2.1 Rakenteellinen jaottelu

2.1.1 Typografiset merkintäjärjestelmät

Ennen kuin sanakirjasisällön digitaaliseen tuotantoon kehitettiin standardeja, sanakirjojen tuotannossa käytettiin tekstinkäsittely- ja taitto-ohjelmien merkintätapoja, joissa typografinen informaatio oli tekstin osana. Näiden merkintäjärjestelmien käyttö oli yleistä noin vuoteen 1995 asti [Debus-Gregor & Heid 2013]. Niiden päätarkoitus oli määritellä tekstin ulkoasua: erilaisilla merkkiyhdistelmillä ilmaistiin esimerkiksi kursiivia, lihavointia tai kirjasintyypin tai -koon vaihtumista. Iden ja kumppanien esimerkki Collins English Dictionaryn käyttämästä merkintätavasta on hakusanan *gin* sanakirja-artikkeli:

```
*Cgin*E*S1*E (d*3T*3Fn) *Fn. *%brew *5Q*A1. *Ean alcoholic drink obtained
by distillation and rectification of the grain of malted barley, rye, or
maize, flavoured with juniper berries. *%brew *5Q*A2. *Eany of various
grain spirits flavoured with other fruit or aromatic essences: *Fsloe
gin. *%brew *5Q*A3. *Ean alcoholic drink made from any rectified spirit.
*5Q*5HC18: shortened from Dutch *Fgenever *Ejuniper, via Old French form
Latin *Fj=u-niperus*Gjuniper*E*5I<
```

Kuva 1. Typografinen merkintä sanakirjatekstin osana [Ide *et al*. 1993]

Tällainen merkintätapa on hankalasti tulkittava, eikä se jäsennä sisältöä semanttisesti eli merkityksen mukaan, joten sen perusteella on vaikea hahmottaa minkälaista tietoa (hakusana, määritelmä, esimerkki, viittaus ja niin edelleen) yksittäinen tekstin osa sisältää. Lisäksi samaa merkintää, esimerkiksi kursiivin symbolia, voidaan käyttää monen erilaisen tiedon yhteydessä.

2.1.2 Hypertekstit

Hypertekstillä, kuten HTML-muotoisella dokumentilla, on sanakirjan tuotannossa samat puutteet kuin typografisilla merkintätavoilla: se ei jäsennä sisältöä semanttisesti, ja samaa elementtiä voidaan käyttää monen erilaisen sisällön merkitsemiseen.

Sen sijaan hyperteksti soveltuu hyvin sanakirjan käyttöliittymän formaatiksi, sillä painettu sanakirja muistuttaa tietyillä tavoilla hypertekstiä: kumpaakaan ei lueta alusta loppuun, vaan siitä etsitään vain halutut kohdat [Storrer 2013]. Lisäksi ristiviitteet yhdistävät sanakirjan eri osia kuten hypertekstin linkit. Storrer kutsuu tämäntyyppisiä, painetussa sanakirjassakin esiintyviä linkkejä objektiivisiksi linkeiksi. Hypertekstissä linkit voivat lisäksi olla kaksisuuntaisia, tai sanan määritelmän jokainen sana voi toimia linkkinä kyseisen sanan omaan määritelmään. Storrerin nimitys tällaiselle, vain digitaalisessa sanakirjassa esiintyvät linkille on subjektiivinen linkki. Linkit voidaan

myös jaotella sisäisiin ja ulkoisiin linkkeihin sen mukaan, osoittavatko ne sanakirjan sisältöön vai johonkin ulkoiseen kohteeseen [Storrer 2013]. Tässä työssä toteuttamani HTML-muotoisessa sanakirjassa linkit ovat ylläolevan luokittelun mukaan objektiivisia ja sisäisiä.

Hyperteksti voi myös sisältää multimediaa: yksi ensimmäisistä hypertekstisanakirjoista oli viittomakielen sanakirja [Storrer 2013]. Toinen hypertekstin etu painettuun sanakirjaan verrattuna on interaktiivisuus. Käyttäjä voi esimerkiksi muokata näkymää, jossa sisältöä esitetään [Storrer 2013].

2.1.3 SGML

SGML (*Structured General Markup Language*) on 1986 julkaistu ISO-standardi, joka perustuu 1970-luvun alusta asti käytössä olleisiin aiempiin merkintäkieliin [ISO 8879:1986]. Se määrittelee säännöt, joiden mukaan voidaan luoda merkintäkieliä tarpeen mukaan erilaisille dokumenttityypeille. SGML:n mukainen merkintäkieli määrittelee joukon elementtejä, joilla dokumentin sisältö merkataan. Elementeillä on alku- ja lopputunniste, jotka ilmaisevat, millaista sisältöä niiden välissä on. Elementit voivat sisältää paitsi tekstiä, myös muita elementtejä, mikä mahdollistaa puumaisen rakenteen määrittelyn. HTML ja XML, jotka perustuvat SGML:ään, käyttävät elementtejä näiden samojen periaatteiden mukaisesti.

Amsler ja Tompa julkaisivat vuonna 1988 SGML-pohjaisen sanakirjamerkintäkielen [Amsler & Tompa 1988], joka otettiin pian mukaan Text Encoding Initiative (TEI) -standardiin [Ide *et al*. 1993]. Heidän SGML-kielensä mukainen sanan *apple* määritelmä on esimerkissä 1:

```
<ME pos=n-a>
 <F>
  <orth hyph=2>apple</orth>
  <pron stress=PU syl=2>ap&schwa;l</pron></F>
 <E>
  <es>
   <etymon lang=ME>appel</etymon></es>
  <es>
   <rel>fr.</rel>
   <etymon lang=OE>&aelig;ppel</etymon></es>
  <es>
   <rel>akin to</rel>
   <eu>
    <etymon lang=OHG>apful</etymon>
    <deftext>apple</deftext></eu>
   <eu>
    <etymon lang=OSlav>abl&breve;ko</etymon></eu></es></E>
 <M>
  <S sn=1>
   <deftext>the fleshy usu. rounded and red or yellow edible pome
    fruit of a tree <taxon lev=G>Malus</taxon> of the rose family
   </deftext>
   <deftext type=also>an apple tree</deftext></S>
  <S sn=2>
```

```
 <deftext>a fruit or other vegetable production suggestive of an
  apple</deftext>
</S></M></ME>
```

Esimerkki 1. Sanan *apple* määritelmä SGML-muodossa [Amsler & Tompa 1988].

Esimerkissä 1 elementti **ME** (*main entry*) sisältää ylimmän tason sanakirja-artikkelin. Attribuutti **pos** (*part of speech*) tarkoittaa sanaluokkaa, ja sen arvo **n-a** tarkoittaa substantiivia (*noun*) joka voi toimia myös toisen sanan määritteenä (*attributive*). Elementti **F** (*form*) sisältää sanan ulkoasun kuvauksen, **E** etymologian, **M** (*meaning*) merkitykset ja **S** (*sense*) yksittäisen merkityksen. Elementtien nimet on siis valittu kuvaamaan niiden sisältöä. Tässä mielessä Amslerin ja Tompan merkintäkieltä voidaan pitää myös semanttisena mallina.

2.1.4 XML

XML (*Extensible Markup Language*) on World Wide Web Consortiumin standardi, jonka ensimmäinen versio esiteltiin vuonna 1998. XML on SGML:n osajoukko [World Wide Web Consortium 1998], ja senkin tarkoituksena on määritellä säännöt, joiden avulla voidaan luoda uusia merkintäkieliä. XML kehitettiin helpottamaan tällaisten kielten luomista. XML:n rakenne perustuu elementteihin samalla tavalla kuin SGML:n. XML on kuitenkin levinnyt laajempaan käyttöön kuin SGML, ja sen tueksi on kehitetty tiedonhaku- ja -käsittelykieliä, kuten XSL ja XQuery.

Tarkempi kuvaus XML-standardista ja XSL-kielestä on luvussa 3, ja esimerkkejä XML-muotoisesta sisällöstä on runsaasti myöhemmillä sivuilla.

2.1.5 Tietokannat

Relaatiotietokantoja on käytetty sanakirjojen tuotannossa jo ainakin 1980-luvulla, jolloin myös huomattiin niiden rajoitukset hierarkkisten rakenteiden esittämisessä [Neff *et al*. 1988]. Vaquero ja kumppanit [2013] kuitenkin pitävät relaatiotietokantoja erinomaisena välineenä sanakirjojen tuotantoon, kunhan sisällön vaatima tietomalli ja sen muokkaamiseen tarkoitettu käyttöliittymä on riittävän huolellisesti suunniteltu. He luettelevat tietokantajärjestelmien eduiksi ainakin operaatioiden atomaarisuuden ja samanaikaisuuden, sisäänrakennetut mekanismit virhetilanteista toipumiseen, automaattiset eheystarkistukset sekä samanaikaisten käyttäjien tuen. Nämä edut pätevät varsinkin sanakirjan kirjoitusvaiheessa.

Ide ja kumppanit [1993] luettelevat relaatiotietokantojen haittapuolia sanakirjasisällön tuotannossa. Käyn niitä läpi tarkemmin luvussa 4, jossa myös alustavasti esittelen, miten kyseiset ongelmat voidaan välttää XML:ää käyttämällä.

2.2 Semanttinen jaottelu

2.2.1 Piirrerakenteet

Piirrerakenne (*feature structure*) koostuu yksinkertaisimmillaan ominaisuudesta eli attribuutista ja sille annetusta arvosta. Esimerkiksi piirrerakenteessa [orth: elephant] sana **orth** (*orthography*, kirjoitusasu) on sanan ominaisuus ja **elephant** on sen arvo. Arvo voi itsekin olla piirrerakenne, mikä mahdollistaa rekursiivisesti luodut hyvinkin laajat hierarkkiset rakenteet. Lisäksi mahdollisten arvojen joukkoa voidaan rajoittaa määrittelemällä piirteelle tyyppi: esimerkiksi sanaluokka-tyyppinen piirre voi saada arvot **substantiivi** tai **verbi**, mutta ei arvoa **feminiini** tai **maskuliini**.

Piirrerakenteita on käytetty sanakirjasisällön esittämiseen tekoälyyn ja luonnollisten kielten käsittelyyn liittyvissä sovelluksissa [Trippel 2013].

2.2.2 Text Encoding Initiative

Text Encoding Initiative (TEI) on 1987 perustettu yhteenliittymä. Se on määritellyt TEI-standardin, jolla erityisesti humanististen alojen tekstejä voidaan esittää digitaalisessa muodossa. TEI-standardia on käytetty tai käytetään yli 200 projektissa [Text Encoding Initiative: Projects Using the TEI 2020].

Kohdassa 2.1.3 mainittu Amslerin ja Tompan SGML-pohjainen sanakirjamerkintäkieli on sittemmin korvattu TEI:ssä XML-pohjaisella merkintäkielellä, joka muodostaa TEI-standardin osan 9 Dictionaries [Text Encoding Initiative: Dictionaries 2020]. Dictionaries-osa mainitsee alaviitteessään Iden ja kumppanien artikkelin aiheeseen liittyvänä aiempana tutkimuksena. TEI:n tietomalli on myös paikoin hyvin samankaltainen kuin piirrerakennemalli (katso esimerkki 27), joten oletan sen Dictionaries-osan olevan jossain määrin piirrerakennemallin toteutus. Ainakin Tutin ja Véronis vahvistavat tämän: ”…the TEI DTD is largely based on the lexical database model proposed by Ide, Le Maitre & Véronis, 1993…” [Tutin & Véronis 1998]. Se eroaa kuitenkin minun tekemästäni toteutuksesta siten, että sen perustana on käytetty muitakin malleja, ja se on siten selvästi laajempi kuin minun toteutukseni.

TEI määrittelee kolme erilaista periaatetta sanakirjatiedon esittämiseen: typografinen (*typographic*), toimituksellinen (*editorial*) ja leksikaalinen (*lexical*). Typografinen esitys pyrkii kuvaamaan painetun sanakirjan typografian mahdollisimman tarkasti, ja se sisältää jopa kahdelle riville jaetun sanan tavuviivan. Toimituksellinen esitys ei ota kantaa typografiaan, mutta sisältää kaiken painettavan tekstin sellaisenaan. Typografinen ja toimituksellinen esitys ovat siis yksittäisen sanakirjajulkaisun kuvauksia. Kolmas esitystapa, leksikaalinen, on näistä kolmesta yleiskäyttöisin, koska se sisältää ainoastaan sanakirjan tekstisisällön ja loogisen rakenteen. Kaksi muuta muotoa voidaan generoida siitä sopivien sääntöjen avulla esimerkiksi XSL-kielellä. XML-pohjaisessa julkaisutoiminnassa on yleisenä periaatteena erottaa sisältö ja muoto toisistaan, ja näistä kolmesta TEI-standardin esitysmuodosta ainoastaan leksikaalinen noudattaa tätä

periaatetta. Tässä työssä toteutan piirrerakennemallin nimenomaan leksikaalista periaatetta noudattaen. Luvussa 7 esittelen, miten lopputulos eroaa TEI-standardin Dictionaries-merkintäkielestä.

2.2.3 ISO 1951

ISO 1951 -standardin ensimmäinen version on vuodelta 1973, mutta sitä on uudistettu viimeksi 2007. Sen alkuperäinen tarkoitus oli yhtenäistää sanakirjojen tuotannossa käytettyjä typografisia merkintäjärjestelmiä, mutta se on sittemmin laajentunut yleisemmäksi sanakirjasisällön tietomalliksi [Lemnitzer *et al*. 2013]. Toisin kuin TEI, se ei sisällä käytännön toteutusta vaan on abstrakti tietomalli: ”It specifies a formal generic structure independent of the publishing media…” [ISO 1951:2007].

Lemnitzer ja kumppanit toteavat ISO 1951:n sisältävän esimerkiksi rekursiivisuuden kannalta turhia tasoja, eivätkä pidä ISO 1951:tä kovin käyttökelpoisena standardina käytännön sovellusten rakentamiseen: "… an incomplete design which makes it hardly usable in concrete applications". Heidän mukaansa TEI on tehnyt ISO 1951:stä käytännössä tarpeettoman, ja esittävät, että ISO 1951:n seuraava versio voisi viitata TEI-standardin XML-dokumenttityyppiin tietomallinsa referenssitoteutuksena. [Lemnitzer *et al*. 2013].

2.2.4 Lexical Markup Framework (LMF)

Lexical Markup Framework (LMF) eli ISO 24613:2008 on 2008 julkaistu ISO-standardi leksikaalisen datan esittämiseen luonnollisen kielen käsittelyä (*natural language processing*, NLP) varten [Francopoulo *et al*. 2007]. Perinteinen ihmislukijalle suunnattu sanakirja keskittyy sanan merkitysten luettelemiseen ja luokittelemiseen, mutta LMF mahdollistaa sen lisäksi sanan muidenkin ominaisuuksien, esimerkiksi sanan syntaktisen roolien, kuvaamisen. LMF:stä on pyritty tekemään mahdollisimman modulaarinen jakamalla se kahteen osaan:

1. Ydinosa, jonka avulla voidaan määrittää kuvattavan sanaston perusrakenne.
2. Laajennusosat, joilla sanastoa voidaan kuvailla tarkemmin. Esimerkiksi semanttinen laajennusosa mahdollistaa sanan merkitysten tarkemman määrittelyn ja luokittelun. Muita laajennusosia on muun muassa monikielisten sanastojen, morfologian, syntaktisten rakenteiden ja monisanaisten ilmaisujen kuvaamiseen [Francopoulo *et al*. 2007].

Kuten monet muutkin tässä luvussa kuvatut standardit, LMF hyödyntää muita ISO-standardeja attribuuttien arvojoukkojen määrittelyyn. Esimerkiksi kielten lyhenteinä LMF käyttää ISO 639:n mukaisia arvoja. Vaikkapa TEI:stä poiketen LMF kuitenkin hakee myös attribuuttien nimiä muista ISO-standardeista:”…attribute-value pairs like /grammatical gender/ and /feminine/ are not defined within LMF. They are to be taken from the ISO Data Category Registry as specified by ISO-12620” [Francopoulo *et al*. 2007]. Sanan suvulle ei siis ole LMF:ssä eksplisiittisesti nimetty omaa attribuuttia. Sen

sijaan TEI:ssä on vastaavalle tiedolle määritelty oma elementti, <gen>. LMF poikkeaa siis TEI:stä siten, että se on abstraktimpi, tarkoitettu ylemmän tason rakenteiden määrittelyyn. LMF:n määrittelemä rakenne on kuitenkin mahdollista toteuttaa XML:nä, ja sekä Francopoulo ja kumppanit [2007] että Romary [2013] antavatkin esimerkin LMF:n mukaisesta XML-muotoisesta sanan määritelmästä.

3 Työssä käytetyt menetelmät

Tässä luvussa esittelen lyhyesti käyttämäni kuvaus- ja käsittelymenetelmät.

3.1 XML

XML (*Extensible Markup Language*) on tiedonesitysformaatti, joka on sitä edeltäneen SGML (*Structured Generalised Markup Language*) –kielen osajoukko. XML on perinyt SGML:ltä kohdassa 2.1.3 esitellyt perusperiaatteet: kaikki XML:llä esitettävä sisältö on jaettu elementeiksi, esimerkiksi `<case>accusative</case>`. Elementit koostuvat alkutunnisteesta (`<case>`) ja lopputunnisteesta (`</case>`) sekä niiden välissä olevasta sisällöstä. Elementin nimi yleensä kuvaa sisältöä. Edellä olevan elementin nimi on **case** (sijamuoto) ja sisältö *accusative* (akkusatiivi). Kuten SGML:ssä, myös XML:ssä elementti voi sisältää muita elementtejä:

```
<sense>
 <def>medicine alleviating pain</def>
</sense>
```

Esimerkki 2. **Sense**-elementti sisältää **def**-elementin, joka sisältää sanan merkityksen. Esimerkin tekstisisältö on lähteestä [Norri 2016].

Elementin alkutunniste voi sisältää yhden tai useampia attribuutteja, esimerkiksi **order=”1”** elementissä `<orth order="1">abater</orth>`. Attribuutti koostuu nimestä ja arvosta, ja se sisältää elementtiä koskevaa lisätietoa. Attribuutti ei voi sisältää muita attribuutteja, vaan sen arvo on aina pelkkä merkkijono. Koska sanalla **attribuutti** voidaan tarkoittaa myös piirrerakenteen ominaisuutta ja tietokannan kentän nimeä, kutsun tässä työssä XML-kielen attribuuttia XML-attribuutiksi, ellei merkitys muuten selviä asiayhteydestä.

XML muistuttaa ulkoisesti SGML:n tunnetumpaa sovellusta, HTML:ää (*HyperText Markup Language*). Niiden olennaisin ero on, että HTML-kieli määrittelee HTML-sisällössä käytettävät elementit ja attribuutit, mutta XML sallii käyttäjän määritellä ne itse. Tämä määrittely voidaan tehdä yksinkertaisesti vain käyttämällä haluttuja elementtejä ja attribuutteja XML-tiedostossa, tai eksplisiittisesti DTD- tai XML Schema-syntaksilla. Jos XML-tiedosto noudattaa XML-standardia, se sanotaan olevan hyvin muodostettu.

XML on World Wide Web Consortiumin määrittelemä standardi, jonka ensimmäinen versio julkaistiin 1998 [World Wide Web Consortium 1998].

3.2 DTD

DTD (D*ocument Type Definition*) on syntaksi, jolla määritellään tietty dokumenttityppi rakenteisille dokumenteille, kuten SGML- tai XML-dokumenteille. DTD-tiedosto määrittelee, mitä elementtejä ja attribuutteja kyseinen dokumenttityppi voi sisältää, missä järjestyksessä ne voivat esiintyä, ja mitkä elementit voivat sisältää toisia elementtejä:

```
<!ELEMENT entry (form, etymology?, additional?, homograph+,
entry*)>
<!ELEMENT form (orth+, hyph*, pron*)>
<!ELEMENT orth (#PCDATA)>
<!ATTLIST orth geo CDATA #IMPLIED order CDATA #IMPLIED>
```

Esimerkki 3. Osa dictionary.dtd-tiedostosta.

Esimerkin 3 ensimmäinen rivi määrittelee, että elementti **entry** sisältää, tässä järjestyksessä, täsmälleen yhden **form**-elementin, sen jälkeen **etymology**-, **additional**-, ja **homograph**-elementtejä korkeintaan yhden kappaleen kutakin, ja sen jälkeen **entry**-elementtejä kuinka monta tahansa. **Etymology**-elementti on tarkoitettu sanan etymologian eli alkuperän esittämiseen, **additional**-elementti sisältää lisätietoa sanan alkuperästä ja **homograph**-elementti sisältää yksittäisen homograafin tiedot. Homograafit ovat sanoja, joilla on keskenään sama kirjoitusasu, mutta joiden merkitys on niin erilainen, että niitä voidaan pitää erillisinä sanoina.

Esimerkin 3 toinen rivi määrittelee, että **form**-elementti voi sisältää yhden tai useampia **orth**-elementtejä, sen jälkeen **hyph**-elementtejä kuinka monta tahansa, ja lopuksi **pron**-elementtejä kuinka monta tahansa. **Orth**-elementti on tarkoitettu sanan kirjoitusasun esittämiseen, **hyph**-elementti sisältää sanan tavutussäännön, ja **pron**-elementti sanan ääntämisohjeen.

Esimerkin 3 kolmas rivi määrittelee, että **orth**-elementti voi sisältää tekstiä mutta ei elementtejä. Neljäs rivi määrittelee, että **orth**-elementillä voi olla attribuutit **geo** ja **order**, jotka kumpikin sisältävät vapaamuotoista tekstiä. **Geo**-attribuutti kertoo, millä maantieteellisellä alueella tiettyä kirjoitusasua käytetään. **Order**-attribuutin avulla eri kirjoitusasuille voidaan määritellä haluttu järjestys.

Attribuuttien arvojoukko voidaan myös määritellä eksplisiittisesti DTD:ssä. Esimerkiksi `gend (f|m|n) #IMPLIED` määrittelee, että **gend**-attribuutti voi saada vain arvot **f**, **m** tai **n**.

Jos XML-tiedosto noudattaa tiettyä dokumenttityyppiä, se ilmoitetaan tiedoston alussa olevalla dokumenttityypin deklaraatiolla: `<!DOCTYPE dictionary SYSTEM "dictionary.dtd">`. Tämä kertoo XML-tiedostoa käsitteleville ohjelmille, mihin DTD:hen tiedostoa pitää verrata. Jos XML-tiedosto noudattaa DTD:n määrittelyjä, se on validi; muussa tapauksessa se on epävalidi.

3.3 XSL

XSL (*Extensible Stylesheet Language*) on World Wide Web Consortiumin määrittelemä standardi, jonka ensimmäinen versio julkaistiin suosituksena 2001 [World Wide Web Consortium 2001]. XSL on XML-sisällön muokkaamiseen tarkoitettu kieliperhe, joka sisältää kolme kieltä:

1. XPath, jonka avulla voidaan viitata XML-sisältöön ja siten tehdä sisältöhakuja tai kohdistaa prosessointi tiettyyn kohtaan XML-rakenteessa.

2. XSLT, jonka avulla XML-sisältö voidaan järjestää uudeksi rakenteeksi tai uuteen tiedostoformaattiin.
3. XSL-FO, jonka avulla XSLT-muunnoksen tulokselle voidaan määritellä haluttu ulkoasu. [World Wide Web Consortium 2017]

XSL on itsekin XML:n mukainen merkintäkieli, joten XSL-standardin mukaiset tiedostot ovat myös valideja XML-tiedostoja. XML-syötteen käsittely on XSL-tiedostossa järjestetty xsl:template-elementeiksi. Jokainen xsl:template-elementti määrittelee säännön jonkin XML-syötteen elementin tai muun osan käsittelemiseksi:

```
<xsl:template match="term">
 <entry>
  <xsl:apply-templates/>
 </entry>
</xsl:template>
```

Esimerkki 4. XSLT-sääntö, joka käsittelee syötteen sisältämän **term**-elementin.

Esimerkin 4 sääntö käsittelee XML-syötteen **term**-elementit. Jokainen syötteen **term**-elementti tuottaa tulosteeseen **entry**-elementin, jonka alku- ja lopputunniste määritellään säännössä sellaisinaan. Niiden välissä oleva xsl:apply-templates-elementti käynnistää **term**-elementin sisällön käsittelyn kyseiselle sisällölle määriteltyjen sääntöjen mukaisesti. Käytännössä se kutsuu muita xsl:template-elementtejä sen mukaan, mitä elementtejä **term**-elementti sisältää. Näin **term**-elementin sisältö siis tulostuu **entry**-elementin sisällöksi.

Vaikka XSL on nimellisesti tyylisivukieli, se sisältää myös elementtejä, jotka mahdollistavat proseduraalisen ohjelmoinnin, esimerkiksi xsl:if, xsl:when ja xsl:otherwise, sekä xsl:for-each. XSL:llä voi määritellä myös kutsuttavia funktioita xsl:function-elementeillä tai nimetyillä xsl:template-elementeillä.

XPath-lausekkeet voivat esiintyä XSLT-säännössä tiettyjen attribuuttien arvoina. Esimerkiksi ”term” elementissä `<xsl:value-of select=”term”/>` on yksinkertainen XPath-lauseke, mutta XPath-lausekkeet voivat olla paljon mutkikkaampiakin:

```
//term[@identifier='87']/preceding-sibling::term[1]/termText
```

Esimerkki 5. XPath-lauseke, joka viittaa tiettyyn kohtaan XMLrakenteessa.

Esimerkin 5 XPath-lauseke tarkoittaa ”sellaisen **term**-elementin lapsielementti **termText**, joka edeltää millä tahansa tasolla olevaa **term**-elementtiä, jonka **identifier**-attribuutti sisältää tekstin '87'”.

XSL-FO-kieli koostuu elementeistä, jotka määrittelevät halutun ulkoasun XSL-muunnoksen lopputulokselle:

```
<xsl:template match="term">
 <entry>
  <fo:block font-style=”italic” color=”grey” keep-with-
   next.within-page=”always”>Edellinen hakusana:
   <xsl:value-of select="preceding-sibling::term[1]/termText">
  </fo:block>
  <xsl:apply-templates/>
 </entry>
</xsl:template>
```

Esimerkki 6. XSL-FO -elementti XSLT-säännön osana.

Esimerkin 6 fo:block-elementti määrittelee, että sen sisältö muotoillaan kappaleeksi, jonka kirjasintyyli on kursiivi ja väri harmaa, ja joka tulostetaan aina samalle sivulle sitä seuraavan kappaleen kanssa. XSL-FO -kielessä on paljon yhteistä CSS-kielen kanssa, kuten esimerkin 6 color- ja font-style -määrittelyt.

Jotta XSL-tiedostossa määritelty muunnos saadaan aikaan, tarvitaan XSL-prosessori. Se on sovellus, jolle XML-sisältö ja XSL-säännöt annetaan syötteenä, ja joka tuottaa XSL-sääntöjen määräämän tuloksen. Tulos voi olla jotakin toista XML-kieltä, HTML-kieltä, tekstiä tai vaikkapa Word-dokumentti. Tässä työssä käytän Saxon-prosessorin versiota 9.1.0.3J.

Jos XSL-säännöt sisältävät XSL-FO-elementtejä, XSL-prosessori tuottaa FO-tiedoston, joka pitää vielä muuntaa haluttuun loppuformaattiin, esimerkiksi PDF-tiedostoksi. Tätä varten tarvitaan vielä erillinen FO-prosessori. Yleisesti käytettyjä prosessoreita ovat esimerkiksi FOP ja AH Formatter. En ole tässä työssä käyttänyt XSL-FO-kieltä, koska kaikkien XSL-muunnosten lopputulokset ovat joko XML- tai HTML-kielisiä.

4 Piirrerakennemalli

Ide ja kumppanit [1993] esittelevät erilaisia tapoja, joilla sanakirjasisältöä on esitetty elektronisessa muodossa artikkelin julkaisua edeltäneenä aikana. He myös mainitsevat näiden tapojen haittapuolia. Koska heidän esittämänsä malli pyrkii ratkaisemaan nämä ongelmat, käyn seuraavassa lyhyesti läpi niistä tärkeimmät ja pohdin, missä määrin samat ongelmat koskevat XML:ää ja XSL:ää.

Kohdassa 2.1.3 esittelin lyhyesti Amslerin ja Tompan [1988] kehittämän SGML-pohjaisen merkintäkielen, jolla sanakirjasisältöä jäsennetään elementtien avulla. Ide ja kumppanit toteavat tämän esitystavan haittapuoleksi, että tällaiseen rakenteeseen tehtävät haut ovat helposti hyvin mutkikkaita, jos haun pitää löytää tietoa tekstirakenteen useammasta osasta: ”The software has no knowledge of the structure of the text, and so searches which involve elements whose relationship is embodied in the structure of the dictionary entry can become prohibitively complex” [Ide *et al.* 1993]. Tämä haittapuoli on korjattavissa käyttämällä XML:ää ja XSLT:tä: mutkikkaitakin tekstihakuja voidaan tehdä melko lyhyillä XSLT-lausekkeilla. Koko XML-dokumentti on jäsennetty DOM-puuksi, joten XPath-viittauksella on näkyvyys koko rakenteeseen kerralla.

Relaatiotietomallin ongelmiksi Ide ja kumppanit mainitsevat tiedon fragmentoitumisen ja redundanssin, jotka kumpikin johtuvat siitä, että eri tietokanta-attribuuttien mahdollisten arvojen lukumäärä vaihtelee suuresti eri sanakirja-artikkeleiden välillä: ”…entries may include several different pronunciations, parts of speech, orthographic variants…” [Ide *et al.* 1993]. Fragmentoitumisen ja redundanssin välillä joudutaankin heidän mukaansa usein valitsemaan jompikumpi: tiedon toistoa voidaan vähentää jakamalla tieto useampaan relaatioon, jolloin fragmentaatio lisääntyy. Jos taas halutaan vähentää fragmentoitumista, tietoa pitää kerätä samaan relaatioon, mikä vaatii tiedon toistoa. XML:ssä tällaista ongelmaa ei synny, koska yksittäiseen artikkeliin voidaan lisätä tarvittava määrä saman elementin esiintymiä.

Relaatiotietomalli soveltuu Iden ja kumppanien mukaan myös huonosti hierarkkisten merkitysten esittämiseen. Vaquero ja kumppanit tosin kyseenalaistavat tämän näkemyksen. Heidän mukaansa huolellisesti suunniteltu relaatiotietokanta kyllä mahdollistaa esimerkiksi tiedon periytymisen rakenteen alemmille tasoille sekä tiedon hierarkian [Vaquero *et al*. 2013]. XML soveltuu puumaisen ja rekursiivisen rakenteensa ansiosta hierarkioiden esittämiseen erityisen hyvin.

Ide ja kumppanit toteavat, että hierarkkisia merkityksiä voidaan jossain määrin esittää NF^2 (*non-first normal form*) -relaatiomallilla. NF^2-relaatiomallissa tietoalkio voi sisältää joko yksittäisen atomisen arvon tai itsenäisen relaation [Roth *et al*. 1988]. Tämä mahdollistaa tiedon faktoroinnin, mikä helpottaa sanakirjasisällön esittämistä hierarkkisessa muodossa. Malli ei kuitenkaan salli rekursiivisia rakenteita, esimerkiksi sisäkkäisiä sanojen määritelmiä. Tällaiset alitermien määritelmät ovat yleisiä varsinkin

tesaurus-tyyppisissä sanakirjoissa. Vaikka samantyyppinen tieto siis esiintyisi hierarkian useammalla tasolla, tiedon sisältävällä kentällä pitää olla eri nimi joka tasolla. Tämä vaatimus mutkistaa sisältöhakuja, koska hauissa pitää ottaa huomioon kenttien eri nimet eri tasoilla, vaikka joka tasolta haettaisiin samaa tietoa. Osittain sama rajoitus estää poikkeusrakenteiden esittämisen: jos esimerkiksi yhdellä sanan merkityksellä on poikkeava ääntämisohje, NF^2-relaatiomalli ei salli saman tietokanta-attribuutin käyttämistä eri tasoilla. [Ide *et al.* 1993]

XML:ssä ei ole kumpaakaan rajoitusta, koska samannimisiä elementtejä voidaan sallia käytettäväksi millä tahansa rakenteen tasolla. Esimerkiksi pääsanan alle voidaan määritellä alitermi käyttäen rekursiivisesti samaa rakennetta kuin pääsanalle:

```
<entry>
 <form>…</form>
 <etymology>…etymology>
 <homograph>…</homograph>
 <entry>
  <form>…</form>
  <etymology>…</etymology>
  <homograph>…</homograph>
 </entry>
</entry>
```

Esimerkki 7. Sisäkkäiset **entry**-elementit.

Kolmas Iden ja kumppanien mainitsema NF^2-relaatiomallien rajoitus liittyy erilaisten sanakirjojen yhdistämiseen. Eri sanakirjoissa sisältö on järjestetty erilaisilla periaatteilla, mistä seuraa, että niiden yhdistäminen on työlästä ja vaatii sisällön uudelleenjärjestämistä. Tämä ongelma voidaan kiertää määrittelemällä eri sanakirjatyypeille yhteinen rakenne, mutta NF^2-relaatiomallit ovat siihen liian jäykkiä, koska niiden faktorointi pitää määritellä tietylle sanakirjatyypille. XML mahdollistaa joustavamman rakenteen määrittelyn, ja XSLT:llä voidaan automatisoida muunnokset eri sanakirjatyyppien välillä.

4.1 Piirrerakennemuotoinen sanakirjamalli

Iden ja kumppanien oman mallin pohjana on piirrerakenne-tietomalli (*feature structure*), jossa tietokanta-attribuutin arvo voi olla atomaarinen tai sisältää useita muita attribuutti-arvo -pareja. Iden ja kumppanien esimerkki tätä piirrerakennemallia noudattavasta sanakirja-artikkelin esityksestä on kuvassa 2. Siinä tietokanta-attribuuttien **form**, **gram** ja **def** arvoina on kullakin oma piirrerakenteensa, ja piirrehierarkian alimman tason attribuuttien (**orth**, **hyph** ja niin edelleen) arvo on atomaarinen, käytännössä merkkijono tai yksittäinen merkki.

com•peti•tor /k@m'petɪt@(r)/ *n* person who competes [OALD]

```
[      [orth: competitor   ]
 form: [hyph: com.peti.tor ]
       [pron: k@m'petIt@(r)]
 gram:[pos:  n]
 def: [text: person who competes]]
```

Kuva 2. Sanan *competitor* sanakirja-artikkelin esitys piirrerakenteena [Ide *et al.* 1993].

Kuvan 2 sanakirjalainauksessa teksti **com·peti·tor** on sanakirja-artikkelin hakusana, johon on merkitty myös sanan tavutus. /k@m'ptIt@(r)/ on sanan ääntämisohje, *n* (*noun* eli substantiivi) sanaluokka, ja teksti "person who competes" sanan merkitys. [OALD] viittaa lähteeseen, Oxford Advanced Learner's Dictionary -sanakirjaan.

Ide ja kumppanit lisäävät piirrerakenne-tietomalliin useita ominaisuuksia. Ensimmäinen lisäyksistä on disjunktio: tietokanta-attribuutin arvo voi olla järjestetty tai järjestämätön joukko, jotka kumpikin voivat sisältää piirrerakenteita tai atomaarisia arvoja. Esimerkkinä he käyttävät sanan eri kirjoitusasuja:

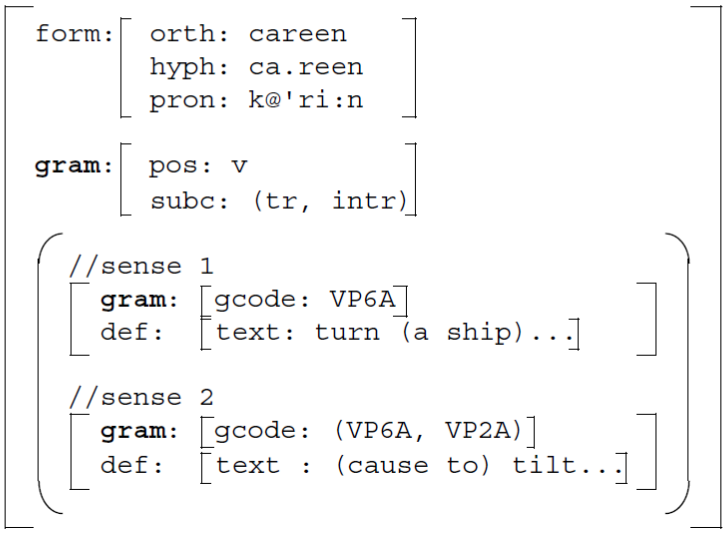
biryani *or* **biriani** (ˌbɪrɪ'A:nɪ) *n*. Any of a variety of Indian dishes... [CED]

```
[form: [orth: (biryani, biriani)]
       [pron: ,biri'A:ni       ]
 gram: [pos:  n]
 def:  [text: Any of a variety  ]
       [      of Indian dishes...]]
```

Kuva 3. Disjunktio: attribuutin orth kaksi vaihtoehtoista atomaarista arvoa [Ide *et al.* 1993].

Ide ja kumppanit laajentavat piirrerakenne-tietomallia myös osittaisfaktoroinnilla (*partial factoring*) ja arvon ohittamisella (*overriding*). Nämä ominaisuudet muistuttavat toisiaan: kummassakin sama tietokanta-attribuutti voi esiintyä sisäkkäisillä hierarkian tasoilla. Ominaisuuksien ero on tavassa, jolla eri tasojen arvoja tulkitaan yhdessä: osittaisfaktoroinnissa kaikkien tasojen arvot ovat voimassa yhtä aikaa, ohittamisessa vain sisimmän tason arvo on voimassa. Iden ja kumppanien esimerkki osittaisfaktoroinnista, sanan kieliopillinen luokittelu, on kuvassa 4.

ca•reen /k@'ri:n/ *vt,vi* **1** [VP6A] turn (a ship) on one side for cleaning, repairing, etc. **2** [VP6A, 2A] (cause to) tilt, lean over to one side. [OALD]



Kuva 4. Osittaisfaktoroitu kieliopillinen luokittelu (gram) [Ide *et al.* 1993]

Kuvan 4 sanakirjalainauksessa sanan *careen* sanaluokka *vt* (*verb, transitive*) tarkoittaa transitiivista verbiä eli verbiä joka vaatii objektin, ja *vi* (*verb, intransitive*) intransitiivista verbiä eli verbiä joka voi esiintyä lauseessa ilman objektia. [VP6A] on sanakirjan omaa, tarkempaa luokittelua. Piirrerakenteessa **s**anan *careen* ensimmäisen merkityksen luokittelu on yhdistelmä ylimmän tason ja sense 1 –tason gram-attribuuttien arvoista:

pos: v
subc: (tr, intr)
gcode: VP6A

Iden ja kumppanien esimerkki arvon ohittamisesta, poikkeavan ääntämisohjeen esittäminen, on kuvassa 5.

con•jure /'k^ndG@(r)/ *vt,vi* **1** [VP2A,15A] do clever tricks which appear magical... **2** [VP15B] ~ ***up,*** cause to appear as if from nothing... **3** /k@n'dGW@(r)/ [VP17] (formal) appeal solemnly to... [OALD]

```
form: [ orth: conjure
        hyph: con.jure
        pron: "kVndZ@(r) ]

gram: [ pos: v
        subc: (tr, intr) ]

( //sense 1
  [ gram: [ gcode: (VP2A, VP15A) ]
    def:  [ text: do clever tricks... ] ]

  //sense 2
  [ gram:    [ gcode: VP15B ]
    related: [ orth : conjure up ] ]

  //sense 3
  [ form:  [ pron: k@n"dZU@(r) ]
    gram:  [ gcode: VP17 ]
    usage: [ reg: formal ]
    def:   [ text : appeal solemnly to... ] ]

  ... )
```

Kuva 5. Ääntämisohjeen (pron) arvon ohittaminen alemmalla tasolla [Ide *et al.* 1993]

Nämä kaksi lisäominaisuutta ovat siis rakenteeltaan hyvin samankaltaisia: ainoa rakenne-ero on, että osittaisfaktoroinnissa eri tasoilla esiintyvä samanniminen tietokanta-attribuutti on piirrerakenne, kun taas arvon ohittamisessa se on atomaarinen arvo. Ide ja kumppanit eivät kuitenkaan kerro, onko tämä ero määräävä tekijä, vai onko mahdollista käyttää näitä rakenteita myös päinvastaisessa tarkoituksessa. Kohdassa 5.4 esitän arvon ohittamiselle XML-toteutuksen, jossa tämä ongelma on ratkaistu.

5 Piirrerakennemallin XML-toteutus

Taustakartoituksessani yritin löytää yleisiä sääntöjä, joilla mielivaltainen piirrerakenne voidaan muuntaa sisältötyypin mukaiseen XML-muotoon. Hakulauseke *"feature structure*" AND conver* AND XML* ei kuitenkaan löytänyt sellaisia Andorista, ACM:stä, ProQuestista eikä IEEE/IET Electronic Librarysta, joten oletan, että sellaista säännöstöä joko ei ole tai se ei ole yleisesti tunnettu. Siksi olen itse kehittänyt säännöt, joilla mikä tahansa piirrerakenne voidaan muuntaa XML-muotoon. Olen nimennyt säännöt FSXML:ksi (*Feature Structure as XML*). Esittelen ne tässä luvussa ja sovellan niitä tässä työssä Iden ja kumppanien piirrerakenteeseen. Tämän luvun kuvat ja XML-esimerkkien sanakirjasisältö ovat lähteestä [Ide *et al.* 1993], ellei esimerkin yhteydessä ole mainittu toisin.

FSXML-säännöt tuottavat aina hyvin muodostetun XML-rakenteen, joka sisältää kaiken tiedon muunnetusta piirrerakenteesta. Näiden sääntöjen tuottama rakenne ei kuitenkaan välttämättä aina ole paras mahdollinen: piirrerakenne on yleensä mahdollista muuntaa useiksi erilaisiksi XML-rakenteiksi, ja käyttötarkoituksesta riippuu, mikä niistä kannattaa valita.

Selostan myös, miten toteutan piirrerakennemallin XML-muotoisena. Toteutus pohjautuu seuraavaksi esiteltäviin FSXML-sääntöihin.

5.1 Perusmuotoinen piirrerakenne

Perusmuotoinen piirrerakenne on yksinkertaista muuntaa XML-muotoon. Piirrerakenteen attribuutti-arvo -pari muunnetaan XML-elementiksi seuraavalla säännöllä:

(1) Piirrerakenne [A:v], jossa v on atomaarinen arvo, muunnetaan XML-rakenteeksi `<A>v</A>`.

Esimerkiksi kuva 2 sivulla 15 ja toistettuna alla sisältää piirrerakenteen [text: person who competes]. Ylläolevilla säännöillä se muunnetaan XML-elementiksi <text>person who competes</text>.

Jos piirrerakenteen attribuutin arvo on itsekin piirrerakenne, koko rakenne muunnetaan sisäkkäisiksi XML-elementeiksi seuraavalla säännöllä:

(2) Piirrerakenne [A:C], jossa C = [A':C'], muunnetaan XML-rakenteeksi `<A><A'>C'</A'></A>`. Jos C' on piirrerakenne, siihen sovelletaan sääntöjä (1) ja (2).

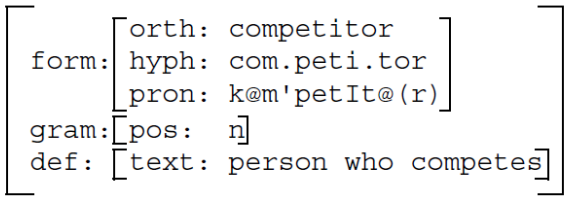
Kuvan 2 piirrerakenteessa [def: [text: person who competes]] ulompi attribuutti on **def**, ja sen arvona on sisempi attribuutti-arvo -pari, jossa attribuutti on **text**. Sääntöjen (1) ja (2) mukaisesti tämä piirrerakenne muunnetaan XML-rakenteeksi `<def><text>person who competes</text></def>`.

Jos piirrerakenteen ylimmällä tasolla on vain yksi attribuutti, se voidaan muuntaa XML:ksi säännöllä (2). Jos niitä on useampia, pitää XML-rakenteelle valita mielivaltainen juurielementti. Tässä työssä käytän yksittäisen sanakirja-artikkelin juurielementtinä elementtiä **entry**. Jos piirrerakenteen ylimmällä tasolla on enemmän kuin yksi attribuutti, se muunnetaan XML-elementiksi seuraavalla säännöllä:

(3) Piirrerakenteen ylin taso [C], missä C = vähintään kaksi piirrerakennetta, muunnetaan XML-rakenteeksi `<entry>C</entry>`, missä **entry** on piirrerakennetta kuvaava nimi. C muunnetaan XML:ksi muiden sääntöjen avulla.

Näillä kolmella säännöillä voidaan muuntaa mielivaltaisen syvä piirrerakenne vastaavan syvyiseksi XML-rakenteeksi. Esimerkiksi kuvan 2 koko rakenne voidaan tätä periaatetta käyttäen muuntaa XML-muotoiseksi esimerkin 8 mukaisesti:

com•peti•tor /k@m'petɪt@(r)/ *n* person who competes [OALD]



Kuva 2. Sanan *competitor* sanakirja-artikkelin esitys piirrerakenteena.

```
<entry>
 <form>
  <orth>competitor</orth>
  <hyph>com.peti.tor</hyph>
  <pron>k@m'petIt@(r)</pron>
 </form>
 <gram>
  <pos>n</pos>
 </gram>
 <def>
  <text>person who competes</text>
 </def>
</entry>
```

Esimerkki 8. Kuvan 2 piirrerakenteen esitys XML-muodossa.

Tätä XML-esitystä vastaava DTD-määrittely on seuraava:

```
<!ELEMENT entry (form, gram, def>
<!ELEMENT form (orth, hyph, pron)>
<!ELEMENT orth (#PCDATA)>
<!ELEMENT hyph (#PCDATA)>
<!ELEMENT pron (#PCDATA)>
<!ELEMENT gram (pos)>
<!ELEMENT pos (#PCDATA)>
<!ELEMENT def (text)>
<!ELEMENT text (#PCDATA)>
```

Esimerkki 9. Esimerkkiä 8 vastaava DTD-määrittely.

Pos (*part of speech*) -elementti sisältää tiedon sanan sanaluokasta. Koska sanaluokkia on melko vähän, olisi hyödyllistä sallia **pos**-elementille vain tietyt arvot. DTD-syntaksi ei salli tätä, mutta jos dokumenttityyppi määritellään XML Schema -kielellä, sallittujen arvojen joukkoa voidaan rajoittaa määrittelemällä sille oma tietotyyppi xs:restriction-elementin avulla [World Wide Web Consortium 2012].

5.2 Disjunktio

Disjunktio eli useasta osasta koostuva attribuutin arvo voidaan ilmaista XML:llä esimerkiksi toistamalla tarvittavaa elementtiä. Jos piirrerakenteen attribuutin arvo on disjunktio, se muunnetaan peräkkäisiksi samannimisiksi XML-elementeiksi seuraavasti:

(4) Piirrerakenteen [A:D], jossa D on disjunktio, jokainen C' ∈ D muunnetaan XML-rakenteeksi `<A>C'</A>`. Jos C' on piirrerakenne, siihen sovelletaan sääntöjä (1) ja (2).

Kuvan 3 esimerkki disjunktiosta on orth: (biryani, biriani). Disjunktion muunnossäännöllä siitä saadaan XML-esitys `<orth>biryani</orth><orth> biriani</orth>`.
Tämä rakenne on seuraavassa esimerkissä, jossa **orth**-elementillä esitetään vaihtoehtoiset kirjoitusasut:

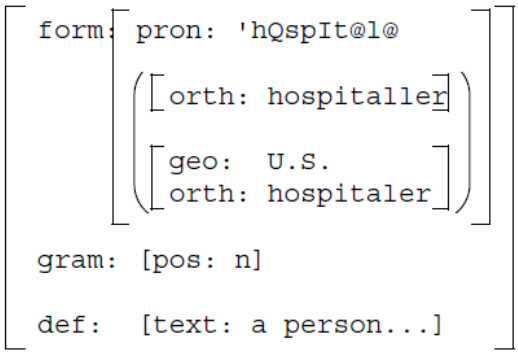
```
<entry>
 <form>
  <orth>absconsio</orth>
  <orth>absconcionem</orth>
  <orth>absconcis</orth>
 </form>
…
</entry>
```

Esimerkki 10. Useita rinnakkaisia orth-elementtejä. Esimerkin tekstisisältö on lähteestä Norri [2016].

Samannimiset rinnakkaiset elementit saadaan järjestettyä joko implisiittisesti tulkitsemalla elementtien esiintymisjärjestys joukon järjestykseksi tai eksplisiittisesti määrittelemällä toistettavalle elementille XML-attribuutti, jonka arvoksi annetaan kyseisen elementin järjestysnumero joukossa: `<orth order=”1”>absconsio</orth>`. On myös mahdollista määritellä vain yksi elementti XML-attribuutin avulla ensisijaiseksi kirjoitusasuksi ja jättää muut, toissijaiset kirjoitusasut keskenään samanarvoisiksi.

Disjunktio voi esiintyä myös samalla tasolla kuin atomaariset arvot, kuten kuvassa 6.

hospitaller *or U.S.* **hospitaler** ('hɑspIt@l@) *n.* a person, esp. a member of certain religious orders... [CED]



Kuva 6. Disjunktio (**orth** sekä **geo** + **orth**) atomaarisen arvon (**pron**) rinnalla.

Atomaarisen arvon esiintyminen samalla tasolla disjunktion kanssa ei välttämättä vaadi mitään erityistä XML-rakennetta: jos kuvan 6 rakenne halutaan muuntaa XML-muotoon, **orth**-elementtejä voidaan jälleen lisätä tarvittava määrä **form**-elementin sisälle kuten esimerkissä 10, siis samalle tasolle kuin **pron**-elementti. Kuvan 6 tapauksessa tilannetta kuitenkin mutkistaa attribuutti geo, joka liittyy ainoastaan jälkimmäiseen orth-attribuuttiin. Tämä kytkös pitää pystyä ilmaisemaan myös XML:ssä esimerkiksi ylimääräisellä elementtitasolla. Jos disjunktion osat sisältävät keskenään erilaisia piirrerakenteita, osat muunnetaan XML:ksi seuraavasti:

(5) Disjunktion D, jonka osilla on keskenään erilainen rakenne, jokainen C' ∈ D muunnetaan XML-rakenteeksi `<alt>C'</alt>`. Jos C' on piirrerakenne, siihen sovelletaan sääntöjä (1), (2) ja (4).

Tämän säännön avulla kuvan 6 piirrerakenne muunnetaan XML:ksi seuraavasti:

```
<form>
 <pron>'hQspIt@l@</pron>
 <alt>
  <orth>hospitaller</orth>
 </alt>
 <alt>
  <geo>U.S.</geo>
  <orth>hospitaler</orth>
 </alt>
</form>
```

Esimerkki 11. Disjunktio (**orth**) atomaarisen arvon (**pron**) rinnalla.

Kuvan 6 disjunktiosta saadaan kuitenkin aikaan yksinkertaisempi rakenne, kun siirretään geo-tieto XML-attribuuttiin:

```
<form>
 <pron>'hQspIt@l@</pron>
 <orth>hospitaller</orth>
 <orth geo="U.S.">hospitaler</orth>
</form>
```

Esimerkki 12. **geo** XML-attribuuttina.

Elementin **orth** DTD-määrittely on tällöin seuraava:

```
<!ELEMENT orth (#PCDATA)>
<!ATTLIST orth
        geo CDATA #IMPLIED
        order CDATA #IMPLIED>
```

Esimerkki 13. **orth**-elementin sekä **geo**- ja **order**-attribuuttien DTD-määrittely.

5.3 Osittaisfaktorointi

Osittaisfaktorointi voidaan toteuttaa XML:nä toistamalla osittaisfaktoroitavaa elementtiä kaikilla tarvittavilla hierarkian tasoilla. Tällainen rakenne on tulkittava siten, että kullakin tasolla ovat voimassa sekä kyseisen tason että kaikkien sitä ylempien tasojen osittaisfaktoroitujen elementtien arvot. Sivulla 16 olevan kuvan 4 sisältö voidaan XML-muotoisena esittää esimerkiksi seuraavasti:

```
<entry>
 <form>
  <orth>careen</orth>
  <hyph>ca.reen</hyph>
  <pron>k@'ri:n</pron>
 </form>
 <gram>
  <pos>v</pos>
  <subc>tr</subc>
  <subc>intr</subc>
 </gram>
 <sense>
  <gram>
   <gcode>VP6A</gcode>
  </gram>
  <def>
   <text>turn (a ship)…</text>
  </def>
 </sense>
 <sense>
  <gram>
   <gcode>VP6A</gcode>
   <gcode>VP2A</gcode>
  </gram>
  <def>
   <text>(cause) to tilt…</text>
  </def>
 </sense>
<entry>
```

Esimerkki 14. Sanan yksittäisen merkityksen kieliopillinen luokittelu (**gram**) jaettuna artikkelitasolle (**entry**) ja merkitystasolle (**sense**).

Kuvan 4 piirrerakenne sisältää disjunktioita attribuuteissa **subc** ja **gcode**, joten ne on esimerkissä 14 muunnettu XML-elementeiksi sääntöjen (4) ja (5) avulla.

5.4 Arvon ohittaminen

Kuten osittaisfaktorointi, myös arvon ohittaminen toteutetaan XML:ssä toistamalla ohitettava elementti tarvittavilla tasoilla. Toisin kuin osittaisfaktoroinnissa, kullakin

tasolla on voimassa vain kyseisen tason elementti, jonka arvo siis ohittaa ylempien tasojen arvot. Sivulla 17 olevan kuvan 5 sisällön XML-muotoinen esitys on esimerkiksi tällainen:

```
<entry>
 <form>
  <orth>conjure</orth>
  <hyph>con.jure</hyph>
  <pron>”kVndZ@(r)</pron>
 </form>
 <gram>
  <pos>v</pos>
  <subc>tr</subc
  <subc>intr</subc
 </gram>
...
 <sense>
  <form>
   <pron>k@n”dZU@(r)</pron>
  </form>
  <gram>
   <gcode>VP17</gcode>
  </gram>
  <usage>
   <reg>formal</reg>
  </usage>
  <def>appeal solemnly to…</def>
 </sense>
<entry>
```

Esimerkki 15. Yhdellä sanan merkityksistä on oma ääntämisohje, joka ohittaa artikkelitason ääntämisohjeen. Vertaa sivun 17 kuvaan 5.

Osittaisfaktorointi ja arvon ohittaminen siis toteutetaan hyvin samankaltaisilla rakenteilla, joten sekaannuksen mahdollisuus on olemassa. Kuten kohdassa 4.1 totesin, Ide ja kumppanit eivät esitä arvon ohittamiselle ja osittaisfaktoroinnille muuta rakenteellista eroa kuin sen, että arvon ohittamisessa on kyse atomaarisesta arvosta, ja osittaisfaktoroinnissa piirrerakenteesta. Voi kuitenkin syntyä tarve toimia päinvastoin: hierarkian alemmalla tasolla halutaankin ohittaa piirrerakenteen arvo tai halutaan osittaisfaktoroida atomaarista arvoa. Yksittäisellä sanan merkityksellä voi esimerkiksi olla vaihtoehtoinen ääntämisohje, joka ei ohitakaan sanan yleistä ääntämisohjetta. Tällöin esimerkin 15 mukainen rakenne pitäisi pystyä tulkitsemaan osittaisfaktoroinniksi, jotta kumpikin ääntämys olisi mahdollinen.

Tilannetta voidaan selventää lisäämällä sisemmän tason elementtiin tieto siitä, ohittaako sen sisältö ylemmän tason vai eikö. Seuraavassa esimerkissä tämä tieto on **pron**-elementin attribuutissa **override**:

```
<sense order=”3”>
 <form>
  <pron override=”yes”>k@n”dZU@(r)</pron>
 </form>
 <gram>
```

```
  <gcode>VP17</gcode>
 </gram>
 <usage>
  <reg>formal</reg>
 </usage>
 <def>appeal solemnly to…</def>
</sense>
```

Esimerkki 16. Ääntämisohjeeseen (**pron**) on lisätty attribuutti **override**, joka ilmaisee, että se ohittaa ylempien tasojen ääntämisohjeet. **Sense**-elementin attribuutti **order=”3”** ilmaisee, että tämä on sanan kolmas merkitys. Vertaa esimerkkiin 15.

Viimeiset FSXML-muunnossäännöt ovat siis seuraavat:

(6) Osittaisfaktoroitu piirrerakenne [A:C] muunnetaan XML-rakenteeksi `<A override="no">C</A>`. Jos C on piirrerakenne, siihen sovelletaan sääntöjä (1), (2), (4) ja (5).

(7) Piirrerakenne [A:C], joka ohittaa aiemman arvon, muunnetaan XML-rakenteeksi `<A override="yes>C</A>`. Jos C on piirrerakenne, siihen sovelletaan sääntöjä (1), (2), (4) ja (5).

Override-attribuutti pitää DTD:ssä sallia kaikille elementeille, joita halutaan osittaisfaktoroida tai joiden arvo halutaan ohittaa. Esimerkkinä tästä **pron**-elementin DTD-määrittely:

```
<!ELEMENT pron (#PCDATA)>
<!ATTLIST pron
        override (yes|no) #IMPLIED>
```

Esimerkki 17. **pron**-elementin sekä **override**-attribuutin DTD-määrittely.

6 XSL-toteutus

Tässä luvussa osoitan, että piirrerakennemallin XML-toteutuksella voidaan esittää todellista sanakirjasisältöä. Käytän tähän tarkoitukseen sanakirjaa *Dictionary of Medical Vocabulary in English, 1375-1550* (edempänä DMV) [Norri 2016], josta on peräisin myös tämän luvun XML-esimerkkien sanakirjasisältö. Sanakirjan XML-esitystä varten olen kehittänyt piirrerakennemallista DTD-muotoisen dokumenttityypin määrityksen edellisessä luvussa esitetyillä FSXML-säännöillä. Olen nimennyt DTD:n dictionary.dtd:ksi, ja se on liitteessä 1. DTD:ssä olen päätynyt joissakin tapauksissa hieman luvun 5 esimerkeistä poikkeaviin rakenteisiin. Esimerkiksi monessa luvun 5 esimerkissä esiintyy rakenne `<def><text>…</text></def>`, mutta koska **text**-elementti ei anna mitään lisätietoa sisällöstä, olen jättänyt sen pois ja sallinut tekstisisällön suoraan **def**-elementissä. Toinen poikkeus on **homograph**-elementti: luvun 5 esimerkeissä **gram**- ja **sense**-elementit esiintyivät suoraan **entry**-elementin lapsina. Saman hakusanan alla voi kuitenkin olla homograafeja eli täysin erillisiä sanoja, joilla vain sattuu olemaan sama kirjoitusasu. Eri homograafien merkityksiä ei ole järkevää esittää samalla tasolla rinnakkain, koska sellainen rakenne viittaa saman sanan eri merkityksiin. Siksi olen lisännyt **entry**- ja **sense**-tasojen väliin kolmannen tason, **homograph**-elementin:

```
<!ELEMENT entry (form+, etymology?, additional?, homograph+,
entry*)>
<!ELEMENT homograph (gram?, sense*, related*)>
```

Esimerkki 18. **Entry**- ja **homograph**-elementtien määrittely dictionary.dtd:ssä.

Käyttämäni sanakirjasisältö on käytännössä XML-muotoinen tietokantatuloste, joka on tuotettu Jukka Ala-Fossin [2021] kehittämällä ohjelmistolla. Tämä tiedosto on siis kohdassa 6.1 kuvatun XSL-muunnoksen syötetiedosto, ja muunnoksen tuloksena syntyy dictionary.dtd:n mukainen XML-tiedosto, jonka rakenne noudattaa piirrerakennemallia.

Kohdassa 6.2 osoitan, että dictionary-dokumenttityypin mukainen XML-sisältö voidaan muuntaa HTML-muotoon. Tässä toisessa XSL-muunnoksessa syötetiedosto on siis kohdassa 6.1 kuvatun XSL-muunnoksen lopputulos, ja muunnoksen tuloksena syntyy HTML-tiedosto.

Ensimmäisen XSL-muunnoksen syötetiedostona toimivassa tietokantatulosteessa on 266857 riviä, ja sen koko on 8,73 megatavua. Tulosteen sisältö on tesaurus-muotoinen, toisin sanoen sen hakusanat on järjestetty termeiksi ja alitermeiksi sisäkkäisille tasoille, joita on enimmillään neljä. Termejä on yhteensä 12608, joista ylimmällä tasolla on 4487. Tulosteen XML-rakenne on melko yksinkertainen, ja olen tehnyt sitä kuvaavan DTD-määrittelyn, thesaurus.dtd:n, joka on liitteessä 2. Yksittäisen tietokantatulosteessa olevan sanakirja-artikkelin tyypillinen rakenne on esitetty esimerkissä 19.

```
<term identifier="4274">
 <termText>fraction</termText>
 <etymology>OF $fraction$ ($FEW$ 3, 743b, $DMF$ s.v.) and ML
  $fraction-em$ ($DML$ s.v.).</etymology>
 <additional/>
 <senses>
  <sense identifier="70">
   <senseText>fracture of a bone</senseText>
   <description/>
  </sense>
 </senses>
 <variants>
  <variant identifier="5955">
   <variantText>fraction</variantText>
  </variant>
 </variants>
 <narrowerTerms>
  <term identifier="4273">
   <termText>fraction of the mind</termText>
   <etymology/>
   <additional/>
   <senses>
    <sense identifier="6656">
     <senseText>mental or emotional disturbance</senseText>
     <description/>
    </sense>
   </senses>
   <variants/>
   <narrowerTerms/>
  </term>
 </narrowerTerms>
</term>
```

Esimerkki 19. Sanan *fraction* määritelmä thesaurus.dtd:n mukaisena XML:nä.

Esimerkin 19 **etymology**-elementissä lyhenne OF (*Old French*) tarkoittaa keskiajan ranskaa ja ML (*Medieval Latin*) keskiajan latinaa. FEW, DMF ja DML ovat muiden sanakirjojen lyhenteitä. S.v. (*sub verbo*) on viittaus mainittuun hakusanaan toisessa sanakirjassa. Esimerkissä käytettyjen elementtien vastaavuus dictionary.dtd:n elementteihin käy ilmi taulukosta 1.

6.1 Muunnos thesaurus-muodosta dictionary-dokumenttityypin mukaiseen XML-muotoon

Muunnan thesaurus-dokumenttityypin mukaisen tietokantatulosteen dictionary-dokumenttityypin mukaiseen muotoon XSLT-muunnoksella DMV_to_dictionary.xsl, joka on liitteessä 3. XSLT-muunnoksen käynnistävä komento sekä selostus sen parametreista on esitetty liitteessä 5.

DMV_to_dictionary.xsl -muunnoksen syötetiedoston elementit vastaavat lopputuloksen elementtejä taulukon 1 esittämällä tavalla.

Syötetiedoston elementit, määritelty thesaurus.dtd:ssä	**Tulosteen elementit, määritelty dictionary.dtd:ssä**
thesaurus	dictionary
term	entry
termText	form/orth order="1"
variants/variant/variantText	form/orth
senses	homograph
sense	sense
senseText	def
etymology	etymology
additional	additional
description	description
$-merkki	b

Taulukko 1. Thesaurus- ja dictionary-dokumenttityyppien elementtien vastaavuudet.

Kuten taulukko 1 osoittaa, osa elementeistä säilyy samannimisinä, osa nimetään uudelleen. Elementin tunniste on kuitenkin vain yksi osa XML-dokumentin rakenteesta; olennaisempi osa on elementtien keskinäinen järjestys. Elementtien sisältöä järjestetään muunnoksessa uudelleen monin tavoin, jotka esittelen seuraavaksi.

6.1.1 Lajittelu aakkosjärjestykseen

Tietokantatulosteessa sanakirjan hakusanat ovat termText-elementeissä. Tuloste ei kuitenkaan ole termText-elementtien mukaisessa aakkosjärjestyksessä. Siksi thesaurus-elementin XSL-sääntö järjestää lähdetiedoston sisällön aakkosjärjestykseen xsl:sort-elementin avulla. **Thesaurus**-elementin käsittely on liitteessä 3 säännössä `<xsl:template match="thesaurus">`. Seuraavassa on säännön toiminta riveittäin esitettynä:

`<xsl:template match="thesaurus">`	Käsittele thesaurus-elementit näin:
`<dictionary>`	Tulosta dictionary-elementin alkutunniste.
`<xsl:variable name="content">`	Luo muuttuja nimeltä $content, jonka sisältö on seuraava:
`<xsl:apply-templates/>`	• kaikki thesaurus-elementin jälkeläiset käsiteltyinä omien sääntöjensä mukaan
`<xsl:if test="$narrowerTermEntry='yes'">`	• jos parametrin $narrowerTermEntry arvo on *yes* (katso kohta 6.1.4),
`<xsl:apply-templates select="term//term" mode="narrowerTermEntry"/>`	• kaikki term/narrowerTerms/term -rakenteen elementit käsiteltyinä erillisen sääntönsä mukaan
`</xsl:if>`	(elementin lopputunniste)

`</xsl:variable>`	(elementin lopputunniste)
`<xsl:for-each select="$content/*">`	Tee tämä jokaiselle muuttujan $content sisältämälle elementille:
`<xsl:sort select="lower-case(form/orth[1])"/>`	• lajittele elementin form/orth[1] mukaan
`<xsl:copy-of select="."/>`	• tulosta sisältö
`</xsl:for-each>`	(elementin lopputunniste)
`</dictionary>`	(elementin lopputunniste)
`</xsl:template>`	Tulosta dictionary-elementin lopputunniste.

Listaus 1. **Thesaurus**-elementin käsittely.

6.1.2 Variant-elementin käsittely

Joillakin sanoilla on vaihtoehtoisia kirjoitusasuja. Tämä oli yleistä varsinkin keskiajan englannin kielessä, kun vakiintuneita oikeinkirjoitussääntöjä ei vielä ollut. Kuten esimerkissä 19, tietokantatulosteessa sanan vaihtoehtoiset kirjoitusasut ovat **variants**-elementin sisällä erillään itse hakusanasta, joka puolestaan on **termText**-elementissä. Dictionary-dokumenttityypissä kaikki kirjoitusasut ovat kuitenkin **form**-elementin alla rinnakkaisina **orth**-elementteinä. Jotta tällainen rakenne saataisiin aikaan, **termText**-elementin käsittelyssä haetaan tulosteeseen myös **variants**-elementin sisältö:

Syötetiedosto (thesaurus-tiedosto)

```
<term identifier="32">
 <termText>absconsio
 </termText>
…
 <variants>
  <variant identifier="114">
   <variantText>absconcionem
   </variantText>
  </variant>
  <variant identifier="115">
   <variantText>absconcis
   </variantText>
  </variant>
 </variants>
</term>
```

Tuloste (dictionary-tiedosto)

```
<entry>
 <form>
  <orth order="1">absconsio
  </orth>
  <orth>absconcionem</orth>
  <orth>absconcis</orth>
 </form>
…
</entry>
```

Esimerkki 20. **TermText**- ja **variant**-elementtien muunnos **orth**-elementeiksi.

Esimerkin 20 kuvaama TermText-elementin käsittely on liitteessä 3 säännössä `<xsl:template match="termText">`. Seuraavassa on säännön toiminta riveittäin esitettynä:

`<xsl:template match="termText">`	Käsittele termText-elementit näin:
`<form>`	Tulosta form-elementin alkutunniste.
`<orth order="1">`	Tulosta orth-elementin alkutunniste ja anna sen order-attribuutin arvoksi 1.
`<xsl:apply-templates/>`	Tulosta termText-elementin sisältö.
`</orth>`	Tulosta orth-elementin lopputunniste.
`<xsl:for-each select="../variants/variant">`	Tee tämä jokaiselle rakenteen ../variants/variant sisältämälle elementille:

`<orth>`	• tulosta orth-elementin alkutunniste
`<xsl:apply-templates/>`	• tulosta sisältö
`</orth>`	• tulosta orth-elementin lopputunniste
`</xsl:for-each>`	(elementin lopputunniste)
`</form>`	Tulosta form-elementin lopputunniste.
`</xsl:template>`	(elementin lopputunniste)

Listaus 2. **TermText**-elementin käsittely.

Ensisijainen kirjoitusasu merkitään säännön rivillä 3 antamalla sen **orth**-elementin **order**-attribuutin arvoksi 1. Koska **variants**-elementin sisältö tulostettiin jo **termText**-säännön rivillä 8, tarvitaan lisäksi **variants**-elementille oma sääntö, joka ei tee mitään: `<xsl:template match="variants"/>`. Muuten XSLT:n oletussääntö tulostaisi **variants**-elementin sisällön uudestaan tekstimuodossa.

6.1.3 Tagify-sääntö

Tietokantatulosteessa sanat, joita halutaan korostaa, ovat $-merkkien välissä. Tämä merkintätapa muistuttaa kohdassa 2.1.1 esiteltyjä typografisia merkintöjä. Jos korostus halutaan säilyttää XML:ssä, $-merkit ja niiden välissä oleva teksti pitää muuttaa omaksi elementikseen. Tämä tapahtuu erillisessä rekursiivisessa säännössä, `<xsl:template name="tagify">`, joka on liitteessä 3. Sääntö vaihtaa saamansa syötteen kaksi ensimmäistä $-merkkiä **b**-elementin alku- ja lopputunnisteeksi ja kutsuu sitten itseään. Seuraavassa on säännön toiminta riveittäin esitettynä:

`<xsl:template name="tagify">`	
`<xsl:param name="input_string"/>`	Sääntö saa kutsussa syötteenä parametrin $input_string.
`<xsl:param name="char" select="'$'"/>`	Sääntö voi saada kutsussa syötteenä parametrin $char. Muuten parametrin arvo on ’$’.
`<xsl:param name="element" select="'b'"/>`	Sääntö voi saada kutsussa syötteenä parametrin $element. Muuten parametrin arvo on ’b’.
`<xsl:choose>`	
`<xsl:when test="contains( $input_string, $char)">`	Jos $input_string sisältää merkin $char,
`<xsl:value-of select="substring-before($input_string, $char)"/>`	• tulosta $input_string-merkkijonon alku merkin $char ensimmäiseen esiintymään asti
`<xsl:variable name="temp_string" select="substring-after( $input_string, $char)"/>`	• luo muuttuja nimeltä $temp_string, jonka arvo on $input_string-merkkijonon loppu merkin $char ensimmäisestä esiintymästä alkaen
`<xsl:element name="{$element}">`	• tulosta elementin $element alkutunniste
`<xsl:value-of select="substring-before($temp_string, $char)"/>`	• tulosta $temp_string-merkkijonon alku merkin $char ensimmäiseen esiintymään asti
`</xsl:element>`	• tulosta elementin $element lopputunniste

`<xsl:variable name="rest" select="substring-after($temp_string, $char)"/>`	• luo muuttuja nimeltä $rest, jonka arvo on $temp_string-merkkijonon loppu merkin $char ensimmäisestä esiintymästä alkaen
`<xsl:call-template name="tagify">`	• kutsu tagify-sääntöä näillä parametreilla:
`<xsl:with-param name="input_string" select="$rest"/>`	• parametrin $input_string arvo on $rest
`</xsl:call-template>`	(elementin lopputunniste)
`</xsl:when>`	(elementin lopputunniste)
`<xsl:otherwise>`	Muussa tapauksessa
`<xsl:value-of select="$input_string"/>`	• tulosta $input_string –parametrin sisältö
`</xsl:otherwise>`	(elementin lopputunniste)
`</xsl:choose>`	(elementin lopputunniste)
`</xsl:template>`	(elementin lopputunniste)

Listaus 3. **$**-merkkien muuntaminen **b**-elementin tunnisteiksi.

Tagify-sääntöä kutsutaan kaikkien sellaisten lähde-elementtien käsittelyssä, jotka voivat sisältää $-merkkejä: **etymology**, **additional**, **senseText** ja **description**.

6.1.4 NarrowerTermEntry-parametri

Tesaurus-muotoisessa sanakirjassa termit ja alitermit on järjestetty sisäkkäin yleisimmästä tarkimpaan. Tämä voi vaikeuttaa sanakirjan käytettävyyttä: jos käyttäjä etsii tiettyä alitermiä, hänen on tiedettävä, minkä yleistermin alla se esiintyy. Tämä hankaluus voidaan haluttaessa estää lisäämällä jokaiselle alitermille myös oma artikkelinsa sanakirjan ylimmälle tasolle. Artikkelin sisällöksi riittää ristiviittaus itse alitermiin, joka sisältää alitermin määritelmän ja muut tiedot. Tämä toiminto on toteutettu DMV_to_dictionary.xsl-muunnoksessa kaikille alitermeille, ja se voidaan kytkeä päälle antamalla XSL-muunnosta käynnistettäessä parametrin **narrowerTermEntry** arvoksi *yes*. Muussa tapauksessa alitermit tulostuvat ainoastaan pääterminsä alle, kuten ne ovat lähdetiedostossakin.

NarrowerTermEntry-parametrin arvo tarkistetaan säännössä `<xsl:template match="thesaurus">`. Jos parametrin arvo on *yes*, sääntö kutsuu term-elementin kaikkien tasojen term-jälkeläisille tehtyä sääntöä `<xsl:template match="term//term" mode="narrowerTermEntry">`, joka puolestaan kutsuu elementin termText vaihtoehtoista sääntöä `<xsl:template match="termText" mode="narrowerTermEntryTermText">`. Näiden toiminta on listauksessa 4 esitetty riveittäin, viimeksi mainitun säännön osuus sisennettynä.

<xsl:template match="term//term" mode="narrowerTermEntry">	Käsittele term-elementin term-jälkeläiset kaikilla tasoilla näin:
<entry>	Tulosta entry-elementin alkutunniste.
<xsl:apply-templates select="termText" mode= "narrowerTermEntryTermText"/>	Käsittele tämän rakenteen termText-lapsielementit omalla säännöllään:
<xsl:template match="termText" mode= "narrowerTermEntryTermText">	Käsittele termText-elementit näin:
<form>	Tulosta form-elementin alkutunniste.
<orth order="1">	Tulosta orth-elementin alkutunniste ja anna sen order-attribuutin arvoksi ”1”.
<xsl:apply-templates/>	Tulosta termText-elementin sisältö.
</orth>	Tulosta orth-elementin lopputunniste.
</form>	Tulosta form-elementin lopputunniste.
</xsl:template>	
<homograph>	Tulosta homograph-elementin alkutunniste.
<related type="see">	Tulosta related-elementin alkutunniste ja anna sen type-attribuutin arvoksi ”see”.
<xref>	Tulosta xref-elementin alkutunniste.
<xsl:value-of select= "ancestor-or-self::term[last()]/termText"/>	Tulosta tämän elementin isovanhemman term-Text-lapsen sisältö, eli tämän termin ylätermi.
</xref>	Tulosta xref-elementin lopputunniste.
</related>	Tulosta related-elementin lopputunniste.
</homograph>	Tulosta homograph-elementin lopputunniste.
</entry>	Tulosta entry-elementin lopputunniste.
</xsl:template>	

Listaus 4. Alitermien tulostus ylimmän tason **entry**-elementiksi.

Esimerkissä 19 esitettiin yksittäisen sanakirja-artikkelin rakenne tietokantatulosteessa. DMV_to_dictionary.xsl-muunnos tuottaa siitä seuraavan tuloksen:

```
<entry>
 <form>
  <orth order="1">fraction</orth>
  <orth>fraction</orth>
 </form>
 <etymology>OF <b>fraction</b> (<b>FEW</b> 3, 743b, <b>DMF</b>
  s.v.) and ML <b>fraction-em</b> (<b>DML</b> s.v.).</etymology>
 <homograph>
  <sense>
   <def>fracture of a bone</def>
  </sense>
 </homograph>
 <entry>
  <form>
   <orth order="1">fraction of the mind</orth>
  </form>
  <homograph>
   <sense>
```

```
     <def>mental or emotional disturbance</def>
    </sense>
   </homograph>
</entry>
</entry>
```

Esimerkki 21. Sanan *fraction* määritelmä dictionary.dtd:n mukaisena XML:nä. Vertaa esimerkkiin 19.

Kummankin **orth**-elementin sisältö on sama, mutta tämä juontuu alkuperäisestä tietokantatulosteesta: siinäkin elementeillä **termText** ja **variantText** on identtinen sisältö.

Jos DMV_to_dictionary.xsl-muunnokselle annetaan parametrin **narrowerTermEntry** arvoksi *yes*, tulostuu esimerkin 21 lisäksi seuraava artikkeli:

```
<entry>
 <form>
  <orth order="1">fraction of the mind</orth>
 </form>
 <homograph>
  <related type="see">
   <xref>fraction</xref>
  </related>
 </homograph>
</entry>
```

Esimerkki 22. Alitermin *fraction of the mind* artikkeli.

Tietokantatuloste sisältää varsinkin **etymology**-elementeissä tekstimuotoisia viittauksia muihin hakusanoihin. En ole muuntanut näitä **related**-elementeiksi, koska niillä ei ole tulosteessa mitään erityistä merkintää, vaan ne ovat tekstisisältöä: `<etymology>cf. $grind$.</etymology>`. Ne voisi tunnistaa ja käsitellä seuraavalla logiikalla:

> Jos **etymology**-elementti sisältää merkkijonon "cf. ", sitä seuraava $-merkkien välissä oleva merkkijono muunnetaan **related**-elementiksi.

”Cf.”-merkintää käytetään kuitenkin viittaamaan myös muissa sanakirjoissa oleviin hakusanoihin. Lisäksi kuvattu logiikka ei käsittele oikein esimerkiksi seuraavaa elementtiä, jossa viittaus osoittaa kahteen erilliseen hakusanaan, *axes* ja *accessum*:

```
<etymology>OF/AN $acces$ ($FEW$ 24, 73a, $AND$ s.v. $accés$)
 and L $accessus$ ($OLD$ s.v.). Cf. $axes, accessum$.
</etymology>.
```

Tuloksena olisi **related**-elementti, joka osoittaisi hakusanaan "axes, accessum", eikä sellaista ole.

6.2 Muunnos dictionary-muodosta HTML-muotoon

Yksi XML-muodon vahvuuksista on sen helppo muunnettavuus muihin formaatteihin. Tämä edellyttää, että muunnettava XML-tiedosto sisältää kaiken kohdeformaatin muodostamiseen tarvittavan tiedon. Lisäksi tiedon pitää olla järjestettynä elementteihin

ja attribuutteihin rakenteeksi, josta jokainen tietoalkio voidaan tunnistaa. Seuraavaksi osoitan, että dictionary.dtd:n mukainen XML täyttää nämä vaatimukset riittävän hyvin, jotta se voidaan muuntaa edelleen HTML-muotoon.

Muunnan dictionary-dokumenttityypin mukaisen tiedoston HTML-muotoon XSLT-muunnoksella dictionary_to_HTML.xsl, joka on liitteessä 4. XSLT-muunnoksen käynnistävä komento sekä selostus sen parametreista on esitetty liitteessä 5.

Dictionary_to_HTML.xsl -muunnoksen syötetiedoston elementit vastaavat lopputuloksen elementtejä taulukon 1 esittämällä tavalla:

Syötetiedoston elementit, määritelty dictionary.dtd:ssä	**Tulosteen elementit, HTML**
dictionary	html
entry, ylin taso	div class=”entry”
entry, muut tasot	div class=”entry_*”, missä * on entry-elementin taso
hyph	div class="hyph"
pron	div class="pron"
form/orth order="1"	h*, missä * on entry-elementin taso + 1
form/orth	div class="form"
homograph	• jos homograph-elementtejä on useita: ol/li • muuten ei mitään
sense	• jos sense-elementtejä on useita: ol/li • muuten div class="sense"
def	pelkkä sisältö
etymology	div class="etym"
additional	div class="additional"
description	pelkkä sisältö
related	div class="xref" / a
b	b

Taulukko 2. Dictionary- ja HTML-dokumenttityyppien elementtien vastaavuudet.

Dictionary_to_HTML.xsl-muunnos on mutkikkaampi kuin kohdassa 6.1 esitelty DMV_to_dictionary.xsl-muunnos, eikä sen tarkka läpikäynti ole tämän työn kannalta yhtä olennaista. Siksi esittelen siitä vain kahden XSL-säännön toimintaa.

6.2.1 Form-elementin käsittely

Form-elementti käsitellään säännössä `<xsl:template match="form">`, joka kutsuu sääntöjä `<xsl:template match="orth[@order='1']">` ja `<xsl:template match="orth[not(@order='1')]"/>`. Näiden toiminta on seuraavassa esitetty riveittäin:

`<xsl:template match="form">`	Käsittele form-elementit näin:
`<xsl:apply-templates/>`	Käsittele form-elementin orth-lapsielementit jompaakumpaa sääntöä käyttäen:
	Ensimmäinen orth-lapsielementti:
`<xsl:template match="orth[@order='1']">`	Käsittele ensimmäinen orth-elementti näin:
`<xsl:variable name="depth" select="count( ancestor::entry) + 1"/>`	Luo muuttuja nimeltä $depth, jonka arvo on tämän elementin entry-esivanhempien lukumäärä + 1
`<xsl:variable name= "h_element" select= "concat('h', $depth)"/>`	Luo muuttuja nimeltä $h_element, jonka arvo on merkin ’h’ ja $depth-muuttujan arvon yhdistelmä.
`<xsl:element name="{$h_element}">`	Tulosta alkutunniste elementille, jonka nimi on muuttujan $h_element arvo.
`<xsl:attribute name="id">`	Tulosta attribuutti nimeltä id.
`<xsl:value-of select="."/>`	Tulosta id-attribuutin arvoksi tämän orth-elementin sisältö. Tämä attribuutti toimii ristiviittausten kohteena.
`</xsl:attribute>`	Sulje id-attribuutti.
`<xsl:apply-templates/>`	Tulosta orth-elementin sisältö.
`</xsl:element>`	Tulosta elementin lopputunniste.
`</xsl:template>`	
	Muut orth-lapsielementit:
`<xsl:template match= "orth[not(@order='1')]"/>`	Älä tee mitään. Nämä elementit käsitellään <xsl:template match="form"> -säännössä seuraavaksi.
`<xsl:if test= "orth[not(@order='1')]">`	Jos tällä form-elementillä on enemmän kuin yksi orth-lapsielementti:
`<div class="form">`	• tulosta elementti
`<xsl:for-each select= "orth[not(@order='1')]">`	• tee ensimmäistä lukuunottamatta jokaiselle orth-lapsielementille tämä:
`<xsl:value-of select="."/>`	• tulosta tämän orth-elementin sisältö
`<xsl:if test="position() != last()">`	• jos orth-elementtejä on vielä lisää,
`<xsl:text>, </xsl:text>`	• tulosta pilkku ja välilyönti
`</xsl:if>`	
`</xsl:for-each>`	
`</div>`	Tulosta div-elementin lopputunniste.
`</xsl:if>`	
`</xsl:template>`	

Listaus 5. **Form**-elementin käsittely.

Tuloksena on HTML-rakenne, jossa sanan ensimmäinen kirjoitusasu on h2-, h3-, h4- tai h5-elementti riippuen siitä, onko se päätermi vai jonkin alemman tason alitermi. Mahdolliset muut kirjoitusasut ovat div-elementin sisällä pilkuin eroteltuina.

6.2.2 Sense-elementin käsittely

Sense-elementti käsitellään säännössä `<xsl:template name="outputSenses">`, jota kutsutaan **homograph**-elementin käsittelun aikana. Se kutsuu kahta muuta sääntöä, `<xsl:template name="senseToListItem">` ja `<xsl:template match="sense">`. Näiden toiminta on seuraavassa esitetty riveittäin:

`<xsl:template name="outputSenses">`	
`<xsl:choose>`	
`<xsl:when test= "count(sense) &gt; 1">`	Jos sense-elementtejä on enemmän kuin yksi:
`<ol type="a">`	• tulosta ol-elementin alkutunniste ja anna sen type-attribuutin arvoksi ”a”
`<xsl:for-each select="sense">`	• tee jokaiselle sense-elementille tämä:
`<xsl:call-template name="senseToListItem"/>`	• kutsu sääntöä <xsl:template name="senseToListItem">
`<xsl:template name="senseToListItem">`	
`<li>`	Tulosta li-elementin alkutunniste.
`<xsl:apply-templates/>`	Tulosta tämän sense-elementin sisältö.
`</li>`	Tulosta li-elementin lopputunniste.
`</xsl:template>`	
`</xsl:for-each>`	
`</ol>`	• tulosta ol-elementin lopputunniste
`</xsl:when>`	
`<xsl:otherwise>`	Muussa tapauksessa:
`<xsl:apply-templates/>`	• käsittele tämän homograph-elementin lapsielementit omilla säännöillään:
`<xsl:template match="sense">`	Käsittele sense-elementit näin:
`<div class="sense">`	Tulosta div-elementin alkutunniste ja anna sen class-attribuutin arvoksi “sense”.
`<xsl:apply-templates/>`	Tulosta tämän sense-elementin sisältö.
`</div>`	Tulosta div-elementin lopputunniste.
`</xsl:template>`	
`</xsl:otherwise>`	
`</xsl:choose>`	
`</xsl:template>`	

Listaus 6. **Sense**-elementin käsittely.

Tuloksena on HTML-rakenne, jossa sanan merkitys on joko div-elementissä tai, jos merkityksiä on useita, ne ovat listassa merkittyinä kirjaimin a, b, c ja niin edelleen.

6.2.3 Esimerkki HTML-muotoisesta sanakirja-artikkelista

Esimerkissä 21 esitettiin dictonary.dtd:n mukainen yksittäisen sanakirja-artikkelin rakenne. Dictionary_to_HTML.xsl-muunnos tuottaa siitä seuraavan tuloksen:

```
<div class="entry">
 <h2 id="fraction">fraction</h2>
 <div class="form">fraction</div>
 <div class="sense">fracture of a bone</div>
 <div class="etym">OF <b>fraction</b> (<b>FEW</b> 3, 743b,
<b>DMF</b> s.v.) and ML <b>fraction-em</b> (<b>DML</b> s.v.).
 </div>
 <div class="entry_2">
  <h3 id="fraction of the mind">fraction of the mind</h3>
  <div class="sense">mental or emotional disturbance</div>
 </div>
</div>
```

Esimerkki 23. Sanan *fraction* sanakirja-artikkelin HTML-esitys. Vertaa esimerkkiin 21.

Jos HTML-muunnoksen lähdetiedostossa on hakusanat myös alitermeille, dictionary_to_HTML.xsl-muunnos tulostaa lisäksi seuraavan artikkelin:

```
<div class="entry">
 <h2 id="fraction of the mind">fraction of the mind</h2>
 <div class="xref">See <a href="#fraction">fraction</a></div>
</div>
```

Esimerkki 24. Sanan *fraction of the mind* sanakirja-artikkelin HTML-esitys.

7 Tulosten vertailu muihin vastaaviin toteutuksiin

Tässä luvussa vertaan oman toteutukseni eri osia kolmeen aiemmin tehtyyn XML-muotoiseen toteutukseen: alkuperäisen tietomallin vaihtoehtoiseen XML-esitykseen, TEI-standardin piirrerakenne-esitykseen sekä TEI-standardin sanakirjatoteutukseen.

7.1 Piirrerakennemallin muut toteutukset

Taustakartoituksessani en ole löytänyt Iden ja kumppanien [1993] piirrerakennemallin XML-toteutusta sellaisenaan. Hyvin lähellä sellaista on kuitenkin Iden ja kumppanien [2000] XML-toteutus, joka perustuu Erjavecin ja kumppanien [2000] sanakirjarakenteeseen. Iden ja kumppanien [2000] malli on eräänlainen piirrerakennemallin laajennus, jossa perusrakenteena on puumainen hierarkia. Alkuperäinen piirrerakennemallihan ei eksplisiittisesti määrittele tällaista hierarkiaa, vaikka piirrerakenne käytännössä muodostaakin sellaisen. Iden ja kumppanien [2000] mallissa puuhierarkian solmukohdissa oleva tieto on määritelty piirrerakenteina eli attribuutti-arvo -pareina. Itse sisällön kuvaus muistuttaa siis suuresti Iden ja kumppanien [1993] mallia. Ide ja kumppanit [2000] esittävät mallilleen XML-toteutuksen, jossa hierarkia on toteutettu kolmella elementillä: **struc**, **brack** ja **alt**. Ainoastaan näillä elementeillä voi olla jälkeläisiä. Varsinainen sisältö on TEI-standardista lainatuissa elementeissä, kuten **orth**, **pron**, ja **hyph**. Monet näistä TEI-elementeistä on nimetty alun perin Iden ja kumppanien [1993] artikkelista lainatuilla nimillä.

```
<?xml version="1.0" encoding="utf-8"?>
<dict>
<struc>
     <orth>overdress</orth>
     <struc><orth>overdress</orth>
            <pos>verb</pos>
            <pron>zzzz</pron>
            <def> To dress (oneself or another) too elaborately or finely </def>
     </struc>
     <struc><orth>overdress</orth>
            <pos>noun</pos>
            <pron>yyyy</pron>
            <def> A dress that may be worn over a jumper, blouse, etc.</def>
     </struc>
</struc>
</dict>
```

Kuva 7. Iden ja kumppanien [2000] mallin XML-esitys.

Iden ja kumppanien [2000] XML-esitys siis tavallaan toteuttaa alkuperäisen piirrerakennemallin. Pidän kuitenkin ongelmallisena sen ratkaisua käyttää hierarkiarakenteen ilmaisemiseen erillisiä elementtejä. Tästä ratkaisusta seuraa, että esimerkiksi **struc**-elementti voi sisältää kaikki muut elementit, joten sen informaatioarvo on itse asiassa hyvin vähäinen. Sama ilmaisuvoima voidaan saavuttaa myös ilman sitä, kuten esimerkki 25 osoittaa. Kuvan 7 sisältö on siinä dictionary.dtd:n mukaisena rakenteena, siis ilman **struc**-tasoa.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE dictionary SYSTEM "dictionary.dtd">
<dictionary>
 <entry>
  <form>
   <orth>overdress</orth>
  </form>
  <homograph>
   <gram>
    <pos>v</pos>
   </gram>
   <sense>
    <form>
     <pron>zzzz</pron>
    </form>
    <def> To dress (oneself or another) too elaborately or finely
    </def>
   </sense>
  </homograph>
  <homograph>
   <gram>
    <pos>n</pos>
   </gram>
   <sense>
    <form>
     <pron>yyyy</pron>
    </form>
    <def> A dress that may be worn over a jumper, blouse, etc.
    </def>
   </sense>
  </homograph>
 </entry>
</dictionary>
```

Esimerkki 25. Kuvan 7 esittämä rakenne dictionary.dtd:n mukaisena XML:nä.

7.2 TEI-standardin piirrerakenne

TEI sisältää piirrerakenteen geneerisen XML-esityksen [Text Encoding Initiative: Feature Structures 2020]. TEI:n sääntöjen määrittelemä XML-sanasto, siis elementtien ja attribuuttien nimet, ei kuitenkaan kuvaa minkään tietyn piirrerakenteen sisältöä. Toisin sanoen se ei ole semanttinen esitys: jokainen TEI:n säännöillä tuotettu XML-rakenne sisältää samannimiset elementit ja attribuutit. Tämä johtuu siitä, että TEI määrittelee oman dokumenttityyppinsä, jonka ennaltamäärättyjä elementtejä ja attribuutteja on pakko käyttää TEI:n mukaisissa dokumenteissa.

XML kuitenkin mahdollistaa oman dokumenttityypin määrittelemisen semanttisesti jokaiselle sisältötyypille erikseen, jolloin elementit ja attribuutit voidaan nimetä kuten halutaan. Tällöin niille on tarkoituksenmukaista antaa mahdollisimman kuvaavat nimet. Esimerkki 26 havainnollistaa tämän lähestymistavan ja TEI:n piirrerakenne-esityksen eroa.

Piirrerakenne:

```
[case: accusative]
[gender: feminine]
[number: plural]
```

TEI:n piirrerakenne-esitys [Text Encoding Initiative: Feature Structures 2020]**:**

```
<fs>
 <f name="case">
  <symbol value="accusative"/>
 </f>
 <f name="gender">
  <symbol value="feminine"/>
 </f>
 <f name="number">
  <symbol value="plural"/>
 </f>
</fs>
```

Tätä sisältötyyppiä varten luotu XML-esitys:

```
<classification>
 <case>accusative</case>
 <gender>feminine</gender>
 <number>plural</number>
</classification>
```

Esimerkki 26. TEI:n piirrerakenne-XML ja vastaava semanttinen XML-esitys

Esimerkissä 26 ylätason elementti, **classification**, on nimetty kuvaamaan tämän ylätason sisältöä, mutta muiden elementtien nimet on saatu suoraan piirrerakenteesta. Tulos on lyhyempi ja selkeämpi kuin TEI:n esitys, ja sellaisena helpommin ihmisen tuotettavissa ja luettavissa. Sen mahdollinen haittapuoli on, että jos tätä XML-rakennetta halutaan prosessoida automaattisesti, sitä varten pitää muokata työkalut tukemaan juuri näitä elementtinimiä. Sama koskee kuitenkin jossain määrin myös TEI:n piirrerakenne-esitystä: esimerkiksi **symbol**-elementin käsittely riippuu sen **value**-attribuutin arvosta. Jos kehitetään työkalut kieliopillisen TEI-piirrerakenteen automaattiseen käsittelyyn, niitä ei voida käyttää esimerkiksi lääketieteellisen TEI-piirrerakenteen käsittelyyn, vaikka itse XML-dokumenttityyppi on kummassakin täsmälleen sama.

7.3 TEI-standardin sanakirjarakenne

Luvuissa 5 ja 6 kuvattu prosessi tuotti DTD:n, jonka semanttinen rakenne on hyvin samankaltainen kuin TEI-standardin sanakirjaosan dokumenttimäärittelyssä. Tämä ei ole yllättävää, koska kumpikin pohjautuu jossain määrin samaan piirrerakennemalliin, kuten totean kohdassa 2.2.2. Samankaltaisuus kuitenkin vahvistaa osaltaan FSXML-sääntöjen käyttökelpoisuuden. Esimerkki 27 havainnollistaa dokumenttityyppien samankaltaisuutta esittämällä yhden DMV–sanakirjan artikkelin sekä dictionary.dtd:n että TEI:n dokumenttityypin tei_all.dtd:n mukaisena XML:nä.

dictionary.dtd:n mukainen XML

```
<entry>
 <form>
  <orth order="1">
   abater</orth>
  <orth>abatere</orth>
 </form>
 <etymology>f. v. <b>
  abate</b> 'relieve,
  mitigate' +
<b>-er</b>.</etymology>
 <homograph>
  <gram>
   <pos>n</pos>
  </gram>
  <sense>
   <def>medicine alleviating
    pain</def>
  </sense>
 </homograph>
</entry>
```

tei_all.dtd:n mukainen XML

```
<entry>
 <form>
  <orth>abater</orth>
  <orth>abatere</orth>
 </form>
 <etym>f. v.
  <emph>abate</emph>
  'relieve, mitigate' +
  <emph>-er</emph>.
 </etym>
 <hom>
  <gramGrp>
   <pos>n</pos>
  </gramGrp>
  <sense>
   <def>medicine alleviating
    pain</def>
  </sense>
 </hom>
</entry>
```

Esimerkki 27. Piirrerakenteen esitys dictionary.dtd:n ja TEI:n mukaisessa XML-muodossa. Sisältö perustuu lähteeseen [Norri 2016].

Jotta dictionary.dtd:n **entry**-elementistä saadaan tei_all.dtd:n mukainen, tarvitsee siis vain poistaa **orth**-elementin attribuutti **order** sekä vaihtaa elementtien **etymology**, **b**, **homograph** ja **gram** nimet muotoon **etym**, **emph**, **hom** ja **gramGrp**. Suuri osa dictionary.dtd:n eroista TEI:hin verrattuna ovat vastaavanlaisia: itse rakenteet ovat valideja myös TEI:ssä, mutta elementtien nimet ja attribuutit poikkeavat TEI:stä.

Dictionary.dtd:ssä on muutama elementti, joille TEI:ssä ei ole vastaavuutta: **additional** sisältää lisätietoa sanan alkuperästä, **description** sanan merkityksestä. Nämä elementit ovat peräisin DMV-sanakirjan tietokantatulosteessa, ja niiden alkuperä on sanakirjan kokoamiseen käytetyn tietokannan käsitteellisessä rakenteessa [Norri *et al*. 2020]. TEI:ssä on muita, monimutkaisempia rakenteita vastaavan tiedon esittämiseen **etym**- ja **sense**-elementtien sisällä. Sen sijaan dictionary.dtd:n **domain**-elementin sisältämälle sanan aihealueelle en ole löytänyt TEI:stä sopivaa elementtiä tai rakennetta, vaikka pidän todennäköisenä, että TEI sisältää sellaisen. TEI:ssä on kylläkin **domain**-niminen elementti, mutta se määrittelee koko dokumentin aihealueen. Neljäs elementti, jolle en ole tunnistanut vastaavuutta TEI:n dokumenttityypistä on **sem**. Se esiintyy alkuperäisen piirrerakennemallin esimerkeissä, mutta Ide ja kumppanit eivät selitä sen tarkoitusta.

Dictionary.dtd:n pohjana oleva piirrerakenne perustuu vain englannin- ja ranskankielisiin esimerkkeihin. TEI:n sanakirjamallilla taas on tarkoitus tukea mahdollisimman monia kieliä ja myös monikielisiä sanakirjoja. Siksi se sisältää huomattavasti enemmän mahdollisia elementtejä kuin dictionary.dtd. Esimerkiksi TEI:n **form**-elementti voi sisältää dictionary.dtd:n tavoin elementit **orth**, **hyph** ja **pron**, mutta

niiden lisäksi 14 muutakin elementtiä, joista suurin osa liittyy kieliopilliseen luokitteluun. Monet niistä ovat piirrerakennemalliin verrattuna hyödyllisiä lisäyksiä varsinkin muiden kielten kuin englannin kuvailussa: esimerkiksi **case** määrittelee sanan sijamuodon ja **mood** verbin moduksen.

Koska TEI pyrkii olemaan mahdollisimman monikäyttöinen standardi, jota hyvin erilaiset sanakirjaprojektit voivat käyttää, se määrittelee useita vaihtoehtoisia tapoja saman informaation esittämiseen. Dictionary.dtd:ssä tällaiseen ei ole tarvetta, koska se on tehty tukemaan vain yhtä sanakirjaa. Esimerkiksi TEI:n **gramGrp**-elementti voi sisältää 10 eri elementtiä, mutta ne voidaan kaikki ilmaista myös **gram**-elementin type-attribuutin avulla. Esimerkiksi nämä kaksi rakennetta ovat merkitykseltään identtiset ja tei_all.dtd:n mukaan validit:

```
<gramGrp>
 <pos>v</pos>
</gramGrp>
```

```
<gramGrp>
 <gram type="pos">v</gram>
</gramGrp>
```

Esimerkki 28. Verbi-sanaluokka kahdella eri tavalla ilmaistuna.

Vaikka sanakirja-artikkelit ovat yleensä hyvin määrämuotoista, monien sanojen kuvaus sisältää myös poikkeusrakenteita. Dictionary.dtd:ssä ei ole tarvinnut varautua tähän, koska DMV-sanakirjan artikkelit ovat rakenteeltaan hyvin homogeenisiä. Poikkeusrakenteet ovat kuitenkin yleinen pulma sanakirjatietomallien määrittelyssä. TEI ratkaisee sen esittelemällä kaksi elementtiä, **entryfree** ja **dictScrap**, joiden sisällä useimmat elementit on sallittu. Tämä mahdollistaa lähes minkä tahansa rakenteen sanakirja-artikkelille, mutta saattaa aiheuttaa ongelmia sisällön automaattisessa käsittelyssä, kuten muunnoksissa eri formaatteihin. Esimerkiksi kattavan XSL-muunnoksen kirjoittaminen voi olla hankalaa, jos sisällön rakenne on vaikeasti ennakoitavissa.

8 Johtopäätökset

Olen tässä pro gradu -työssä osoittanut, että Iden ja kumppanien [1993] piirrerakenteisiin perustuva tietomalli voidaan toteuttaa XML-dokumenttityyppinä. Esittämilläni FSXML-säännöillä mikä tahansa piirrerakenne voidaan muuntaa XML-muotoon. Sovelsin FSXML-sääntöjä piirrerakennemalliin, minkä tuloksena määrittelin dictionary-dokumenttityypin. Osoitin myös, että dictionary-dokumenttityypillä voidaan esittää todellista sanakirjasisältöä XML-muodossa, ja että tämä XML-muotoinen esitys voidaan edelleen muuntaa HTML-esitysformaattiin. Lisäksi kehitin XSL-tyylisivut, joilla nämä muunnokset XML:ksi ja edelleen HTML:ksi voidaan tehdä.

Dictionary-dokumenttityyppi toteuttaa piirrerakennemallin mahdollisimman tarkasti. Lisäksi DMV (*Dictionary of Medical Vocabulary in English, 1375-1550*) -sanakirja sisältää rakenteita, joita Ide ja kumppanit eivät käyttäneet. Tämä oli odotettavissa, sillä kuten jo johdannossa totesin, Iden ja kumppanien artikkeli ei pyri esittämään täydellistä sanakirjan tietomallia, vaan pikemminkin periaatteet, joilla sellainen voidaan luoda. Näiden piirrerakennemallista puuttuvien rakenteiden kuvaamiseksi olen lisännyt dictionary.dtd:hen muutaman elementin, joiden nimet ja sisältötyypit ovat peräisin suoraan DMV:n käyttämästä käsitteellisestä rakenteesta.

Mahdollinen jatkotutkimuksen aihe olisi jonkin muun sanakirjan sisällön siirtäminen dictionary.dtd:n mukaiseen muotoon. Oletan, että tämä paljastaisi dictionary-dokumenttityypin puutteita, kuten puuttuvia elementtejä tai liian tiukkoja DTD-määrittelyjä, jotka eivät salli elementtiä paikassa, jossa sitä tarvittaisiin. Voi myös paljastua, että jokin toinen sanakirja sisältää artikkeleita, joiden rakenne poikkeaa paljonkin tyypillisestä. Tällöin tulee harkittavaksi, väljennetäänkö DTD:n määrittelyjä huomattavasti, mikä lisää semanttisesti virheellisten rakenteiden riskiä, vai lisätäänkö DTD:hen TEI-standardin tapaan vaihtoehtoinen **entry**-tason elementti, jonka sisälle sallitaan väljempi rakenne.

Toinen lisätutkimuksia ansaitseva aihe on luvussa 5 esitellyt FSXML-säännöt, jolla mielivaltainen piirrerakenne voidaan muuntaa XML:ksi. Tätäkin aihetta olisi syytä tutkia muulla sisällöllä, käytännössä siis soveltaa FSXML-sääntöjä laajempiin piirrerakenteisiin. Piirrerakenteisiin voidaan myös liittää sisällön tyypitystä, mitä tässä työssä ei ole tarvinnut ottaa huomioon. Oletukseni on, että tyypityksellä ei ole vaikutusta itse FSXML-sääntöihin, mutta se saattaa vaatia DTD:n sijaan XML Scheman käyttöä muunnoksen tuloksena syntyvän dokumenttityypin määrittämiseen.

Viiteluettelo

Ala-Fossi, Jukka. 2021. Pro gradu –työ, käsikirjoitus.

Amsler, Robert A., & Frank W. Tompa. 1988. An SGML-based standard for English monolingual dictionaries. *Proceedings of the 4th Annual Conference of the UW Centre for the New Oxford English Dictionary*. Waterloo, Ontario, 61-80.

Debus-Gregor, Esther & Ulrich Heid. 2013. Design criteria and 'added value' of electronic dictionaries for human users. In: Gouws, Rufus H., Ulrich Heid, Wolfgang Schweickard, Herbert Ernst Wiegand (eds.), *Dictionaries: An International Encyclopedia of Lexicography. Supplementary Volume: Recent Developments with Focus on Electronic and Computational Lexicography.* De Gruyter Mouton.

Erjavec, Tomaž, Roger Evans, Nancy Ide & Adam Kilgarriff. 2000. The concede model for lexical databases. *LREC 2000 Conference proceedings*.

Francopoulo, Gil, Nuria Bel, Monte George, Nicoletta Calzolari, Monica Monachini, Mandy Pet, Claudia Soria. 2007. Lexical Markup Framework: ISO Standard for Semantic Information in NLP Lexicons. GLDV (Gesellschaft für linguistische Datenverarbeitung), Tubingen.

Ide, Nancy, Jacques Le Maitre & Jean Véronis. 1993. Outline of a model for lexical databases. *Information Processing & Management*. Volume 29, 1993, 159-186.

Ide, Nancy, Laurent Romary & Adam Kilgarriff. 2000. A Formal Model of Dictionary Structure and Content. *Proceedings of the Ninth EURALEX International Congress, EURALEX 2000*. Stuttgart, Germany, 113-126.

ISO 8879:1986. Standard Generalised Markup Language (SGML). Preview. https://www.iso.org/obp/ui/#iso:std:iso:8879:ed-1:v1:en (haettu 14.1.2021)

ISO 1951:2007. Presentation/representation of entries in dictionaries — Requirements, recommendations and information. https://www.iso.org/standard/36609.html (haettu 18.2.2021)

Lemnitzer, Lothar, Laurent Romary & Andreas Witt. 2013. Representing human and machine dictionaries in markup languages (SGML, XML). In: Gouws, Rufus H., Ulrich Heid, Wolfgang Schweickard, Herbert Ernst Wiegand (eds.), *Dictionaries: An International Encyclopedia of Lexicography. Supplementary Volume: Recent Developments with Focus on Electronic and Computational Lexicography.* De Gruyter Mouton.

Neff, Mary S., Roy J. Byrd & Omneya A. Rizk.1988. Creating and querying lexical data bases. *Proceedings of the second conference on applied natural language processing.* Stroudsburg, Pennsylvania, 84–92.

Norri, Juhani. 2016. *Dictionary of Medical Vocabulary in English, 1375-1550*. Ashgate Publishing.

Norri, Juhani, Marko Junkkari, & Timo Poranen. 2020. Digitization of Data for a Historical Medical Dictionary. *Language resources and evaluation*. Volume 54, 2020, 615–643.

Romary, Laurent. 2013. Standardization of the Formal Representation of Lexical Information for NLP. In: Gouws, Rufus H., Ulrich Heid, Wolfgang Schweickard, Herbert Ernst Wiegand (eds.), *Dictionaries: An International Encyclopedia of Lexicography. Supplementary Volume: Recent Developments with Focus on Electronic and Computational Lexicography.* De Gruyter Mouton.

Roth, Mark, Henry Korth, & Abraham Silberschatz. 1988. Extended Algebra and Calculus for Nested Relational Databases. *ACM transactions on database systems*. Volume 13, 1988, 389–417.

Storrer, Angelika. 2013. Representing computational dictionaries in hypertextual form. In: Gouws, Rufus H., Ulrich Heid, Wolfgang Schweickard, Herbert Ernst Wiegand (eds.), *Dictionaries: An International Encyclopedia of Lexicography. Supplementary Volume: Recent Developments with Focus on Electronic and Computational Lexicography.* De Gruyter Mouton.

Text Encoding Initiative. Projects Using the TEI. https://tei-c.org/activities/Projects/ (haettu 26.9.2020)

Text Encoding Initiative. Dictionaries. Version 4.1.0, 2020. https://www.tei-c.org/release/doc/tei-p5-doc/en/html/DI.html (haettu 20.9.2020)

Text Encoding Initiative. Feature Structures. Version 4.1.0, 2020. https://www.tei-c.org/release/doc/tei-p5-doc/en/html/FS.html (haettu 15.11.2020)

Trippel, Thorsten. 2013. Representing computational dictionaries in feature-structure-based representation formalisms and typed feature logic. In: Gouws, Rufus H., Ulrich Heid, Wolfgang Schweickard, Herbert Ernst Wiegand (eds.), *Dictionaries: An International Encyclopedia of Lexicography. Supplementary Volume: Recent Developments with Focus on Electronic and Computational Lexicography.* De Gruyter Mouton.

Tutin, Agnès & Jean Véronis. 1998. Electronic Dictionary Encoding: Customizing the TEI Guidelines. *Proceedings of Euralex 1998*. Liège, Belgium.

Vaquero, Antonio, Francisco Alvarez & Fernando Sáenz. 2013. Representing computational dictionaries in relational databases. In: Gouws, Rufus H., Ulrich Heid, Wolfgang Schweickard, Herbert Ernst Wiegand (eds.), *Dictionaries: An International Encyclopedia of Lexicography. Supplementary Volume: Recent Developments with Focus on Electronic and Computational Lexicography.* De Gruyter Mouton.

Véronis, Jean & Nancy Ide. 1992. A feature-based model for lexical databases. *Proceedings of the 14th international conference on computational linguistics, COLING 1992*. Stroudsburg, Pennsylvania, 588–594.

World Wide Web Consortium. Extensible Markup Language (XML) 1.0, 1998. https://www.w3.org/TR/1998/REC-xml-19980210 (haettu 1.6.2020)

World Wide Web Consortium. Extensible Stylesheet Language (XSL) Version 1.0, 2001. https://www.w3.org/TR/2001/REC-xsl-20011015/ (haettu 1.6.2020)

World Wide Web Consortium. XML Schema Definition Language (XSD) 1.1 Part 1: Structures, 2012. https://www.w3.org/TR/xmlschema11-1/ (haettu 24.3.2021)

World Wide Web Consortium. The Extensible Stylesheet Language Family (XSL), 2017. https://www.w3.org/Style/XSL/ (haettu 1.6.2020)

Liite 1: dictionary.dtd

```
<!-- Perustuu artikkeliin Ide, Nancy, Jacques Le Maitre & Jean
Véronis. 1993. Outline of a model for lexical databases. Information
Processing & Management. Volume 29, 1993, 159-186. -->

<!-- Juurielementti, oma lisäys -->
<!ELEMENT dictionary (entry)+>

<!-- Yksittäinen sana -->
<!ELEMENT entry (form+, etymology?, additional?, homograph+, entry*)>

<!-- Sanan ilmiasut -->
<!ELEMENT form (orth*, hyph*, pron*)>

<!-- Kirjoitusasu -->
<!ELEMENT orth (#PCDATA)>

<!-- Kirjoitusasun varianttien luokittelu:
           geo: kielialue
           order: varianttien järjestys, 1 = ensisijainen kirjoitusasu
-->
<!ATTLIST orth
           geo CDATA #IMPLIED
           order CDATA #IMPLIED>

<!-- Tavutus -->
<!ELEMENT hyph (#PCDATA)>

<!-- Äänneasu -->
<!ELEMENT pron (#PCDATA)>

<!-- Esivanhemman samannimisen elementin ohittaminen:
           yes = tämä arvo kumoaa esivanhemman samannimisen elementin
           arvon
           no = tämä arvo yhdistetään esivanhemman samannimisen
           elementin arvoon -->
<!ATTLIST pron
           override (yes|no) #IMPLIED>

<!-- Alkuperä, oma lisäys -->
<!ELEMENT etymology (#PCDATA|b)*>

<!-- Korostus, oma lisäys -->
<!ELEMENT b (#PCDATA)>

<!-- Lisätietoa sanan alkuperästä, oma lisäys  -->
<!ELEMENT additional (#PCDATA|b)*>

<!-- Homograafit -->
<!ELEMENT homograph (gram?, sense*, related*)>

<!-- Kielioppiluokitus -->
<!ELEMENT gram (pos, gend?, numb?, (subc|gramc)*)>

<!-- Sanaluokka -->
<!ELEMENT pos (#PCDATA)>

<!-- Suku -->
<!ELEMENT gend (#PCDATA)>
```

```
<!-- Luku -->
<!ELEMENT numb (#PCDATA)>

<!-- Sanaluokan alakategoria -->
<!ELEMENT subc (#PCDATA)>

<!-- Kieliopillinen kategoria -->
<!ELEMENT gramc (#PCDATA)>

<!-- Merkitys -->
<!ELEMENT sense (form?, gram*, usage?, sem?, def+, description?, ex*,
xref*, sense*)>

<!-- Käyttötavat -->
<!ELEMENT usage (#PCDATA|reg|domain)*>

<!-- Tarkemmin määrittelemätöntä metadataa, ei esiinny painetussa
sanakirjassa -->
<!ELEMENT sem (#PCDATA)>

<!ATTLIST sem
           scode CDATA #IMPLIED
           boxc CDATA #IMPLIED>

<!-- Määritelmä -->
<!ELEMENT def (#PCDATA|b)*>

<!-- Lisätietoa määritelmään, oma lisäys -->
<!ELEMENT description (#PCDATA|b)*>

<!-- Esimerkki -->
<!ELEMENT ex (#PCDATA)>

<!-- Viittaus esim. synonyymiin -->
<!ELEMENT xref (#PCDATA)>

<!-- Sanan rekisteri -->
<!ELEMENT reg (#PCDATA)>

<!-- Sanan käyttöalue -->
<!ELEMENT domain (#PCDATA)>

<!-- Viittaus läheisesti liittyvään sanaan, oma lisäys -->
<!ELEMENT related (xref+)>

<!-- Viittaustyypit:
           see = katso
           cf = vertaa
           related = liittyvä sana
           synonym = synonyymi
           antonym = vastakohta
           -->
<!ATTLIST related
           type (see|cf|related|synonym|antonym) #IMPLIED>
```

Liite 2: thesaurus.dtd

```
<!ELEMENT thesaurus (term)+>

<!ELEMENT term (termText, etymology, additional, senses, variants,
narrowerTerms)>
<!ATTLIST term identifier CDATA #REQUIRED>

<!ELEMENT termText (#PCDATA)>
<!ELEMENT etymology (#PCDATA)>
<!ELEMENT additional (#PCDATA)>

<!ELEMENT senses (sense*)>

<!ELEMENT sense (senseText, description)>
<!ATTLIST sense identifier CDATA #REQUIRED>

<!ELEMENT senseText (#PCDATA)>

<!ELEMENT description (#PCDATA)>

<!ELEMENT variants (variant*)>

<!ELEMENT variant (variantText)>
<!ATTLIST variant identifier CDATA #REQUIRED>

<!ELEMENT variantText (#PCDATA)>

<!ELEMENT narrowerTerms (term*)>
```

Liite 3: DMV_to_dictionary.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Simo Voutilainen, 2021 -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="2.0">
 <xsl:output doctype-system="..\..\Dictionary\dictionary.dtd"
  indent="yes"/>

 <xsl:param name="narrowerTermEntry" select="'no'"/>

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

 <xsl:template match="thesaurus">
  <dictionary>
   <xsl:variable name="content">
    <xsl:apply-templates/>
    <xsl:if test="$narrowerTermEntry='yes'">
     <xsl:apply-templates select="term//term"
      mode="narrowerTermEntry"/>
    </xsl:if>
   </xsl:variable>
   <xsl:for-each select="$content/*">
    <xsl:sort select="lower-case(form/orth[1])"/>
    <xsl:copy-of select="."/>
   </xsl:for-each>
  </dictionary>
 </xsl:template>

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

 <xsl:template match="term//term" mode="narrowerTermEntry">
  <entry>
   <xsl:apply-templates select="termText"
    mode="narrowerTermEntryTermText"/>
   <homograph>
    <related type="see">
     <xref>
      <xsl:value-of select="ancestor-or-self::term[last()]/termText"/>
     </xref>
    </related>
   </homograph>
  </entry>
 </xsl:template>

 <xsl:template match="termText">
  <form>
   <orth order="1">
    <xsl:apply-templates/>
   </orth>
   <xsl:for-each select="../variants/variant">
    <orth>
     <xsl:apply-templates/>
    </orth>
```

```
  </xsl:for-each>
 </form>
</xsl:template>

<xsl:template match="termText" mode="narrowerTermEntryTermText">
 <form>
  <orth order="1">
   <xsl:apply-templates/>
  </orth>
 </form>
</xsl:template>

<xsl:template match="etymology">
 <xsl:if test="string-length(.) &gt; 0">
  <etymology>
   <xsl:call-template name="tagify">
    <xsl:with-param name="input_string" select="."/>
   </xsl:call-template>
  </etymology>
 </xsl:if>
</xsl:template>

<xsl:template match="additional">
 <xsl:if test="string-length(.) &gt; 0">
  <additional>
   <xsl:call-template name="tagify">
    <xsl:with-param name="input_string" select="."/>
   </xsl:call-template>
  </additional>
 </xsl:if>
</xsl:template>

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

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

<xsl:template match="senseText">
 <def>
  <xsl:call-template name="tagify">
   <xsl:with-param name="input_string" select="."/>
  </xsl:call-template>
 </def>
</xsl:template>

<xsl:template match="description">
 <xsl:if test="string-length(.) &gt; 0">
  <description>
   <xsl:call-template name="tagify">
    <xsl:with-param name="input_string" select="."/>
   </xsl:call-template>
  </description>
 </xsl:if>
</xsl:template>
```

```
 <xsl:template match="variants"/>

 <xsl:template match="narrowerTerms">
  <xsl:variable name="content">
   <xsl:apply-templates/>
  </xsl:variable>
  <xsl:for-each select="$content/*">
   <xsl:sort select="lower-case(form/orth[1])"/>
   <xsl:copy-of select="."/>
  </xsl:for-each>
 </xsl:template>

 <xsl:template name="tagify">
  <xsl:param name="input_string"/>
  <xsl:param name="char" select="'$'"/>
  <xsl:param name="element" select="'b'"/>
  <xsl:choose>
   <xsl:when test="contains($input_string, $char)">
    <xsl:value-of select="substring-before($input_string, $char)"/>
    <xsl:variable name="temp_string" select="substring-
     after($input_string, $char)"/>
    <xsl:element name="{$element}">
     <xsl:value-of select="substring-before($temp_string, $char)"/>
    </xsl:element>
    <xsl:variable name="rest" select="substring-after($temp_string,
     $char)"/>
    <xsl:call-template name="tagify">
     <xsl:with-param name="input_string" select="$rest"/>
    </xsl:call-template>
   </xsl:when>
   <xsl:otherwise>
    <xsl:value-of select="$input_string"/>
   </xsl:otherwise>
  </xsl:choose>
 </xsl:template>

</xsl:stylesheet>
```

Liite 4: dictionary_to_HTML.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Simo Voutilainen, 2021 -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="2.0">
 <xsl:output method="html" encoding="UTF8" indent="yes" use-character-
  maps="charmap"/>

 <xsl:character-map name="charmap">
  <xsl:output-character character="&#156;" string="œ"/>
 </xsl:character-map>

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

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

 <xsl:template match="entry">
  <xsl:variable name="level" select="count(ancestor-or-self::entry)"/>
  <xsl:choose>
   <xsl:when test="$level=1">
    <xsl:variable name="thisEntryInitial">
     <xsl:value-of select="substring(upper-case(form/orth[
      @order='1']), 1, 1)"/>
    </xsl:variable>
    <xsl:variable name="previousEntryInitial">
     <xsl:value-of select="substring(upper-case(preceding-
      sibling::entry[1]/form/orth[@order='1']), 1, 1)"/>
    </xsl:variable>
    <xsl:if test="$thisEntryInitial != $previousEntryInitial">
     <h1>
      <xsl:choose>
       <xsl:when test="contains('0123456789', $thisEntryInitial)">
        <xsl:if test="not(contains('0123456789',
         $previousEntryInitial)) or $previousEntryInitial = ''">
         <xsl:text>Numbers</xsl:text>
        </xsl:if>
       </xsl:when>
       <xsl:otherwise>
        <xsl:value-of select="$thisEntryInitial"/>
       </xsl:otherwise>
      </xsl:choose>
     </h1>
    </xsl:if>
    <div class="entry">
     <xsl:apply-templates select="hyph"/>
     <xsl:apply-templates select="pron"/>
     <xsl:apply-templates select="form"/>
     <xsl:call-template name="outputHomographs"/>
     <xsl:apply-templates select="etymology"/>
     <xsl:apply-templates select="additional"/>
     <xsl:apply-templates select="entry"/>
    </div>
    <br/>
```

```
  </xsl:when>
  <xsl:otherwise>
   <xsl:variable name="class" select="concat('entry_', $level)"/>
   <div>
    <xsl:attribute name="class">
     <xsl:value-of select="$class"/>
    </xsl:attribute>
    <xsl:apply-templates select="form"/>
    <xsl:call-template name="outputHomographs"/>
    <xsl:apply-templates select="etymology"/>
    <xsl:apply-templates select="additional"/>
    <xsl:apply-templates select="entry"/>
   </div>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>

<xsl:template match="hyph">
 <div class="hyph">
  <xsl:apply-templates/>
 </div>
</xsl:template>

<xsl:template match="pron">
 <div class="pron">
  <xsl:apply-templates/>
 </div>
</xsl:template>

<xsl:template match="form">
 <xsl:apply-templates/>
 <xsl:if test="orth[not(@order='1')]">
  <div class="form">
   <xsl:for-each select="orth[not(@order='1')]">
    <xsl:value-of select="."/>
    <xsl:if test="position() != last()">
     <xsl:text>, </xsl:text>
    </xsl:if>
   </xsl:for-each>
  </div>
 </xsl:if>
</xsl:template>

<xsl:template match="orth[@order='1']">
 <xsl:variable name="depth" select="count(ancestor::entry) + 1"/>
 <xsl:variable name="h_element" select="concat('h', $depth)"/>
 <xsl:element name="{$h_element}">
  <xsl:attribute name="id">
   <xsl:value-of select="."/>
  </xsl:attribute>
  <xsl:apply-templates/>
 </xsl:element>
</xsl:template>

<xsl:template match="orth[not(@order='1')]"/>

<xsl:template match="etymology">
 <div class="etym">
  <xsl:apply-templates/>
 </div>
</xsl:template>
```

```
<xsl:template match="additional">
 <div class="additional">
  <xsl:apply-templates/>
 </div>
</xsl:template>

<xsl:template match="homograph">
 <xsl:call-template name="outputSenses"/>
</xsl:template>

<xsl:template name="outputHomographs">
 <xsl:choose>
  <xsl:when test="count(homograph) &gt; 1">
   <ol>
    <xsl:for-each select="homograph">
     <xsl:call-template name="homographToListItem"/>
    </xsl:for-each>
   </ol>
  </xsl:when>
  <xsl:otherwise>
   <xsl:apply-templates select="homograph"/>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>

<xsl:template name="homographToListItem">
 <li>
  <xsl:call-template name="outputSenses"/>
 </li>
</xsl:template>

<xsl:template match="description">
 <xsl:text> (</xsl:text>
 <xsl:apply-templates/>
 <xsl:text>)</xsl:text>
</xsl:template>

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

<xsl:template name="outputSenses">
 <xsl:choose>
  <xsl:when test="count(sense) &gt; 1">
   <ol type="a">
    <xsl:for-each select="sense">
     <xsl:call-template name="senseToListItem"/>
    </xsl:for-each>
   </ol>
  </xsl:when>
  <xsl:otherwise>
   <xsl:apply-templates/>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>

<xsl:template match="sense">
 <div class="sense">
```

```
   <xsl:apply-templates/>
  </div>
 </xsl:template>

 <xsl:template name="senseToListItem">
  <li>
   <xsl:apply-templates/>
  </li>
 </xsl:template>

 <xsl:template match="related">
  <xsl:variable name="linktype" select="@type"/>
  <div class="xref">
   <xsl:choose>
    <xsl:when test="$linktype='see'">
     <xsl:text>See </xsl:text>
    </xsl:when>
    <xsl:when test="$linktype='cf'">
     <xsl:text>Cf </xsl:text>
    </xsl:when>
    <xsl:when test="$linktype='related'">
     <xsl:text>Related term: </xsl:text>
    </xsl:when>
    <xsl:when test="$linktype='synonym'">
     <xsl:text>Synonym: </xsl:text>
    </xsl:when>
    <xsl:when test="$linktype='antonym'">
     <xsl:text>Antonym: </xsl:text>
    </xsl:when>
    <xsl:otherwise>
     <xsl:value-of select="."/>
    </xsl:otherwise>
   </xsl:choose>
   <a>
    <xsl:attribute name="href">
     <xsl:variable name="target" select="xref"/>
     <xsl:value-of select="concat('#', $target)"/>
    </xsl:attribute>
    <xsl:apply-templates/>
   </a>
  </div>
 </xsl:template>

</xsl:stylesheet>
```

Liite 5: XSLT-muunnosten käynnistyskomennot

Komento joka käynnistää XSLT-muunnoksen thesaurus-muodosta dictionary-muotoon:

```
java -jar saxon9.jar -o term_thesaurus2_result.xml
term_thesaurus2.xml DMV_to_dictionary.xsl narrowerTermEntry=yes
```

Parametri	**Merkitys**
saxon9.jar	XSLT-prosessori
term_thesaurus2_result.xml	kohdetiedosto
term_thesaurus2.xml	lähdetiedosto
DMV_to_dictionary.xsl	muunnoksessa käytettävä XSL-tyylitiedosto
narrowerTermEntry=yes	parametri, jonka toiminta on selostettu kohdassa 6.1.4.

Taulukko 3. DMV_to_dictionary.xsl -muunnoksen parametrit.

Komento joka käynnistää XSLT-muunnoksen dictionary-muodosta HTML-muotoon:

```
java -jar saxon9.jar -o dictionary.html thesaurus_result.xml
dictionary_to_HTML.xsl
```

Parametri	**Merkitys**
saxon9.jar	XSLT-prosessori
dictionary.html	kohdetiedosto
thesaurus_result.xml	lähdetiedosto
dictionary_to_HTML.xsl	muunnoksessa käytettävä XSL-tyylitiedosto

Taulukko 4. Dictionary_to_HTML.xsl-muunnoksen parametrit.